BSc (Hons) Computer Science

University of Portsmouth

Third Year

Distributed Systems and Security

M30225

Semester 2

Hugh Baldwin

hugh.baldwin@myport.ac.uk

Contents

Lecture - Introduction to DS&S

10:00 27/01/25 Amanda Peart

Module Info

Spit into two parts, Distributed Systems covered in the first 8 weeks and lectured by Amanda, Security covered in the last 4 weeks, lectured by Fahad.

- Distributed Systems covered by a 60% exam, Weds 26th March
- Security covered by a 40% exam

Lecture - Distributed Systems

10:00 27/01/25 Amanda Peart

What is a Distributed System?

A distributed system is the interactions between two or more computers, which are connected in some way to form a single computing system. The two main issues with distributed systems are how to connect the devices, and how to make them interact.

Definition

A distributed system is one which is made up of a set of independent computers which appear to the user as a single system. They are typically connected using a coordination system, which may use any of many networking and communications standards.

Networking Requirements

Most systems typically require fast and reliable networking to coordinate between systems. The often also require access to large amounts of data, which may needed to be accessed by any or all of the systems at the same time, so fast and reliable storage is also necessary.

Parts of a Distributed System

There are several levels to a distributed system, from what runs on each machine to the system as a whole. The low-level systems work on a per-machine level, and include processes, threads, concurrency, etc. Middleware acts to synchronise and coordinate the different parts of the system and ensure efficiency. The application itself sits on top of the Middleware and manages high-availability and fault tolerance.

Design Issues

Naming

It's important that every part of a distributed system has a useful and descriptive name which have a global meaning. This may be supported by a name interpretation or translation system to allow programs to access named resources.

Access

It's important to limit who and what can access the system, both in terms of security and accessibility. They should support as many systems as possible

Communication

The performance and reliability of the communication system is very important to both the availability and overall performance of the system as a whole.

Software

Data abstraction is very important as it allows many different systems to interact with the same data on a high-level without needing to convert data constantly between different formats. Part of this is well designed and documented APIs that allow access to the information and systems.

Resource Management

Optimisation of resources is very important to make sure that there is capacity and availability where it's needed. This includes load balancing and load shifting.

Consistency

The data must be consistent across the system, and must appear the same for all users.

Lecture - Distributed Shared Memory

09:00 17/02/25 Amanda Peart

Distributed Shared Memory (DSM) simplifies communication between nodes in distributed systems by providing a single memory space that all nodes have equal access to. It simplifies programming, as there is no need to perform explicit message passing between nodes or processes. It theoretically also reduces the overhead inherent in resource sharing, as it does not require the data to be duplicated on every node.

DSM is an abstraction which allows multiple computers which do not share any physical memory to access the same data as if it were a single unified address space. It is usually entirely transparent to the system running on it, as there is typically no need for the program to directly interact with the system implementing the abstraction.

Key Features

- **Transparency**– The user and the system do not need to know that the memory is distributed
- **Scalability**– Nodes should be able to be added and removed without affecting performance or reliability
- **Consistency Control**– It is essential to ensure that the memory is exactly the same on all nodes at the same time, to ensure that nodes are not using different or stale values
- **Synchronisation**– Using locks, semaphores or other synchronisation techniques to prevent conflicts between nodes
- **Performance**– Store data locally to ensure quick access

Centralised vs Distributed DSM

- Centralised
  - Shared memory managed by a single server
  - Data is accessed only through the single server
  - Data is stored in a single place
  - This makes it easier to manage and ensure consistency, but has a single point of failure and a large bottleneck on the performance of the single server
- Distributed
  - Memory is stored across multiple nodes but appears as one
  - Each node manages part of the memory space
  - This reduces the reliance on a single machine both in terms of reliability, and performance, but is significantly harder to manage and ensure consistency between nodes

DSM Models

- Page-Based
  - Transfers entire memory pages

  - Can be efficient for large datasets
  - Is susceptible to page faults causing large overhead
- Object-Based
  - Transfers single object instances
  - Reduced data transfers for each object
  - Harder to enforce consistency
- Variable-Based
  - Transfers single variable values
  - Ideal for sharing small amounts of data infrequently
  - Very bad in terms of scalability
- Library-Based
  - Run-time RPC/IPC
  - Very scalable if implemented correctly
  - Code may fail at runtime, causing unrecoverable errors

Consistency Models

Insert table here

Lecture - Web Services and Naming Systems

10:00 22/02/25 Amanda Peart

Web Service Architecture

- Provider
  - Creates and offers web services
  - Describes their services in a standardised format to be published in a registry
- Registry
  - Contains data about services offers by different providers
  - Usually includes technical details, as well as company info like addresses and contact details
- Consumer
  - Retrieves information about services from the registry
  - Uses information from the description to directly connect to the provider and actually use the service

Web Service Definition Language (WDSL)

WDSL is a standard method of describing the services offered by a web service. It tells the client what services it offers, how the client should connect to the service, and what the expected results should be. It specifies the protocols and endpoints used, as well as the location of the service.

