DC UNIT 1

A distributed system is a collection of independent computers that communicate and coordinate their actions by passing messages. The computers in a distributed system can be physically located anywhere in the world, and they can be of different types and sizes.

Distributed systems are used in a wide variety of applications, including:

- The World Wide Web
- Email
- Social media
- Online banking
- E-commerce
- Cloud computing
- Distributed databases
- High-performance computing

Distributed systems have a number of advantages over traditional centralized systems, including:

- Scalability: Distributed systems can be scaled up or down to meet the demands of the application.
- Reliability: Distributed systems are more reliable than centralized systems because they are not dependent on a single point of failure.
- Performance: Distributed systems can often outperform centralized systems by distributing the workload across multiple computers.

However, distributed systems also have some challenges, including:

- Complexity: Distributed systems are more complex to design and implement than centralized systems.
- Concurrency: Distributed systems must be able to handle concurrent access to data and resources.
- Fault tolerance: Distributed systems must be able to withstand the failure of individual components.

Despite the challenges, distributed systems are becoming increasingly important as the world becomes more interconnected and data-driven.

Here are some basic concepts of distributed systems:

- Concurrency: Distributed systems must be able to handle concurrent access to data and resources. This means that the system must be able to ensure that data is consistent and that resources are not overallocated.
- Fault tolerance: Distributed systems must be able to withstand the failure of individual components. This means that the system must be able to continue to operate even if some of its components fail.
- Communication: Distributed systems must be able to communicate with each other to coordinate their activities. This communication can be done over a variety of networks, including the Internet, local area networks (LANs), and wide area networks (WANs).
- Transparency: Distributed systems should be transparent to users. This means that users should not be aware that they are interacting with a distributed system.

Distributed systems are a complex topic, but the basic concepts are relatively straightforward. By understanding the basic concepts, you can gain a better understanding of how distributed systems work and the challenges they face.

Characterization of Distributed Systems:

Distributed systems are characterized by the following:

- Concurrency: Multiple processes can access and modify shared data and resources at the same time.

- Distribution: The components of the system are physically distributed across multiple computers.
- Heterogeneity: The components of the system may be of different types and sizes, and may run different operating systems and software.
- Autonomy: The components of the system are autonomous, meaning that they can operate independently of each other.

Issues in Distributed Systems:

Due to their distributed nature, distributed systems face a number of challenges, including:

- Concurrency control: The system must ensure that concurrent access to data and resources does not lead to inconsistencies or corruption.
- Fault tolerance: The system must be able to withstand the failure of individual components without compromising its overall functionality.
- Transparency: The system should be transparent to users, meaning that they should not be aware that they are interacting with a distributed system.
- Security: The system must be secure against unauthorized access and malicious attacks.

Goals of Distributed Systems:

The primary goals of distributed systems are to:

- Improve performance: By distributing the workload across multiple computers, distributed systems can often outperform centralized systems.
- Increase scalability: Distributed systems can be scaled up or down to meet the demands of the application.
- Improve reliability: Distributed systems are more reliable than centralized systems because they are not dependent on a single point of failure.
- Provide access to shared resources: Distributed systems allow users to access shared resources, such as files, databases, and printers, from anywhere in the world.

Examples of Distributed Systems:

Some common examples of distributed systems include:

- The World Wide Web
- Email
- Social media
- Online banking
- E-commerce
- Cloud computing
- Distributed databases
- High-performance computing

Distributed systems are an essential part of modern computing, and they play a vital role in many different industries. By understanding the characteristics, issues, and goals of distributed systems, you can gain a better appreciation of the challenges and complexities involved in designing and implementing them.

