DS - 2023 June

(01)

a) SOA is an architectural pattern in software design where services are provided to other components by application components, through a communication protocol over a network. A service is a discrete unit of functionality that can be accessed remotely & acted upon and updated ~~ide~~ independantly, such as receiving credit card statement online.

Traditional RPC frameworks like CORBA, JAVA RMI tend to be tightly coupled & have several security issues related to trust, firewalls, and limited interoperability between different components./frameworks. These frameworks rely on synchronous communication, where the client & server are closely dependent on each other's interface & implementation details, causing integration and scalability challenges.

SoA, on other ~~h~~hand, promotes loose coupling. Services communicate over a network using standard protocols such as HTTP/~~HTT~~PS and data formats like XML or JSON, which are platform-independent. This allows different services to be developed, deployed and scaled independently. SOA also enhances interoperability & reusability of services across different platforms and technologies.

b) Monolithic architecture is a traditional model of ~~soft~~software application structure, where all components of software are resides together and interdependent. It involves a single codebase & deployment, making it simpler to develop intially, but harder to scale and maintain. Any change in a module requires a redeploying the entire application, and scaling often involves duplicating the entire application rather than just the necessary components.

Service-Oriented Architecture (SOA) organize software components as discrete services that communicate over a network. Each service performs a distinct function and can be independently developed,

deployed, and maintained. SOA supports scalability and flexibility better than monolithic architecture but can be complex to manage due to its middleware requirements.

Microservices Architecture takes the principles of SOA further by decomposing an application into small, loosely-coupled services that operate independently. Each microservice corresponds to a single function or business capability and runs in its own process. Microservices use lightweight protocols like HTTP/REST or m messaging queues for communication. This architecture improves scalability and development speed by allowing different teams to develop, deploy and scale services independently. However, it introduces complexity in service coordination and requires robust service discovery and management solution.

c) Service discovery in a microservice architecture is crutial for dynamically locating services. On the server side, this is often implemented using service registry and discovery pattern. When a microservice starts, it registers its location (IP address, port) and metadata with a service registry.

Clients or other microservices query the service registry to find the network location of service instances. There are 2 main types of service discovery: client-side and server-side. In server-side service discovery, the client makes a request to a load balancer, which queries the service registry to find available instances and forwards the request to one of them. This offloads the discovery logic from the client to the server, simplyfying client design and centralizing control over request routing.

d) The 4 main dependability factors of a software system are:

Availability ⇒ means the system is ready to be used immediately. This measures the propotion of time that a system is operational and accessible when needed for use. High availability ensures that the system can perform its function whenever required, minimizing downtime.

Reliability ⇒ refers the system can run continuously without failure. This defines the ability of a system to perform its required functions under stated conditions for a specified period of time. A reliable system operates without failure over a defined period.

Safety ⇒ This ensures that the system operates without causing unacceptable risk of harm. Safety is critical in systems where failure can result in catastrophic consequences, such as healthcare or aerospace systems.

Maintainability ⇒ This refers to the ease with which a system can be repaired or modified. High maintainability ensures that any issues can be quickly resolved, & the system can be easily updated to meet new requirements or correct defects.

e) Given MTBF = 250,000 h

MTTR = 8 H

$$\text{Availability} = \frac{MTBF}{MTBF + MTTR}$$

$$= \frac{250\,000}{250\,000 + 8}$$

$$\approx 0.999$$

∴ Availability is approximately 99.9 %

f) Redundancy is in fault tolerance can be categorized into 4 sections.

I. Hardware Redundancy ⇒ This involves adding extra hardware components such as additional processors, memory modules, or complete systems than can take over in case of a failure. Techniques include passive (static), active (dynamic), and hybrid redundancy. Passive redundancy masks faults using voting mechanisms, while active redundancy detects & replaces faulty components.

II. Software Redundancy ⇒ This involves using multiple software versions to perform the same task. Techniques include N-version programming (NVP) where multiple versions run concurrently, & recovery blocks where versions run serially with acceptance tests to ensure correctness.

III Time Redundancy ⇒ This involves repeating operations over time to ensure reliability. If an error occures, the operation is retired after some interval. This is useful for transeint faults that might not occur in subsequent attempts.

IV Information Redundancy ⇒ This involves adding extra information to the data to detect and correct errors. Techniques include error-correcting codes like hamming codes, parity checks, and cyclic redundancy checks (CRC) to ensure data integrity during transmisston and storage.

(02)

a) (i) To get an alert from a remote health monitoring system, only when the heart rate exceeds a particular value.

Java RMI with asyne. callback functions : This method allows the server to notify the client asynchronously when the heart rate exceeds a certain threshold, ensuring timely alerts without constant polling.

