Scheme of Evaluation
Internal Assessment Test 3 – July 2021

Sub:	Cloud Computing and its Application							Code:	18CS643
Date:	30/07/2021	Duration:	90mins	Max Marks:	50	Sem:	VI	Branch:	ISE

Note: **Answer Any five full questions.**

Question #		Description	Marks Distribution		Max Marks
1	a)	The hospital wants to set up remote ECG monitoring service. Describe how cloud computing technology can be applied to remote ECG monitoring. Explain with proper diagram. Diagram Explanation	3M 3M	6M	10M
1	b)	Compare and contrast Azure and AWS. Any 4 differences	1M*4 4M	4M	
2	a)	Explain the SQL Azure architecture with neat diagram. Diagram- 3M Explanation-3M	6M	6M	10M
2	b)	Categorize the following AWS services in the compute, storage, communication and other services category a) AMI b) S3 c) EC2 d)Amazon EBS e) SimpleDB f) Virtual Private Network g) CloudWatch h) Simple Queue Service	0.5M*8	4M	
3	a)	Discuss the storage services provided by AWS. S3 , EBS ,Elasticache, SimpleDB, RDS,CloudFront	1M*6	6M	10M
3	b)	Explain how cloud computing is used in media application with neat diagram (Any one application). Animoto / Maya Rendering Diagram Explanation	2M 2M	4M	

4	b)	Compare Local Thread and Aneka thread (Draw diagrams wherever required) Interface compatibility – Thread Life Cycle- Thread Synchronization- Thread Priorities- Thread Serialization-	 2M 2M 2M 2M 2M	10M	
5	a)	A company needs to develop the technique for the parallel computation. Analyze the different techniques that company can use to parallelize computations? Draw neat diagrams wherever required. Domain Decomposition Functional Decomposition	 3M 3M	6M	10M
5	b)	Define Task. Explain the computing categories that relate to task. Task High Performance Computing High Throughput Computing Many Task Computing	 1M 1M 1M 1M	4M	
6	a)	Explain the MPI reference scenario and MPI programming structure with the required diagram. MPI reference scenario Diagram MPI Program Structure Explanation	 2M 2M 2M	6M	
6	b)	Explain the Task Programming Model with neat diagram. Diagram Explanation	 2M 2M	4M	

Scheme Of Evaluation
Internal Assessment Test 3 – July 2021

Sub:	Cloud Computing and its Application							Code:	18CS643
Date:	30/07/2021	Duration:	90mins	Max Marks:	50	Sem:	VI	Branch:	ISE

Note: **Answer Any full five questions**

Q. 1 The hospital wants to set up remote ECG monitoring service. Describe how cloud computing technology can be applied to remote ECG monitoring. Explain with proper diagram.

Healthcare: ECG analysis in the cloud

Healthcare is a domain in which computer technology has found several and diverse applications: from supporting the business functions to assisting scientists in developing solutions to cure diseases.

An illustration of the infrastructure and model for supporting remote ECG monitoring is shown in **Figure 10.1**. Wearable computing devices equipped with ECG sensors constantly monitor the patient's heartbeat. Such information is transmitted to the patient's mobile device, which will eventually forward it to the cloud-hosted Web service for analysis.