Types of distributed systems

There are many different ways to classify distributed systems, but one common way is by their architecture. Some of the most common types of distributed systems include:

- Client-server systems: Client-server systems are the simplest and most common type of distributed system. In a client-server system, there are two types of nodes: clients and servers. Clients send requests to servers, and servers process the requests and return the results to the clients.
- Peer-to-peer (P2P) systems: P2P systems are a type of distributed system in which all nodes are equal. In a P2P system, each node can be both a client and a server. P2P systems are often used for file sharing and distributed computing.
- Clustered systems: Clustered systems are a type of distributed system that consists of a group of interconnected computers that act as a single system. Clustered systems are often used for high-performance computing and mission-critical applications.
- Grid systems: Grid systems are a type of distributed system that consists of a large number of geographically dispersed computers that are connected by a

network. Grid systems are often used for scientific computing and other applications that require a lot of processing power.
- Cloud systems: Cloud systems are a type of distributed system that provides computing resources on demand over the Internet. Cloud systems are often used for web applications, data storage, and other business applications.

In addition to these architectural types, distributed systems can also be classified by their purpose or application. For example, there are distributed systems for database management, distributed systems for real-time processing, and distributed systems for high availability.

Here are some examples of specific distributed systems:

- The World Wide Web is a client-server distributed system.
- The BitTorrent file-sharing network is a P2P distributed system.
- The Google search engine is a clustered distributed system.
- The SETI@home project uses a grid distributed system to search for extraterrestrial intelligence.
- The Amazon Web Services cloud platform is a cloud distributed system.

Distributed systems are a complex and diverse topic, but the basic concepts are relatively straightforward. By understanding the different types of distributed systems and their characteristics, you can gain a better appreciation of the challenges and complexities involved in designing and implementing them.

Distributed System Models

Distributed system models are abstract representations of the different components and interactions of a distributed system. They can be used to design and analyze distributed systems, and to identify potential problems and solutions.

Some of the most common distributed system models include:

- Architectural models: Architectural models describe the overall design and structure of a distributed system, and how its different components are organized to interact with each other and provide the desired functionalities.
- Interaction models: Interaction models describe how the different components of a distributed system communicate with each other. They typically include information about the protocols used, the types of messages exchanged, and the sequence of messages exchanged.
- Fault models: Fault models describe the different types of failures that can occur in a distributed system, and how the system can respond to these failures.

Here are some examples of distributed system models:

- The client-server model is an architectural model that divides a distributed system into two types of components: clients and servers. Clients send requests to servers, and servers process the requests and return the results to the clients.
- The request-reply model is an interaction model in which a client sends a request to a server, and the server sends a reply back to the client.
- The message-passing model is an interaction model in which the different components of a distributed system communicate with each other by sending and receiving messages.
- The fail-stop model is a fault model in which a failed component simply stops responding to requests.
- The Byzantine fault model is a fault model in which a failed component may behave arbitrarily, including sending incorrect or malicious messages.

Distributed system models can be used to design and analyze distributed systems, and to identify potential problems and solutions. For example, a designer can use a distributed system model to identify potential concurrency problems, and to develop mechanisms to address these problems. A designer can also use a distributed system model to analyze the fault tolerance of a system, and to identify potential single points of failure.

Distributed system models are a powerful tool for designing and analyzing distributed systems. By understanding the different types of distributed system models and their

characteristics, you can gain a better appreciation of the challenges and complexities involved in designing and implementing them.

Distributed systems can be categorized into several models based on their architectural and communication patterns. These models help define how components in the system interact and cooperate. Here are some common distributed system models:

1. Client-Server Model:

- In this model, there are two main roles: clients and servers.

- Clients make requests for services or resources, and servers fulfill those requests.

- It can have two tiers (client and server) or three tiers (client, application server, and data server) depending on the complexity of the system.

2. Peer-to-Peer (P2P) Model:

- In P2P systems, there is no clear distinction between clients and servers; all participants are considered peers.

- Peers both request and provide services or resources to each other.

- Examples include file-sharing networks like BitTorrent and some decentralized cryptocurrency networks like Bitcoin.

3. Middleware Model:

- Middleware acts as an intermediary layer that abstracts and simplifies communication and coordination between distributed components.

- Common middleware types include message-oriented middleware (MOM), remote procedure call (RPC) middleware, and object request brokers (ORBs).

4. Message-Passing Model:

- In this model, components communicate by sending messages to each other.

- It's often used in distributed systems where communication between nodes is not tightly coupled, and asynchronous communication is preferred.