(ii) To login to an online banking application using the account id and password.

Java RMI blocking : This method ensures a synchronous call where the client waits for the server to authenticate the credentials before proceeding, maintaining the necessary security for login operations.

(iii) To check the heart rate using a remote health monitoring system, every 5 minutes.

Java RMI-based polling : This method is suitable for periodic checks where the client polls the server at regular intervals to get the current heart rate, ensuring up-to-date monitoring.

(iv) Fire alerm system that can sending/receving in an internal system with high internet traffic.

Socket programming : This method is efficient for high-frequency, low-latency communication in environments with high traffic, ensuring reliable and fast data transfer for the fire alarm system.

b) Remote interface ⇒ This defines the methods that can be called remotely by clients. It extends the "java.rmi.Remote" interface and declares methods that can throw "RemoteException". This interface is implemented by the remote object, which provides the actual functionality.

Stub and skelton ⇒ The stub acts as a proxy on the client side, forwarding calls to the remote object. The skelton (older versions) receives these calls on the server side and passes them to the actual remote object. In newer versions, the skelton is replaced by dynamically generated proxies.

c) (i)

sphere service interface

```
public interface SphereService extends Remote {
    public double calculateArea (double radius) throws
                                        RemoteException;
    public double calculateVolume ( double radius) throws
                                        RemoteException;
}
```

```
public class SphereServiceImpl extends UnicastRemoteObject
                                implements SphereService {
    protected SphereServiceImpl() throws RemoteException {
        super();
    }

    @Override
    public double calculateArea (double radius) throws RemoteException {
        return 4 * Math.PI * radius * radius;
    }
}
```

```
    @Override
    public double calculateVolume (double radius) throws
                                                RemoteException {
        return (4.0/3) * Math.PI * radius * radius * radius;
    }
}

public class SphereServer {
    Public static void main (String [] args) {
        try {
            SphereService service = new SphereServiceImpl ();
            Registry registry = LocationRegistry.createRegistry
                                                (1099);
            registry.rebind ("Sphereserver", service);
            System.out.println ("Sphereservice is running at
                                tcp://localhost/Sphereservice");
        } catch (Exception e) {
            e.printStackTrace ();
        }
    }
}
```

continues to last page

(03)

a) Select a specific customer ⇒ '/customers/{customerId}' GET
select all customers ⇒ '/customers' GET
create new customer ⇒ '/customers' POST
update a customer ⇒ '/customers/{customerId}' PUT
delete a customer ⇒ '/customers/{customerId}' DELETE

b) Centralized control and management ⇒

~~Orchestration allows for a single point of control where all interactions and dependencies between services are managed centrally. This central management simplifies monitoring, debugging, and managing~~

In orchestration, a central coordinator (orchestor) controls and manages the interactions between different services. This centralization simplifies monitoring, error handling, and updating the workflow, making it easier to manage the overall process.

Easier implementation and modification ⇒

Orchestration allows for easier implementation and modification of workflows. Changes to the process can be made in the orchestrator without requiring changes to the individual services. This is specially benificial when dealing with complex business processes that frequently change.

c)
```
<?xml version="1.0" encoding="UTF-8" ?>
<Subjects>
    <Subject id="1">
        <subject_name> Mathematics </subject_name>
        <lecturer>
            <firstname> John </firstname>
            <lastname> Smith </lastname>
        </lecturer>
    </subject>
    <subject>id="2">
        <subject_name> Silence </subject_name>
        <lecturer>
            <firstname> Jane </firstname>
            <lastname> Sarah </lastname>
        </lecturer>
    </subject>
</Subjects>
```

d)
```
"Subjects" : {
  "Subject" : [
    {
      "id" : 1;
      "subject-name" : "Mathematics";
      "Lecturer" : {
        "firstname" : "John";
        "lastname" : "Smith";
      }
    },
    {
      "id" : 2;
      "subject-name" : "Science";
      "Lecturer" : {
        "firstname" : "Jane";
        "lastname" : "Sarah";
      }
    }
  ]
}
```

e) web content presentation ⇒ XSLT can be used to convert XML documents HTML, facilitating the display of data on web pages.
Data exchange and interoperability ⇒ XSLT helps in converting XML data into other formats, enabling interoperability between systems with different data requirements.

f)
```
<html>
  <body>
    <h2> My CD collection </h2>
    <table border = "1" >
      <tr bgcolor = "#9acd32" >
```

```
        <th> Title </th>
        <th>Artist </th>
        <th> Year </th>
    </tr>
    <tr>
        <td>Empire Burlesque </td>
        <td> Bob Dylan </td>
        <td> 1985 </td>
    </tr>
    <tr>
        <td> Hide your heart </td>
        <td> Bonnie Tyler </td>
        <td> 1988 </td>
    </tr>
</table>
</body>
</html>
```

(04)

a) (i) A highend VM with a GPU being obtained from the cloud to train a ML model.