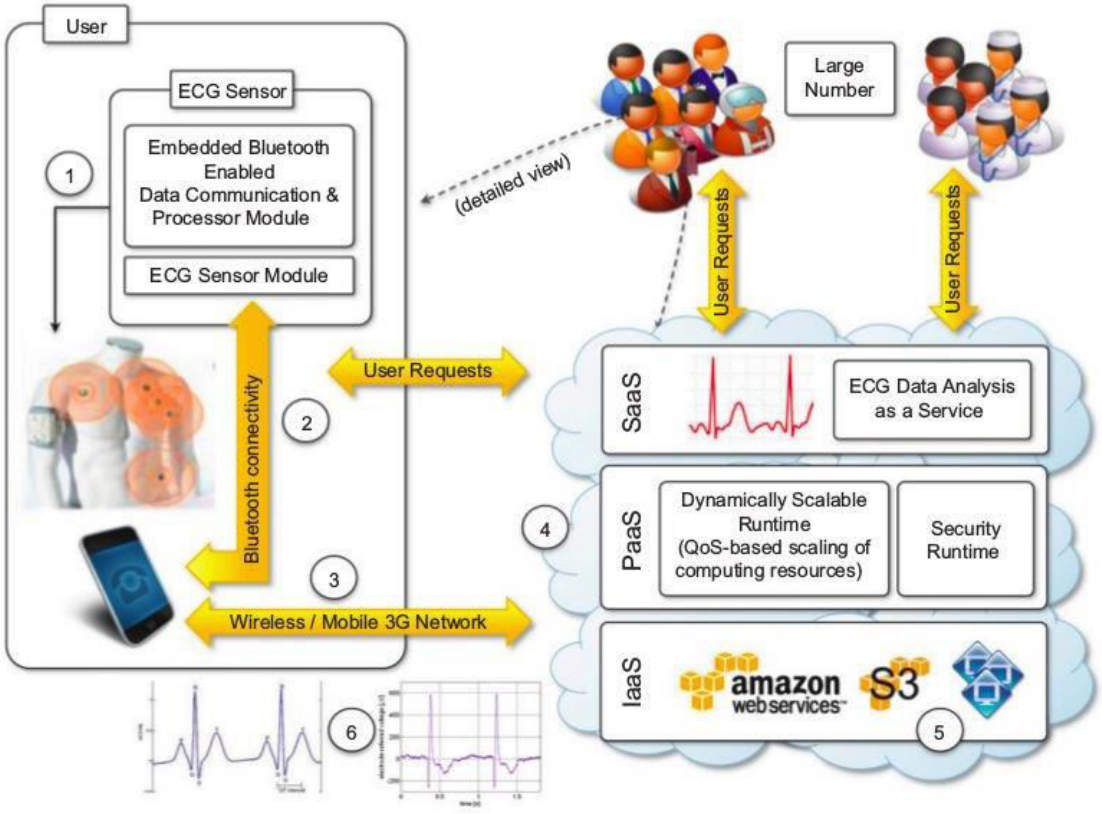


FIGURE 10.1

An online health monitoring system hosted in the cloud.

The Web service forms the front-end of a platform that is hosted in cloud and leverages three layers of cloud computing stack: SaaS, PaaS, and IaaS.

The Web service constitute SaaS application that will store ECG data in the Amazon S3 service and issue a processing request to the scalable cloud platform.

The runtime platform is composed of a dynamically sizable number of instances running the workflow engine and Aneka.

The number of workflow engine instances is controlled according to the number of requests in the queue of each instance, while Aneka controls the number of EC2 instances used to execute the single tasks defined by the workflow engine for a single ECG processing job.

Advantages

1. The elasticity of cloud infrastructure that can grow and shrink according to the requests served. As a result, doctors and hospitals do not have to invest in large computing infrastructures designed after capacity planning, thus making more effective use of budgets.
2. Ubiquity. Cloud computing technologies are easily accessible and promise to deliver systems with minimum or no downtime. Computing systems hosted in cloud are accessible from any Internet device through simple interfaces (such as SOAP and REST-based Web services). This makes systems easily integrated with other systems maintained on hospital's premises.
3. Cost savings. Cloud services are priced on a pay-per-use basis and with volume prices for large numbers of service requests.

Q.1 b) Compare and contrast Azure and AWS

AZURE VERSUS AWS

Azure	AWS
Azure is the public cloud platform for Microsoft.	AWS is an on-demand cloud computing platform for Amazon.
Microsoft has had a bad relationship with the open source community.	Amazon has been friendly with the open source model from the start.
Azure excels in the Hybrid Cloud space allowing companies to integrate onsite servers with cloud instances.	AWS is still looking to strengthen its offerings to support Hybrid Clouds.
Azure has a lesser reach when it comes to Government cloud offerings.	AWS has a little edge over Azure in terms of Government cloud offerings.
As an enterprise-grade cloud computing platform, Azure has a strong global presence spanning over 36 regions all around the world.	Amazon is more about numbers than geographic expansion with AWS regions worldwide.
Microsoft provides a less flexible pricing model.	Amazon offers a more flexible pricing model.

