Unit-4

The Common Object Request Broker Architecture (CORBA)

The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

CORBA is a standard for distributing objects across networks so that operations on those objects can be called remotely. CORBA is not associated with a particular programming language, and any language with a CORBA binding can be used to call and implement CORBA objects. Objects are described in a syntax called Interface Definition Language (IDL).

CORBA includes four components:

Object Request Broker (ORB)

The Object Request Broker (ORB) handles the communication, marshaling, and unmarshaling of parameters so that the parameter handling is transparent for a CORBA server and client applications.

CORBA server

The CORBA server creates CORBA objects and initializes them with an ORB. The server places references to the CORBA objects inside a naming service so that clients can access them.

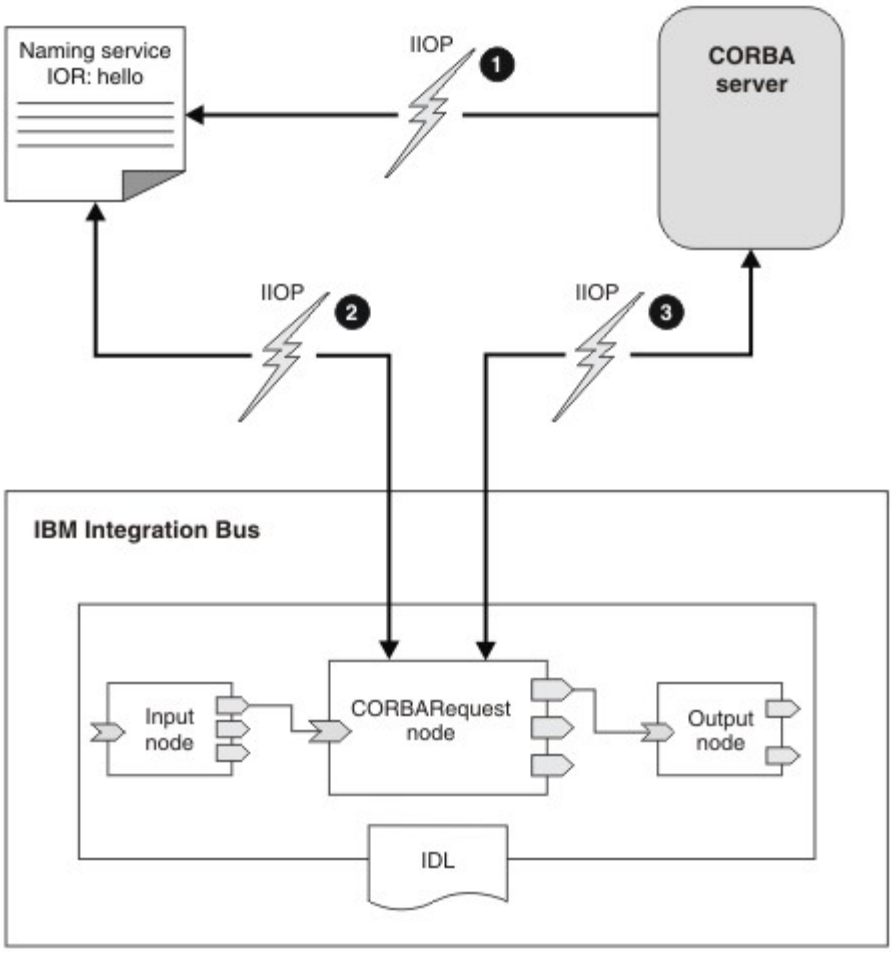
Naming service

The naming service holds references to CORBA objects.

CORBARequest node

The CORBARequest node acts as a CORBA client.

The following diagram shows the layers of communication between IBM® Integration Bus and CORBA.



The diagram illustrates the following steps.

1. CORBA server applications create CORBA objects and put object references in a naming service so that clients can call them.
2. At deployment time, the node contacts a naming service to get an object reference.
3. When a message arrives, the node uses the object reference to call an operation on an object in the CORBA server.

