1) What determines the openness of distributed systems?

Definition: Openness in distributed systems refers to the ease with which they can be extended, modified, and integrated with new components and services.

Open systems are built using standardized interfaces and protocols, allowing components from different vendors or developers to collaborate seamlessly.

Example: Imagine a distributed email system built on open protocols like SMTP and POP3.

This allows different email clients and servers to communicate and exchange messages, regardless of their specific software implementation.

Advantage: Adapting to changing needs and integrating new technologies is easier.

Users can choose from various vendors and solutions.

Fosters a diverse ecosystem of developers and vendors.

Challenges: Openness can introduce vulnerabilities if not implemented carefully.

Managing various components and maintaining compatibility can be intricate.

2) Define Transparency. What are its types?

Definition: Transparency in distributed systems refers to the concealment of the underlying complexity and distribution of the system from the user's perspective.

It aims to make the system appear as a single, unified entity, regardless of its geographically dispersed components and operational intricacies.

Types of Transparency:

Access Transparency: Users are unaware of the location of data or resources they access. Accessing remote resources feels no different than accessing local ones.

Location Transparency: Users don't need to know where processes or services are physically located.

Concurrency Transparency: Users are unaware of the concurrent execution of multiple processes or tasks.

Replication Transparency: Users are unaware of data replication across multiple locations. The system ensures data consistency and availability transparently.

Failure Transparency: Users are unaware of system failures or recoveries. The system masks failures and continues operation transparently.

Mobility Transparency: Users are unaware of changes in resource mobility (e.g., migration of processes or services).

Performance Transparency: Users are unaware of variations in performance across different distributed components.

Scaling Transparency: Users are unaware of changes in system scale (e.g., addition or removal of resources). The system scales seamlessly without impacting user experience.

3) What are the difficulties and threats for distributed systems?

Difficulties:

Complexity: Managing and debugging distributed systems is often difficult due to the geographically dispersed components.

Consistency: Maintaining data consistency across multiple nodes can be challenging, especially when facing network partitions, concurrent updates, or failures.

Scalability: Balancing load, ensuring consistency, and managing communication overhead becomes increasingly complex as the system grows.

Security: Distributed systems present a larger attack surface due to their wider communication channels and diverse components.

Fault Tolerance: This requires mechanisms to detect, and recover from failures in individual components without compromising overall system functionality and data integrity.

Threats:

Cyberattacks: Distributed systems are vulnerable to various cyberattacks such as denial-of-service (DoS) attacks, man-in-the-middle attacks, and data breaches.

Physical threats: Hardware failures, natural disasters, or power outages can disrupt distributed systems, leading to data loss and service disruption.

Software vulnerabilities: Unpatched software vulnerabilities in any component of the system can be exploited by attackers to gain unauthorized access or disrupt operations.

Privacy concerns: Collecting and storing data across multiple locations in a distributed system raises privacy concerns.

4) Discuss the major issues in designing a distributed operating system. Explain the main characteristics of a distributed event-based system.

Issues:

Transparency: Hiding distribution complexity from users (access, location, failure), balancing it with practicality.

Consistency: Maintaining data consistency across multiple nodes despite concurrency, failures, and network partitions.

Concurrency Control: Preventing race conditions and deadlocks when multiple processes access shared resources concurrently.

Scalability: Gracefully handling growing demands in users, resources, and data.

Fault Tolerance: Ensuring continued operation despite individual component failures.

Security: Securing a wider attack surface from cyberattacks, insider threats, and vulnerabilities.

Heterogeneity: Managing diverse hardware, software, and network configurations.

Characteristics of a distributed event-based system:

Event-driven communication: React to events (significant changes) instead of explicit messages.

Decentralized control: No central coordinator, each component reacts independently.

Asynchronous communication: Improved performance and reduced latency.

Local state management: Each component maintains its state, consistency might need management.

Dynamic event processing: Filtering, transforming, and aggregating events for flexibility.

Scalability: Easy to scale by adding more event producers and consumers.

Fault tolerance: Failures in individual components don't necessarily disrupt the entire system.

5) Explain various RPC protocols supporting client-server communication.

Topic is not covered in the class hence there is a high chance that it will not be there in midsem.

6) Explain why and how a client is prevented from calling arbitrary code within a server under lightweight RPC.

Topic is not covered in the class hence there is a high chance that it will not be there in midsem.

7) Discuss the general organization of a distributed computing system and explain its characteristic features.

Unlike centralized systems, distributed systems spread their components across multiple geographically dispersed computers. This distribution is typically layered, with each layer handling specific tasks:

Application Layer: Highest level, where user applications interact with the system.

Service Layer: The middleware layer provides services like communication, naming, and security.

Resource Layer: Manages physical resources like processors, memory, and storage.

Network Layer: Provides communication infrastructure for data exchange between components.

Resource Sharing: Multiple users and applications can access and share resources across the network.

Concurrency: Multiple processes can execute concurrently on different machines, potentially interacting with shared resources.

Transparency: Users generally perceive the system as a single entity, unaware of the distributed nature.

Scalability: Systems can be easily expanded by adding more resources to meet growing demands.

Fault Tolerance: Individual component failures shouldn't bring down the entire system.

Heterogeneity: Different machines with varying hardware and software can participate in the system.

8) What are the main objectives and challenges of distributed systems?

Objectives:

Resource Sharing: Maximize resource utilization for multiple users and applications.

Scalability: Easily adapt to growing demands by adding resources.

Fault Tolerance: Continue operating despite individual component failures.

Performance: Potentially achieve better performance through distributed processing.

Geographical Dispersion: Enable global collaboration and access regardless of location.

Challenges:

Complexity: Management and debugging require careful design and strategies.

Consistency: Maintaining data consistency across nodes can be intricate.

Concurrency Control: Preventing race conditions and deadlocks is crucial.

Security: A distributed system provides a wider vulnerable interface hence security is a big challenge there.