Q. 2 a) Explain the SQL Azure architecture with neat diagram.

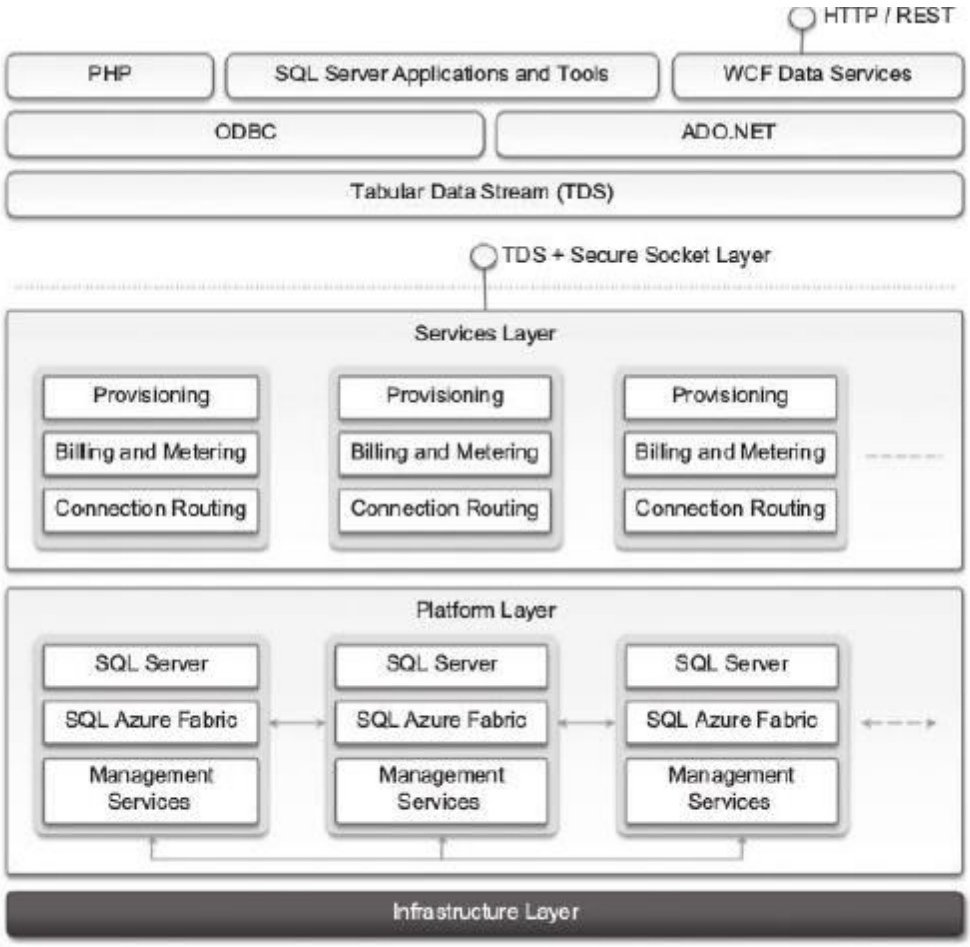


FIGURE 9.4

SQL Azure architecture.

SQL Azure

SQL Azure is a relational database service hosted on Windows Azure and built on the SQL Server technologies. The service extends the capabilities of SQL Server to the cloud and provides developers with a scalable, highly available, and fault-tolerant relational database. SQL Azure is accessible from either the Windows Azure Cloud or any other location that has access to the Azure Cloud. It is fully compatible with the interface exposed by SQL Server, so applications built for SQL Server can transparently migrate to SQL Azure. **Figure 9.4** shows the architecture of SQL Azure. Access to SQL Azure is based on the Tabular Data Stream (TDS) protocol, which is the communication protocol underlying all the different interfaces used by applications to connect to a SQL Server-based installation such as ODBC and ADO.NET.

Developers have to sign up for a Windows Azure account in order to use SQL Azure. Once the account is activated, they can either use the Windows Azure Management Portal or the REST APIs to create servers and logins and to configure access to servers.

SQL Azure servers are abstractions that closely resemble physical SQL Servers: They have a fully qualified domain name under the database.windows.net (i.e., server-name.database.windows.net) domain name. This simplifies the management tasks and the interaction with SQL Azure from client applications.

Currently, two different editions are available: Web Edition and Business Edition. The former is suited for small Web applications and supports databases with a maximum size of 1 GB or 5 GB. The latter is suited for independent software vendors, line-of-business applications, and enterprise applications and supports databases with a maximum size from 10 GB to 50 GB, in increments of 10 GB.

Q. 2 b) Categorize the following AWS services in the compute, storage, communication and other services category

a) AMI b) S3 c) EC2 d)Amazon EBS e) SimpleDB f) Virtual Private Network g) CloudWatch h) Simple Queue Service

a) AMI- Compute Service
b) S3-storage service
c) EC2-Compute Service
d) EBS- storage service
e) SimpleDB- Storage Service
f) Virtual Private Network-communication service
g) CloudWatch-Additional services
h) Simple Queue Service- communication service

Q. 3a) Discuss the storage services provided by AWS.

Storage services

The core service is represented by Amazon Simple Storage Service (S3). This is a distributed object store that allows users to store information in different formats. The core components of S3 are two: buckets and objects. Buckets represent virtual containers in which to store objects; objects represent the content that is actually stored. Objects can also be enriched with metadata that can be used to tag the stored content with additional information.