Distributed Component Object Model (DCOM)

DCOM stands for Distributed Component Object Model. It is a Microsoft technology that enables software components to communicate with each other in a networked environment. DCOM extends the Component Object Model (COM), which is a platform-independent, distributed, object-oriented system for creating binary software components that can interact.

DCOM allows software components to communicate across a network in a distributed environment. This communication can occur between objects on the same machine or between objects on different machines connected by a network. DCOM uses a client-server model where a client application can request services from a server application.

.

Key features and concepts of DCOM include:

1. **Object-Oriented Communication:** DCOM allows objects (software components) to communicate with each other regardless of their physical location in a network.
2. **Location Transparency:** DCOM provides a level of abstraction that allows clients to interact with objects without needing to know their physical location or details about the communication mechanism.
3. **Security:** DCOM includes security features to ensure that communication between components is secure. This includes authentication and authorization mechanisms.
4. **Interface Definition Language (IDL):** DCOM uses Interface Definition Language to define the interfaces that components expose to the network. This ensures a standardized way for components to communicate.
5. **Marshaling:** DCOM includes marshaling mechanisms to convert data between different formats, making it possible for components written in different programming languages to communicate seamlessly.
6. **Registry Services:** DCOM relies on the Windows Registry to store information about available components and their locations. This allows clients to discover and connect to the desired components.

Advantages of DCOM:

1. **Interoperability:**
   - **Advantage:** DCOM facilitates interoperability by allowing software components written in different programming languages to communicate seamlessly.
2. **Component Reusability:**
   - **Advantage:** DCOM promotes component-based development, making it easier to reuse and integrate existing software components in various applications.
3. **Location Transparency:**
   - **Advantage:** DCOM abstracts the physical location of components, providing location transparency. Clients can access objects without needing to know where they are located in the network.
4. **Object-Oriented Communication:**
   - **Advantage:** DCOM supports object-oriented communication, allowing for the creation and interaction of objects across a distributed environment.
5. **Security Features:**

- **Advantage:** DCOM includes security mechanisms, such as authentication and authorization, to ensure secure communication between components.

6. **Integration with Windows Infrastructure:**
    - **Advantage:** DCOM is well-integrated with the Windows operating system, making it suitable for building applications within the Windows ecosystem.

Disadvantages of DCOM:

1. **Complexity:**
    - **Disadvantage:** DCOM can be complex to configure and manage, especially in large and diverse network environments. Configuring security settings, managing the Windows Registry, and dealing with DCOM permissions can be challenging.
2. **Platform Dependency:**
    - **Disadvantage:** DCOM is primarily designed for Windows environments. While efforts have been made to provide interoperability with other platforms, it may not be as seamless as other cross-platform solutions.
3. **Firewall Challenges:**
    - **Disadvantage:** DCOM communication may face challenges when crossing firewalls, as it may require specific port configurations and settings, potentially leading to security concerns.
4. **Performance Overhead:**
    - **Disadvantage:** DCOM can introduce performance overhead due to its complexity, especially in comparison to lighter-weight communication protocols used in more modern distributed systems.
5. **Versioning Issues:**
    - **Disadvantage:** Handling versioning and compatibility between different versions of DCOM components can be challenging and may require careful management.
6. **Limited Adoption in Modern Architectures:**
    - **Disadvantage:** With the rise of web services, RESTful APIs, and other lightweight communication protocols, DCOM has become less prevalent in modern distributed system architectures.

Distributed File System (DFS)

A Distributed File System (DFS) is a file system that allows multiple users and applications to access and manage files and data stored on multiple servers as if it were a single, centralized file system. The goal of a distributed file system is to provide a

unified and coherent view of files and data across a network, often spanning multiple physical locations.

Key characteristics and concepts of distributed file systems include:

1. **Scalability:** Distributed file systems are designed to scale horizontally, allowing the addition of more storage capacity and servers to accommodate growing data requirements.
2. **Fault Tolerance:** Many distributed file systems incorporate fault-tolerance mechanisms to ensure data availability even if some components or servers fail. This often involves data replication or distribution across multiple nodes.
3. **Transparency:** Users and applications accessing files through a distributed file system may not be aware of the underlying physical storage locations or the complexities of the distributed architecture. This transparency simplifies the user experience.
4. **Concurrency Control:** Distributed file systems often provide mechanisms for controlling access to files concurrently, ensuring that multiple users or applications can read and write to files without conflicts.
5. **Caching:** To improve performance, distributed file systems may employ caching strategies to store frequently accessed data locally on client machines or within the distributed network.
6. **Security:** Security features such as authentication, authorization, and encryption are important aspects of distributed file systems to protect data from unauthorized access.