Iaas ---- provisioning virtualized computing resources (VM with GPU) from the cloud (provides the fundamental building blocks of computing)

(ii) A cloud-based online student management system being used.

Saas --- complete s/w application hosted on the cloud, which is accessed over internet

(iii) An IoT integration platform that is hosted in the cloud being used to build an IoT application.

Paas ---- using a cloud-based platform that provides tools and services to build, test and deploy IoT applications

</subjects>

b) (i) Standard ecommerce application deployed by start-up company that doesn't have lot of capital.

Public cloud --- most cost effective

(ii) Hosting defense department data in a virtualized environment...

Private cloud --- sensitive data

(iii) continue to use university's existing in-house data centers, without investing to buy additional h/w when there's additional peak-hour traffic during online exams in the exam time period.

Hybrid cloud

c) Web application load balancer.

specified service ⇒ routes traffic to multiple servers.

derived service ⇒ redirects traffic to operational ~~nodes~~ servers during a server failure.

Database replication.

specified service ⇒ stores and retrieves data.

derived service ⇒ backup server takes over in case of primary server failure, ensuring continous data access.

d) Key problem of single commit protocol

Single commit protocol does not ensure consistent commit across all participants, leading to potential inconsistencies.

solution by two-phase commit protocol

introduces a prepare phase and commit phase to ensure all participants either commit or abort, achieving consistency.

e) drift ⇒ Gradual deviation of a clock from the correct time.

skew ⇒ Instantaneous difference in time readings between 2 clocks.

f) similarity : both aim to synchronize clocks in a distributed system.
differences :

Christian's Algorithm ⇒ uses a single time server.

Berkerly clock sync. Algorithm ⇒ uses a central coordinator to average times from all nodes.

g) key improvement of vector clocks over Lampart clocks ; and necessity of the improvement.

↳ vector clocks provide the advantage of detecting concurrent events which Lampart clocks cannot, enhancing the ability to maintain a consistent state across distributed systems.

Question (02)

c)

(ii)

```
public class SphereClient {
    public static void main(String[] args) {
        try {
            SphereService sphereService = (SphereService) Naming.lookup
                                          ("rmi://localhost/SphereServer");
            double radius = 5.0;
            double area = sphereService.calculateArea(radius);
            double volume = sphereService.calculateVolume(radius);
            System.out.println("Area of sphere : " + area);
            System.out.println("Volume of sphere : " + volume);
        } catch (Exception e) {
            System.out.println(e.toString());
            e.printStackTrace();
        }
    }
}
```

2023 Regular

(01)

a) SOA ⇒ SOA refers to group of functionalities and provide them as atomic services. In SOA, we separate entire system into atomic services. When separating atomic services, we have to use a middle layer, where we can add or remove services whenever we needed. We need to have a specific data access protocols and (HTTP/HTTPS) ⟷ data transfer formats like XML and JSON when ~~using~~ dealing with the above mentioned middle layer.

Traditional RPC frameworks like CORBA, Java RMI have several issues with tight coupling, security problems and interoperability between frameworks. These RPC frameworks rely on synchronous communication, where the client and the server closely depend on each other's. And Functionalities include within the component.

~~SOA~~ In SOA, we get the functionalities out of the business layer. Instead of adding the functionality within the component, we have a middle layer as "business service layer", add all the atomic services there. SOA promotes loose-coupling. Services communicate over the network using HTTP/HTTPS data access protocols and XML, JSON data transfer formats, which are platform independent. SOA also enhances interoperability & resuability of services across different platforms and technologies.

b) Monolithic architecture ⇒ traditional method of ~~software~~ development, where all the components of software are resides toghther as a whole unit and they are interdependent. It involves a single codebase and deployment, making it simpler to develop initially, but harder to scale and maintain. Any change in a component ~~application~~ requires re-deploying entire application, rather than the necessary component.

SOA ⇒ SOA groups the functionalities and provide them as atomic services. In SOA, we have our atomic services in a separate, ~~layer~~ middle layer known as "business service layer", where we can add ~~are~~ or remove (atomic) services whenever we needed. ~~SOA promotes loose coupling among services, resuability of services and interoperability among different services technologies and platforms. we re~~ Each services perform a particular function and can be independently deployed. SOA supports scalability and flexibility better than monolithic architecture.

Microservices ⇒ Microservices is the extended version of SOA. Microservices consist of small, loosely coupled services (components) that act independently. Each microservices can run its own business function. Microservices use lightweight communication protocols like HTTP/REST or messaging queues. for communication. This architecture improves scalability.