1 S3 key concepts
2 Amazon elastic block store
3 Amazon ElastiCache
4 Structured storage solutions
5 Amazon CloudFront

Amazon S3 allows controlling the access to buckets and objects by means of Access Control Policies (ACPs). An ACP is a set of grant permissions that are attached to a resource expressed by means of an XML configuration file.
A policy allows defining up to 100 access rules, each of them granting one of the available permissions to a grantee.

1 S3 key concepts

S3 has been designed to provide a simple storage service that's accessible through a Representational State Transfer (REST) interface.

- The storage is organized in a two-level hierarchy.
- Stored objects cannot be manipulated like standard files.
- Content is not immediately available to users.
- Requests will occasionally fail.

Access to S3 is provided with RESTful Web services. These express all the operations that can be performed on the storage in the form of HTTP requests (GET, PUT, DELETE, HEAD, and POST).

Resource naming

Buckets, objects, and attached metadata are made accessible through a REST interface. Therefore, they are represented by uniform resource identifiers (URIs) under the s3.amazonaws.com domain.

Amazon offers three different ways of addressing a bucket:

1. Canonical form: http://s3.amazonaws.com/bukect_name/
2. Subdomain form: http://bucketname.s3.amazon.com/
3. Virtual hosting form: http://bucket-name.com/

Buckets

A bucket is a container of objects. It can be thought of as a virtual drive hosted on the S3 distributed storage, which provides users with a flat store to which they can add objects. Buckets are top-level elements of the S3 storage architecture and do not support nesting. That is, it is not possible to create "subbuckets" or other kinds of physical divisions.

Objects and metadata

Objects constitute the content elements stored in S3. Users either store files or push to the S3 text stream representing the object's content. An object is identified by a name that needs to be unique within the bucket in which the content is stored. The name cannot be longer than 1,024 bytes when encoded in UTF-8, and it allows almost any character. Buckets do not support nesting.

Access control and security

2 Amazon elastic block store

The Amazon Elastic Block Store (EBS) allows AWS users to provide EC2 instances with persistent storage in the form of volumes that can be mounted at instance startup. They accommodate up to 1 TB of space and are accessed through a block device interface, thus allowing users to format them according to the needs of the instance they are connected to.

EBS volumes normally reside within the same availability zone of the EC2 instances that will use them to maximize the I/O performance. It is also possible to connect volumes located in different availability zones. Once mounted as volumes, their content is lazily loaded in the background and according to the request made by the operating system. This reduces the number of I/O requests that go to the network.

3 Amazon ElastiCache

ElastiCache is an implementation of an elastic in-memory cache based on a cluster of EC2 instances.

It provides fast data access through a Memcached-compatible protocol so that applications can transparently migrate to ElastiCache.

ElastiCache is based on a cluster of EC2 instances running the caching software, which is made available through Web services.

An ElastiCache cluster can be dynamically resized according to the demand of the client applications.

4 Structured storage solutions

Amazon provides applications with structured storage services in three different forms:

- Preconfigured EC2 AMIs,
- Amazon Relational Data Storage (RDS), and
- Amazon SimpleDB.

Preconfigured EC2 AMIs are predefined templates featuring an installation of a given database management system. EC2 instances created from these AMIs can be completed with an EBS volume for storage persistence. Available AMIs include installations of IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase, and Vertica.

RDS is relational database service that relies on the EC2 infrastructure and is managed by Amazon. Developers do not have to worry about configuring the storage for high availability, designing failover strategies, or keeping the servers up-to-date with patches. Moreover, the service provides users with automatic backups, snapshots, point-in-time recoveries, and facilities for implementing replications.

Amazon SimpleDB is a lightweight, highly scalable, and flexible data storage solution for applications that do not require a fully relational model for their data. SimpleDB provides support for semistructured data, the model for which is based on the concept of domains, items, and attributes.

SimpleDB uses domains as top-level elements to organize a data store. These domains are roughly comparable to tables in the relational model. Unlike tables, they allow items not to have all the same column structure; each item is therefore represented as a collection of attributes expressed in the form of a key-value pair.

5 Amazon CloudFront

CloudFront is an implementation of a content delivery network on top of the Amazon distributed storage infrastructure. It leverages a collection of edge servers strategically located around the globe to better serve requests for static and streaming Web content so that the transfer time is reduced.

AWS provides users with simple Web service APIs to manage CloudFront. To make available content through CloudFront, it is necessary to create a distribution. This identifies an origin server, which contains the original version of the content being distributed, and it is referenced by a DNS domain under the Cloudfront.net domain name.