Popular examples of distributed file systems include:

- **Hadoop Distributed File System (HDFS):** Designed for use with the Apache Hadoop framework, HDFS is a distributed file system that provides high-throughput access to application data.
- **Google File System (GFS):** Developed by Google, GFS is a distributed file system designed for large-scale data-intensive applications.
- **Microsoft Distributed File System (DFS):** A component of the Microsoft Windows operating system that enables users to access and manage files that are physically distributed across a network.

Advantages and Disadvantages of Distributed File System

Advantages

There are various advantages of the distributed file system. Some of the advantages are as follows:

1. It allows the users to access and store the data.
2. It helps to improve the access time, network efficiency, and availability of files.
3. It provides the transparency of data even if the server of disk files.
4. It permits the data to be shared remotely.
5. It helps to enhance the ability to change the amount of data and exchange data.

Disadvantages

There are various disadvantages of the distributed file system. Some of the disadvantages are as follows:

1. In a DFS, the database connection is complicated.
2. In a DFS, database handling is also more complex than in a single-user system.
3. If all nodes try to transfer data simultaneously, there is a chance that overloading will happen.
4. There is a possibility that messages and data would be missed in the network while moving from one node to another.

Goal and design issue of distributed file system:

The goal of a distributed file system (DFS) is to provide a unified and coherent view of files and data across a network, even though the underlying storage may be distributed across multiple servers. Several key goals and design issues are considered in the development of distributed file systems:

Goals:

1. **Scalability:**
    - **Goal:** The system should scale horizontally to accommodate increasing amounts of data and user access.

- **Design Consideration:** Distributed file systems should be able to add new servers or storage nodes seamlessly to handle growing storage requirements.

2. **Fault Tolerance:**
   - **Goal:** Ensure high availability and reliability by handling node failures gracefully.
   - **Design Consideration:** Implement mechanisms such as data replication or erasure coding to protect against data loss in the event of server failures.
3. **Transparency:**
   - **Goal:** Provide a transparent and unified view of the file system to users and applications, regardless of the physical distribution of data.
   - **Design Consideration:** Abstract away complexities of distributed storage, making it appear as a single, centralized file system to users.
4. **Concurrency Control:**
   - **Goal:** Allow multiple users or applications to access and modify files concurrently without conflicts.
   - **Design Consideration:** Implement robust concurrency control mechanisms to handle simultaneous read and write operations.
5. **Performance:**
   - **Goal:** Optimize data access and retrieval for high performance.
   - **Design Consideration:** Employ caching strategies, distributed data placement, and efficient data transfer protocols to enhance system performance.
6. **Security:**
   - **Goal:** Ensure data integrity, confidentiality, and access control.
   - **Design Consideration:** Implement encryption, authentication, and authorization mechanisms to secure data and control access to files.

Design Issues:

1. **Consistency Model:**
   - **Design Issue:** Choosing between strong consistency, eventual consistency, or a consistency model that balances trade-offs based on application requirements.
2. **Data Distribution:**
   - **Design Issue:** Deciding how to distribute data across multiple nodes, whether through replication, sharding, or other strategies.
3. **Metadata Management:**
   - **Design Issue:** Managing metadata efficiently, including file attributes, directory structures, and access control information.

4. **Concurrency Control Mechanisms:**
    - **Design Issue:** Implementing methods to handle concurrent access to files, ensuring data consistency without sacrificing performance.
5. **Failure Handling:**
    - **Design Issue:** Developing mechanisms to detect and recover from node failures, minimizing downtime and data loss.
6. **Network Topology:**
    - **Design Issue:** Considering the network architecture and topology to optimize data transfer and minimize latency.
7. **Security Protocols:**
    - **Design Issue:** Defining and implementing security protocols to protect data during transmission and storage.
8. **Scalability Strategies:**
    - **Design Issue:** Planning for scalable architectures, including load balancing and the ability to add or remove nodes dynamically.

