

Name:

WAQAS ASHIQ

Roll No:

Fa11-Ciit-DDP-BCS-201

Section:

C

Submitted To:

DR. Syed Asim Ali

**Q1-a)**

**Distributed Transparency:**

An important goal of distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers. A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent.

**Access transparency:**

Example of Access transparency, A distributed system may have computer systems that run different operating systems, each having their own file-naming conventions, difference in naming conventions, as well as how files can be manipulated or they may have different architecture, should all be hidden from users and applications.

**Location Transparency:**

Example of Location Transparency, User cannot tell where the resource is physically located in the system. All the URL addresses are examples of location transparency e.g. <http://www.prenhall.com/index.html>, user cannot identify where the index.html actually resides in the system.

**Relocation Transparency:**

An example of relocation transparency is when mobile users can continue to use wireless laptops while moving from one place to other without being disconnected.

**Q1-b)**

**Role of middle Middleware:**

Distributed systems are organized by means of a layer of software-that is called middleware, it is logically placed between a higher-level layer consisting of users and applications and the layer underneath consisting of operating systems and basic communication facilities. Middleware actually provides the communication between other components at the same time it hides the as best as possible the differences in hardware and operating system in general it help to achieve distributed transparency.

**Q1-c)**

**Scalable system:**

Scalable systems are meant to be those systems which can be expandable systems can be scalable in different dimensions. First system can be scalable with respect to its size, meaning can add more users and resources to the system. Second a geographically scalable system is one in which the users and resources may lie far apart. Third, a system can be administratively scalable meaning it can still be easy to manage even of it spans many independent administrative organizations.

**Q2-a)**

**Three-tiered architecture:**

A three-tiered client-server architecture consists of three logical layers,

where each layer is, in principle, implemented at a separate machine. The

highest layer consists of a client user interface, the middle layer contains the

actual application, and the lowest layer implements the data that are being

used

**Q2-b)**

**Structured overlay network:**

The problem is that we are dealing only with *logical* paths. It may very well be the case that two nodes *A* and *B* which are neighbors in the overlay network are physically placed far apart. As a consequence, the logically short path between *A* and *B* may require routing a message along a very long path in the underlying physical network.

**Q3-a)**

**Threads:**

Yes, for two reasons. First, threads require memory for setting up their own

private stack. Consequently, having many threads may consume too much

memory for the server to work properly. Another, more serious reason, is that,

to an operating system, independent threads tend to operate in a chaotic manner.

In a virtual memory system it may be difficult to build a relatively stable

working set, resulting in many page faults and thus I/O. Having many threads

may thus lead to a performance degradation resulting from page thrashing.

Even in those cases where everything .it’s into memory, we may easily see that

memory is accessed following a chaotic pattern rendering caches useless.

Again, performance may degrade in comparison to the single-threaded case.

**Q3-b)**

**Concurrent and multithreaded servers:**

While both servers accomplish the same basic task, they do so in different ways. A concurrent server that uses separate server processes to handle each connection might not work as efficiently as a multithreading server, since the server must spawn a new server instance for each connection. Multithreading servers, in using lightweight threads, are usually more efficient for brief connections. Users that connect to the server for long periods or engage in heavy data usage, might be better off having their own instance through a process rather than a thread.

**Q3-c)**

**TCP/IP:**

Assuming the server maintains no other information on that client, one could justifiably argue that the server is stateless. The issue is that not the server, but the transport layer at the server maintains state on the client. What the local operating systems keep track of is, in principle, of no concern to the server.

**Q4-a)**

**Transport-level communication:**

They hardly offer distribution transparency meaning that application developers

are required to pay significant attention to implementing communication,

often leading to proprietary solutions. The effect is that distributed applications,

for example, built directly on top of sockets are difficult to port and to

interoperate with other applications.

**Q4-b)**

**Reliable multicast service:**

In principle, a reliable multicast service could easily be part of the transport

layer, or even the network layer. As an example, the unreliable IP multicasting

service is implemented in the network layer. However, because such services

are currently not readily available, they are generally implemented using transport-

level services, which automatically places them in the middleware. However,

when taking scalability into account, it turns out that reliability can be

guaranteed only if application requirements are considered. This is a strong

argument for implementing such services at higher, less general layers.