Q. 3 b) Explain how cloud computing is used in media application with neat diagram (Any one application).

1 Animoto

Animoto is the most popular example of media applications on the cloud. The Website provides users with a very straightforward interface for quickly creating videos out of images, music, and video fragments submitted by users. Users select a specific theme for a video, upload the photos and videos and order them in the sequence they want to appear, select the song for the music, and render the video. The process is executed in the background and the user is notified via email once the video is rendered.

A proprietary artificial intelligence (AI) engine, which selects the animation and transition effects according to pictures and music, drives the rendering operation. Users only have to define the storyboard by organizing pictures and videos into the desired sequence.

The infrastructure of Animoto is complex and is composed of different systems that all need to scale (**Figure 10.8**). The core function is implemented on top of the Amazon Web Services infrastructure. It uses Amazon EC2 for the Web front-end and worker nodes; Amazon S3 for the storage of pictures, music, and videos; and Amazon SQS for connecting all the components.

The system's auto-scaling capabilities are managed by Rightscale, which monitors the load and controls the creation of new worker instances.

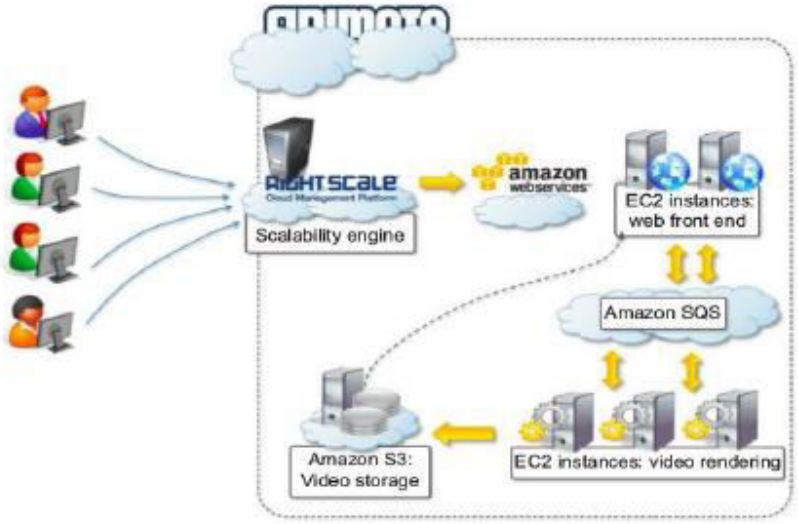


FIGURE 10.8

Animoto reference architecture.

Q. 4 a) Compare Aneka thread with Local thread. (Draw diagrams wherever required).

Aneka thread vs. common threads

To efficiently run on a distributed infrastructure, Aneka threads have certain limitations compared to local threads.

These limitations relate to the communication and synchronization strategies.

1 Interface compatibility
2 Thread life cycle
3 Thread synchronization
4 Thread priorities
5 Type serialization

1 Interface compatibility

The Aneka.Threading.AnekaThread class exposes almost the same interface as the System.Threading.Thread class with the exception of a few operations that are not supported. Table 6.1 compares the operations that are exposed by the two classes.