Type of distributed file system:

Distributed file systems come in various types, each designed to meet specific requirements and use cases. Here are some common types of distributed file systems:

1. **Distributed File System (DFS):**
    - **Description:** A general term for file systems that distribute storage and retrieval tasks across multiple servers.
    - **Example:** Microsoft Distributed File System (DFS)
2. **Network File System (NFS):**
    - **Description:** A distributed file system protocol that allows a user on a client computer to access files over a network in a manner similar to how local storage is accessed.
    - **Example:** NFSv4 (Network File System version 4)
3. **Andrew File System (AFS):**
    - **Description:** A distributed file system that enables users to access and share files seamlessly across a network.
    - **Example:** OpenAFS
4. **Hadoop Distributed File System (HDFS):**
    - **Description:** Specifically designed for storing and managing large datasets used in big data processing. Part of the Apache Hadoop framework.
    - **Example:** Hadoop Distributed File System (HDFS)
5. **Google File System (GFS):**

- **Description:** A distributed file system developed by Google to handle large-scale data processing tasks.
- **Example:** Google Cloud Storage

6. **Ceph File System (CephFS):**
    - **Description:** Part of the Ceph distributed storage system, CephFS provides a POSIX-compliant file system on top of Ceph's object storage system.
    - **Example:** Ceph
7. **GlusterFS:**
    - **Description:** An open-source, scalable network file system that clusters together storage resources from multiple servers.
    - **Example:** GlusterFS
8. **Lustre File System:**
    - **Description:** A parallel distributed file system used for high-performance computing (HPC) environments.
    - **Example:** Lustre
9. **Amazon S3 (Simple Storage Service):**
    - **Description:** While not a traditional file system, it's a widely used object storage service that can be considered a distributed storage system.
    - **Example:** Amazon S3
10. **Windows Distributed File System (DFS):**

- **Description:** A Microsoft technology that allows users to access and manage files that are physically distributed across a network.
- **Example:** Microsoft Distributed File System (DFS)

The Sun Network File System (NFS)

The Sun Network File System (NFS) is a distributed file system protocol that allows a user on a client computer to access files over a network in a manner similar to how local storage is accessed. It was originally developed by Sun Microsystems in the 1980s and has since become a widely adopted standard for sharing files and directories among Unix and Linux systems.

Key features and aspects of NFS include:

1. **Client-Server Architecture:**
    - NFS operates on a client-server model where the server hosts the file system, and clients can mount and access remote directories as if they were local.
2. **Stateless Protocol:**

- NFS is considered a stateless protocol, meaning each request from the client to the server is independent, and the server does not need to retain information about the client's state between requests.

3. **Versioning:**
   - NFS has gone through several versions, with NFSv4 being the latest major version. Each version introduces improvements and enhancements, including increased security features.
4. **Mounting:**
   - Clients can "mount" remote directories from an NFS server, making them appear as part of the client's local file system.
5. **File Locking:**
   - NFS supports file locking mechanisms to coordinate access to files among multiple clients, ensuring data consistency.
6. **Security:**
   - NFS traditionally lacked strong security features, but later versions (especially NFSv4) have introduced improvements in terms of authentication and encryption.
7. **Cross-Platform Compatibility:**
   - NFS is designed to be platform-independent and can be used to share files between different operating systems, including Unix, Linux, and some versions of Windows.
8. **Performance:**
   - NFS is optimized for efficiency, but performance can be influenced by factors such as network speed, latency, and server load.
9. **Use Cases:**
   - NFS is commonly used for sharing files and resources in Unix and Linux environments. It is often employed in scenarios where multiple systems need to access a shared set of files, such as in corporate networks.
10. **Standardization:**

- NFS is an open standard, and its specifications are documented in Request for Comments (RFC) documents. This has contributed to its widespread adoption and interoperability.