5. Hybrid Models:

- Many distributed systems combine elements from multiple models to meet specific requirements. For example, a system might use microservices for scalability and SOA for interoperability.

Hardware concepts

Hardware concepts are the fundamental principles and technologies that underlie the physical components of a computer system. This includes everything from the central processing unit (CPU) and memory to the input and output devices.

Some of the most important hardware concepts include:

- CPU: The CPU is the brain of the computer. It is responsible for processing all of the instructions that are given to the computer.
- Memory: Memory is used to store data and instructions that the CPU needs to access. There are two main types of memory: primary memory (RAM) and secondary memory (hard drives, SSDs, etc.).
- Input and output devices: Input and output devices are used to interact with the computer. Input devices, such as keyboards and mice, allow users to provide input to the computer. Output devices, such as monitors and printers, allow users to view and print the results of their computations.
- Buses: Buses are used to connect the different components of a computer system together. They allow the CPU, memory, and input/output devices to communicate with each other.
- Storage devices: Storage devices are used to store data and programs permanently. Common storage devices include hard drives, solid-state drives (SSDs), and optical drives.

In addition to these basic concepts, there are a number of other hardware concepts that are important to understand, such as:

- Operating systems: Operating systems are software that manages the hardware resources of a computer system. They provide a platform for applications to run on, and they handle tasks such as memory management, file management, and device management.
- Networking: Networking is the process of connecting two or more computers together so that they can share resources and communicate with each other. There are a variety of networking technologies available, including Ethernet, Wi-Fi, and Bluetooth.
- Cloud computing: Cloud computing is a model of computing in which resources, such as storage and processing power, are provided over the internet. Cloud computing can be used to reduce costs and improve scalability.

Hardware concepts are essential for understanding how computers work. By understanding the basic hardware concepts, you can gain a better appreciation of the capabilities and limitations of computer systems.

Here are some examples of hardware concepts in action:

- When you open a web page, the CPU retrieves the necessary files from the hard drive and loads them into memory. The CPU then processes the HTML, CSS, and JavaScript code to render the page on the monitor.
- When you play a video game, the CPU decompresses the video data and sends it to the graphics card for rendering. The graphics card then outputs the rendered video to the monitor.
- When you save a document, the CPU compresses the document data and writes it to the hard drive.

Hardware concepts are an essential part of computer science, and they are used in a wide variety of applications. By understanding the basic hardware concepts, you can gain a better understanding of how computers work and how to use them effectively.

Software Concept

Software concepts are the basic principles and ideas that underlie the development and use of software. These concepts include things like algorithms, data structures, programming languages, and software engineering principles.

Some of the most important software concepts include:

- Algorithms: Algorithms are step-by-step procedures for solving problems. They are used in all aspects of software development, from designing the overall structure of a program to implementing specific tasks.
- Data structures: Data structures are ways of organizing data so that it can be efficiently accessed and processed. Common data structures include arrays, linked lists, and hash tables.
- Programming languages: Programming languages are used to write instructions for computers to follow. They are the primary tool used by software developers to create software.
- Software engineering principles: Software engineering principles are guidelines for developing and maintaining software in a systematic and efficient way. These principles include things like modularity, abstraction, and testing.

In addition to these basic concepts, there are a number of other software concepts that are important to understand, such as:

- Object-oriented programming: Object-oriented programming is a programming paradigm that is based on the concept of objects. Objects are self-contained entities that have data and behavior. Object-oriented programming is one of the most popular programming paradigms in use today.
- Databases: Databases are used to store and organize data. They are used in a wide variety of applications, including web applications, e-commerce systems, and accounting systems.
- Networks: Networks are used to connect computers together so that they can share resources and communicate with each other. Networking is essential for many software applications, such as web browsers, email clients, and file-sharing programs.

Software concepts are essential for understanding how software works and how to develop software effectively. By understanding the basic software concepts, you can gain a better appreciation of the challenges and complexities involved in software development.