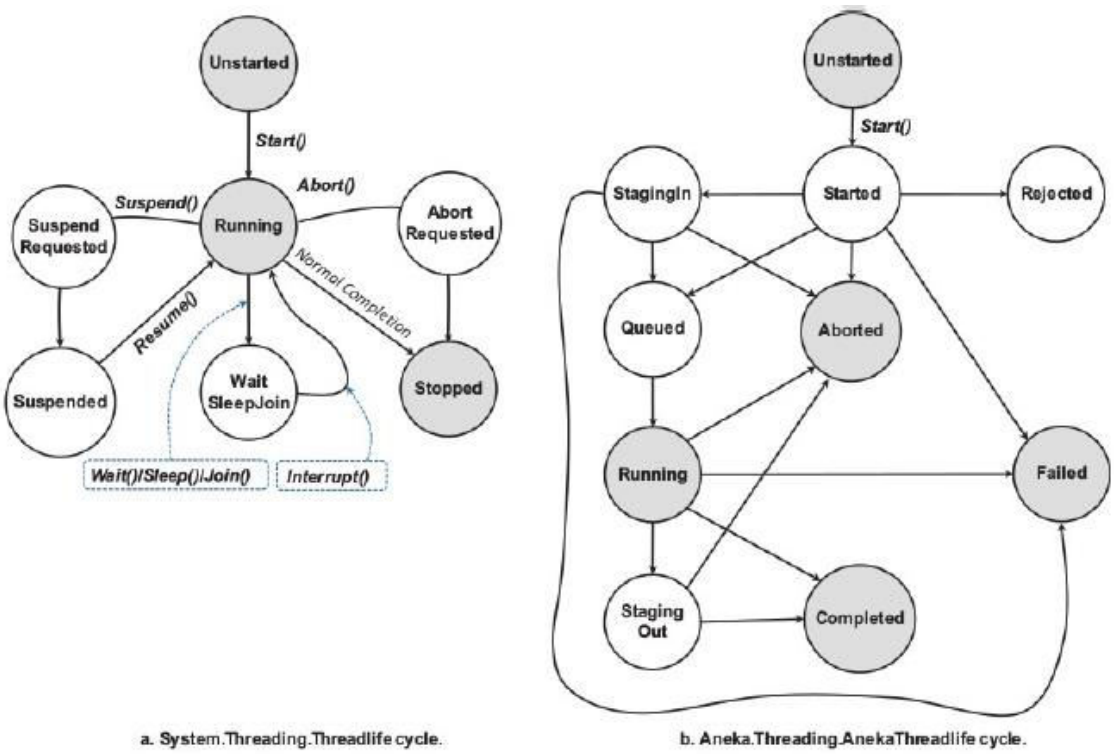
Table 6.1 Thread API Comparison

.Net Threading API	Aneka Threading API
System.Threading	*Aneka.Threading*
Thread	*AnekaThread*
Thread.ManagedThreadId (int)	*AnekaThread.Id (string)*
Thread.Name	*AnekaThread.Name*
Thread.ThreadState (ThreadState)	*AnekaThread.State*
Thread.isAlive	*AnekaThread.IsAlive*
Thread.isRunning	*AnekaThread.IsRunning*
Thread.IsBackground	*AnekaThread.IsBackground[false]*
Thread.Priority	*AnekaThread.Priority[ThreadPriority.Normal]*
Thread.IsThreadPoolThread	*AnekaThread.IsThreadPoolThread [false]*
Thread.Start	*AnekaThread.Start*
Thread.Abort	*AnekaThread.Abort*
Thread.Sleep	[Not provided]
Thread.Interrupt	[Not provided]
Thread.Suspend	[Not provided]
Thread.Resume	[Not provided]
Thread.Join	*AnekaThread.Join*

The basic control operations for local threads such as Start and Abort have a direct mapping, whereas operations that involve the temporary interruption of the thread execution have not been supported.
The reasons for such a design decision are twofold. **First**, the use of the Suspend/Resume operations is generally a deprecated practice, even for local threads, since Suspend abruptly interrupts the execution state of the thread. **Second**, thread suspension in a distributed environment leads to an ineffective use of the infrastructure, where resources are shared among different tenants and applications.
Sleep operation is not supported. Therefore, there is no need to support the Interrupt operation, which forcibly resumes the thread from a waiting or a sleeping state.

2 Thread life cycle

Aneka Thread life cycle is different from the life cycle of local threads. It is not possible to directly map the state values of a local thread to Aneka threads. **Figure 6.6** provides a comparative view of the two life cycles. The white balloons in the figure indicate states that do not have a corresponding mapping on the other life cycle; the shaded balloons indicate the common states.
In local threads most of the state transitions are controlled by the developer, who actually triggers the state transition by invoking methods. Whereas inAneka threads, many of the state transitions are controlled by the middleware.



a. System.Threading.Threadlife cycle.

b. Aneka.Threading.AnekaThreadlife cycle.

FIGURE 6.6

Thread life-cycle comparison.

Aneka threads support file staging and they are scheduled by the middleware, which can queue them for a considerable amount of time.

An Aneka thread is initially found in the Unstarted state. Once the Start() method is called, the thread transits to the Started state, from which it is possible to move to the StagingIn state if there are files to upload for its execution or directly to the Queued state. If there is any error while uploading files, the thread fails and it ends its execution with the Failed state, which can also be reached for any exception that occurred while invoking Start().

Another outcome might be the Rejected state that occurs if the thread is started with an invalid reservation token. This is a final state and implies execution failure due to lack of rights.
Once the thread is in the queue, if there is a free node where to execute it, the middleware moves all the object data and depending files to the remote node and starts its execution, thus changing the state into Running.
If the thread generates an exception or does not produce the expected output files, the execution is considered failed and the final state of the thread is set to Failed. If the execution is successful, the final state is set to Completed. If there are output files to retrieve, the thread state is set to StagingOut while files are collected and sent to their final destination.
At any point, if the developer stops the execution of the application or directly calls the Abort() method, the thread is aborted and its final state is set to Aborted.