Typically formatted as XML, especially when used with SOAP APIs, and tells the client exactly what the format of SOAP messages should be, both the request the client must make, and the response it should receive.

Simple Object Access Protocol

An API standard which defines how web services talk to each other, and how clients should invoke methods on the server. Typically uses XML as the serialized format which is exchanged between the client and server. Each message consists of an envelope that defines what is contained within a message and how to process it, and a set of standards for how the message is formatted. Most SOAP services use HTTP but it is not the only protocol that can be used.

Naming Systems

Types of Names

- **Human Readable Names**– Names used by humans, typically abstracted, e.g. URLs, domain names, hostnames, etc.
- **Identifiers**– Unique identifiers used within systems, e.g. IP addresses, MAC addresses, etc.
- **Addresses**– Internal addresses used by the application, e.g. Memory addresses, storage addresses, etc.

Naming Schemes

- **Flat Naming**– No structure, typically used in peer-to-peer networking
- **Hierarchical Naming**– Organised into a hierarchy like a tree, used in DNS, Active Directory, etc.
- **Attribute Based Naming**– Names based on the attributes of resources

Name Resolution

How names are mapped to an address or identifier. Can be centralized to a single server, or can be distributed hierarchically, as in DNS. Can also use caching to improve speed and efficiency, but can introduce consistency issues.

Challenges

The main challenges in naming schemes are scalability, consistency and security. This is because a naming system needs to be able to handle a large number of names, always be correct, and needs to prevent unauthorized access to name resolution and spoofing naming. There's also the issue of making a naming system that actually makes sense, since it may be based heavily on business-specific schemes, which may not be useful to anyone outside of the organization.

Lecture - Mobile and Ubiquitous Computing

10:00 03/03/25 Amanda Peart

TODO

Lecture - Fault Tolerance

10:00 17/03/25 Amanda Peart

Fault tolerance is the ability of a system to continue operating in spite of any failures. It is very important as failures are more or less inevitable, given the sheer complexity of many distributed systems.

Types of Failure

- Crashing
  - A node or the software running on it completely stops responding, for example if power fails to the node, or database software crashing in the middle of transactions.
- Omission
  - Messages are lost or dropped in transmission, for example network congestion causing transactions to take much longer than expected, or network devices failing and packets being lost.
- Timing
  - Part of the system responds too late or too early, for example a banking system not processing transactions in realtime, or transactions not being processed in chronological order leading to inconsistent results.
- Byzantine
  - A node sending incorrect data due to code errors or malicious attacks, for example a node in a blockchain network intentionally sending false transactions, or the software producing incorrect results due to a logic error.
- Network Partition
  - Some nodes become disconnected from the system as a whole, for example two datacenters losing communication, causing inconsistencies in the stored data.

Redundancy and Replication Techniques

- Data Replication
  - Primary-Backup model, where one server is the main server which handles writes, but there is one or more backup nodes which are ready to take over requests if the primary node fails
- Quorum-Based Redundancy
  - The majority of nodes must agree before an update or transaction can be completed
- Process Redundancy
  - Active Replication– All nodes run in parallel
  - Passive Replication– One node acts as a primary, all others act as passive backups
- Voting Mechanisms
  - Used in datastores like databases to ensure all nodes agree before committing a transaction

Recovery Mechanisms

- Checkpointing
  - Save the state of the entire system periodically, so only data modified after that point is lost in the event of a failure
- Rollback
  - Restoring the state of the system to a checkpoint after a failure
- Message Logging
  - Store messages before they are processed so they can be re-processed in the event of a crash or other failure
- Self-healing
  - Using algorithms or machine learning to monitor the state of the system and predict if system is likely to fail imminently, and either resolve automatically or page administrators

Consensus Algorithms

- Paxos Algorithm
- Raft Algorithm
- Byzantine Fault Tolerance

(To be filled in after seminar)

Fault Detection Mechanisms

- Heartbeats
  - Nodes periodically send a ‘heartbeat’, effectively just a message saying they’re still online and working as expected
- Timeouts
  - If responses are taking longer than expected or aren’t being received at all, the node is marked as failed
- Gossip Protocols
  - Nodes spread information about failures throughout the system using peer-to-peer communication