Here are some examples of software concepts in action:

- When you search for a word in a word processor, the word processor uses a data structure called a hash table to quickly find the word in the document.
- When you play a web game, the game server uses algorithms to calculate the movements of the game objects and to detect collisions.
- When you upload a photo to a social media website, the website uses a database to store the photo and to track who has viewed the photo.

Software concepts are used in a wide variety of applications, and they are essential for understanding how software works and how to develop software effectively. By understanding the basic software concepts, you can gain a better appreciation of the capabilities and limitations of software systems.

Middleware: Models of Middleware

Middleware is software that acts as a layer between applications and operating systems or networks. It provides services that applications can use to communicate with each other, access data, and perform other tasks without having to implement those services themselves.

There are many different models of middleware, but some of the most common include:

- Remote Procedure Call (RPC): RPC middleware allows applications to call procedures on remote servers as if they were local procedures. This is done by marshaling the parameters of the call into a message, sending the message to the server, unmarshalling the parameters on the server, executing the procedure, and marshalling the results of the procedure back into a message and sending it back to the client. RPC middleware is a very popular model of middleware, and it

is used in a wide variety of applications, including web services, distributed databases, and distributed file systems.

- Message Queuing (MQ): MQ middleware allows applications to send and receive messages to each other asynchronously. This is done by placing the messages in a queue, which is a persistent store of messages. Applications can then read messages from the queue at their own pace. MQ middleware is a very versatile model of middleware, and it can be used for a wide variety of tasks, such as decoupling applications, buffering messages, and load balancing.
- Object Request Broker (ORB): ORB middleware allows applications to communicate with each other using objects. This is done by translating the object references into network addresses and marshalling the parameters of the calls into messages. ORB middleware is often used in distributed object-oriented systems, such as enterprise resource planning (ERP) systems and customer relationship management (CRM) systems.
- Transaction Processing Monitor (TPM): TPM middleware provides services for managing transactions. Transactions are sequences of operations that must be completed successfully or not at all. TPM middleware ensures that transactions are completed atomically, consistently, isolatedly, and durably (ACID). TPM middleware is often used in financial systems and other mission-critical applications.

These are just a few of the many different models of middleware. The best model of middleware for a particular application will depend on the specific requirements of the application.

Here are some examples of middleware in action:

- When you use a web browser to access a website, the web browser uses RPC middleware to call procedures on the web server.
- When you use an email client to send and receive email, the email client uses MQ middleware to send and receive email messages.
- When you use an ATM machine to withdraw money, the ATM machine uses ORB middleware to communicate with the bank's back-end systems.

- When you make a purchase online, the e-commerce website uses TPM middleware to manage the transaction.

Middleware is an essential part of many modern software systems. It provides services that make it easier to develop and deploy distributed applications. By understanding the different models of middleware, you can gain a better appreciation of the challenges and complexities involved in distributed systems development.

Services offered by middleware

Middleware offers a wide range of services to applications, including:

- Communication: Middleware provides services that allow applications to communicate with each other, regardless of their programming language or operating system. This can be done using a variety of protocols, such as RPC, MQ, and ORB.
- Data access: Middleware provides services that allow applications to access data from a variety of sources, such as databases, file systems, and web services. This can be done in a transparent way, so that applications do not need to know the details of the underlying data source.
- Security: Middleware can provide services to help applications protect their data and resources from unauthorized access. This can be done using a variety of techniques, such as authentication, authorization, and encryption.
- Transaction management: Middleware can provide services to help applications manage transactions. Transactions are sequences of operations that must be completed successfully or not at all. Middleware can ensure that transactions are completed atomically, consistently, isolatedly, and durably (ACID).
- Load balancing: Middleware can distribute workload across multiple servers to improve performance and reliability. This can be done using a variety of algorithms, such as round robin and least connections.
- Routing: Middleware can route messages between applications based on criteria such as the destination address, the type of message, or the priority of the message.
- Caching: Middleware can cache data to improve performance by reducing the number of times that applications need to access the underlying data source.

- Monitoring: Middleware can monitor applications and their resources to detect and troubleshoot problems. This can be done using a variety of techniques, such as logging, tracing, and alerting.