3 Thread synchronization

The .NET base class libraries provide advanced facilities to support thread synchronization by the means of monitors, semaphores, reader-writer locks, and basic synchronization constructs at the language level. Aneka provides minimal support for thread synchronization that is limited to the implementation of the join operation for thread abstraction.
This requirement is less stringent in a distributed environment, where there is no shared memory among the thread instances and therefore it is not necessary.
Providing coordination facilities that introduce a locking strategy in such an environment might lead to distributed deadlocks that are hard to detect. Therefore, by design Aneka threads do not feature any synchronization facility except join operation.

4 Thread priorities

The System.Threading.Thread class supports thread priorities, where the scheduling priority can be one selected from one of the values of the ThreadPriority enumeration: Highest, AboveNormal, Normal, BelowNormal, or Lowest.
Aneka does not support thread priorities, the Aneka.Threading.Thread class exhibits a Priority property whose type is ThreadPriority, but its value is always set to Normal, and changes to it do not produce any effect on thread scheduling by the Aneka middleware.

5 Type serialization

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file. Main purpose is to save the state of an object to recreate it when needed. The reverse process is called deserialization.
Local threads execute all within the same address space and share memory; therefore, they do not need objects to be copied or transferred into a different address space. Aneka threads are distributed and execute on remote computing nodes, and this implies that the object code related to the method to be executed within a thread needs to be transferred over the network.
A .NET type is considered serializable if it is possible to convert an instance of the type into a binary array containing all the information required to revert it to its original form or into a possibly different execution context. This property is generally given for several types defined in the .NET framework by simply tagging the class definition with the Serializable attribute.
Aneka threads execute methods defined in serializable types, since it is necessary to move the enclosing instance to remote execution method. In most cases, providing serialization is as easy as tagging the class definition with the Serializable attribute; in other cases it might be necessary to implement the ISerializable interface and provide appropriate constructors for the type.

Q. 5a) A company needs to develop the technique for the parallel computation. Analyze the different techniques that company can use to parallelize computations? Draw neat diagrams wherever required.

Techniques for parallel computation with threads

Developing parallel applications requires an understanding of the problem and its logical structure. Decomposition is a useful technique that aids in understanding whether a problem is divided into components (or tasks) that can be executed concurrently. it allows the breaking down into independent units of work that can be executed concurrently with the support provided by threads.

1 Domain decomposition
2 Functional decomposition
3 Computation vs. Communication

1 Domain decomposition

Domain decomposition is the process of identifying patterns of functionally repetitive, but independent, computation on data. This is the most common type of decomposition in the case of throughput computing, and it relates to the identification of repetitive calculations required for solving a problem. The master-slave model is a quite common organization for these scenarios:

- The system is divided into two major code segments.
- One code segment contains the decomposition and coordination logic.
- Another code segment contains the repetitive computation to perform.

- A master thread executes the first code segment.
- As a result of the master thread execution, as many slave threads as needed are created to execute the repetitive computation.
- The collection of the results from each of the slave threads and an eventual composition of the final result are performed by the master thread.

Embarrassingly parallel problems constitute the easiest case for parallelization because there is no need to synchronize different threads that do not share any data. Embarrassingly parallel problems are quite common, they are based on the strong assumption that at each of the iterations of the decomposition method, it is possible to isolate an independent unit of work. This is what makes it possible to obtain a high computing throughput. If the values of all the iterations are dependent on some of the values obtained in the previous iterations, the problem is said to be **inherently sequential**. **Figure 6.3** provides a schematic representation of the decomposition of embarrassingly parallel and inherently sequential problems.

The matrix product computes each element of the resulting matrix as a linear combination of the corresponding row and column of the first and second input matrices, respectively. The formula that applies for each of the resulting matrix elements is the following:

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} B_{kj}$$

Two conditions hold in order to perform a matrix product:

- Input matrices must contain values of a comparable nature for which the scalar product is defined.
- The number of columns in the first matrix must match the number of rows of the second matrix.

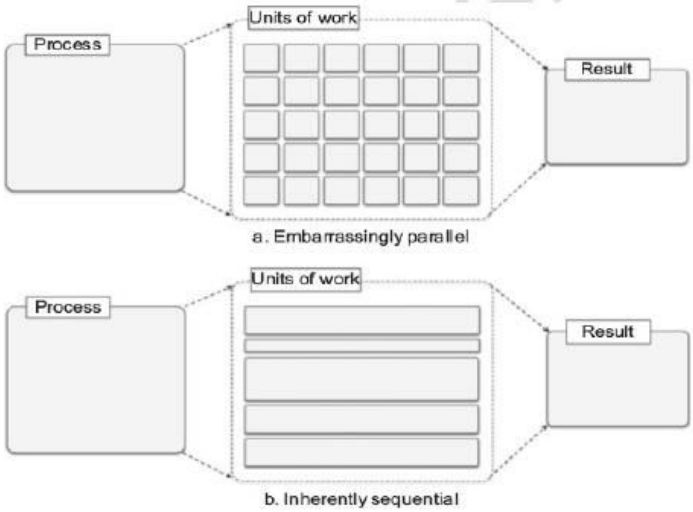


FIGURE 6.3

Domain decomposition techniques.