In addition to these basic services, middleware can also provide a wide range of other services, such as:

- Workflow management: Middleware can provide services to help applications manage workflows. Workflows are sequences of tasks that must be completed in a specific order. Middleware can ensure that workflows are executed correctly and efficiently.
- Business process management (BPM): Middleware can provide services to help businesses automate their business processes. BPM is the discipline of managing and improving business processes. Middleware can help businesses to streamline their processes, reduce costs, and improve efficiency.
- Event processing: Middleware can provide services to help applications process events. Events are occurrences that happen at a specific time and place. Middleware can help applications to detect, filter, and respond to events.

Middleware is an essential part of many modern software systems. It provides services that make it easier to develop and deploy distributed applications. By understanding the different services offered by middleware, you can gain a better appreciation of the challenges and complexities involved in distributed systems development.

Client Server model

The client-server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs, which share their resources with clients. A client usually does not share any of its resources, but it requests content or service from a server.

In a client-server model, the client is responsible for generating requests and sending them to the server. The server is responsible for processing the requests and sending

the responses back to the client. The client and server can communicate using a variety of protocols, such as HTTP, FTP, and SMTP.

The client-server model has a number of advantages over other distributed application models, including:

- Scalability: The client-server model is very scalable. This means that it can be easily scaled up or down to meet the needs of a growing or shrinking user base.
- Reliability: The client-server model is more reliable than other distributed application models. This is because the server is responsible for managing the state of the application. If the client fails, the server can continue to operate and serve other clients.
- Security: The client-server model can be more secure than other distributed application models. This is because the server can be centrally secured and monitored.

The client-server model is used in a wide variety of applications, including:

- Web applications: Web applications are a common example of client-server applications. In a web application, the client is typically a web browser and the server is a web server. The web browser sends requests to the web server for web pages and other resources. The web server processes the requests and sends the responses back to the web browser.
- Email: Email is another common example of a client-server application. In an email system, the client is typically an email client and the server is an email server. The email client sends requests to the email server to send and receive email messages. The email server processes the requests and sends the responses back to the email client.
- File sharing: File sharing applications are also often client-server applications. In a file sharing application, the client is typically a file sharing client and the server is a file sharing server. The file sharing client sends requests to the file sharing server to upload and download files. The file sharing server processes the requests and sends the responses back to the file sharing client.

The client-server model is a very versatile and powerful distributed application model. It is used in a wide variety of applications, both large and small.

Key Characteristics of the Client-Server Model:

1. Request-Response Model:

- In the client-server model, clients initiate requests to servers, and servers respond with the requested information or services.

- This request-response interaction is the core of the model, allowing clients to access server resources.

2. Separation of Concerns:

- The client-server model enforces a separation of concerns between clients and servers. Clients focus on user interfaces and user interactions, while servers handle data processing and management.

- This separation simplifies application development and maintenance.

3. Scalability:

- The model allows for scalability by adding more clients or servers as needed.

- Servers can handle multiple client requests concurrently, allowing systems to handle large workloads.

4. Centralized vs. Decentralized:

- In a centralized client-server model, there is a single server or a set of central servers that provide services to multiple clients.

- In decentralized scenarios, multiple servers collaborate to provide services, often referred to as distributed systems.

5. Stateless vs. Stateful:

- Some client-server interactions are stateless, meaning each request from the client includes all the information needed for the server to process it.

- Others are stateful, where the server maintains information about the client's state between requests.

Examples of the Client-Server Model:

1. Web Browsing: When you use a web browser to access a website, your browser acts as the client, and the web server hosting the site responds to your requests by sending HTML pages, images, and other resources.

2. Email: Email clients (e.g., Microsoft Outlook, Gmail) communicate with email servers to send and receive messages. The email server stores and manages email messages.

3. File Sharing: In a file-sharing scenario, clients request files or data from file servers, which respond by providing the requested files.

The client-server model is a versatile and widely adopted architecture that enables the development of scalable, distributed, and efficient systems for a wide range of applications and services. It forms the basis for much of today's computing infrastructure and internet services.