Figure 6.4 provides an overview of how a matrix product can be performed.

2 Functional decomposition

Functional decomposition is the process of identifying functionally distinct but independent computations. The focus here is on the type of computation rather than on the data manipulated by the computation. This kind of decomposition is less common and does not lead to the creation of a large number of threads, since the different computations that are performed by a single program are limited.
Functional decomposition leads to a natural decomposition of the problem in separate units of work.
Figure 6.5 provides a pictorial view of how decomposition operates and allows parallelization.

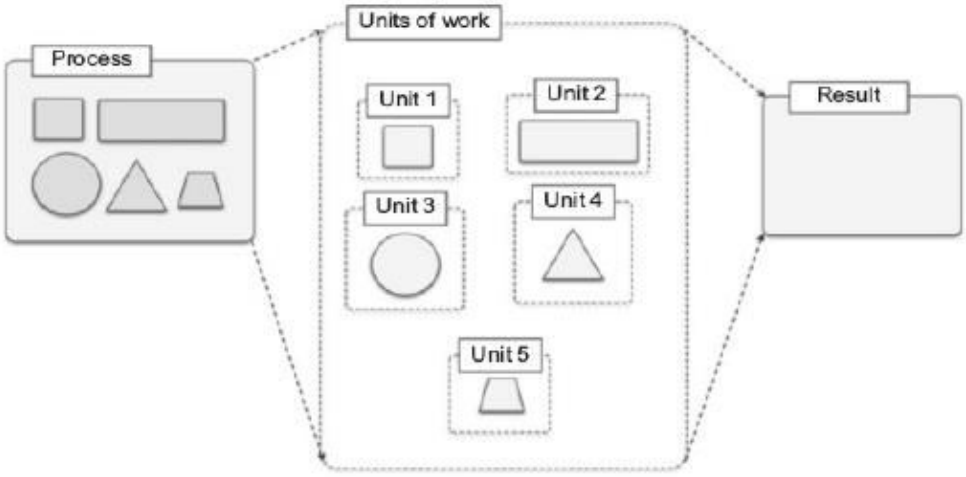


FIGURE 6.5

Functional decomposition.

The problems that are subject to functional decomposition can also require a composition phase in which the outcomes of each of the independent units of work are composed together.

In the following, we show a very simple example of how a mathematical problem can be parallelized using functional decomposition. Suppose, for example, that we need to calculate the value of the following function for a given value of x:

$$f(x) = \sin(x) + \cos(x) + \tan(x)$$

Once the value of x has been set, the three different operations can be performed independently of each other. This is an example of functional decomposition because the entire problem can be separated into three distinct operations. A possible implementation of a parallel version of computation is shown in **Listing 6.3.**

Q. 5 b) Define Task. Explain the computing categories that relate to task.

Computing categories

These categories provide an overall view of the characteristics of the problems. They implicitly impose requirements on the infrastructure and the middleware.

Applications falling into this category are:

1 High-performance computing
2 High-throughput computing
3 Many-task computing

1 High-performance computing

High-performance computing (HPC) is the use of distributed computing facilities for solving problems that need large computing power.

The general profile of HPC applications is constituted by a large collection of compute-intensive tasks that need to be processed in a short period of time.
The metrics to evaluate HPC systems are floating-point operations per second (FLOPS), now tera-FLOPS or even peta-FLOPS, which identify the number of floating- point operations per second.
Ex: supercomputers and clusters are specifically designed to support HPC applications that are developed to solve "Grand Challenge" problems in science and engineering.

2 High-throughput computing

High-throughput computing (HTC) is the use of distributed computing facilities for applications requiring large computing power over a long period of time.
HTC systems need to be robust and to reliably operate over a long time scale.
The general profile of HTC applications is that they are made up of a large number of tasks of which the execution can last for a considerable amount of time.
Ex: scientific simulations or statistical analyses.
It is quite common to have independent tasks that can be scheduled in distributed resources because they do not need to communicate.
HTC systems measure their performance in terms of jobs completed per month.

3 Many-task computing

MTC denotes high-performance computations comprising multiple distinct activities coupled via file system operations.
MTC is the heterogeneity of tasks that might be of different nature: Tasks may be small or large, single-processor or multiprocessor, compute-intensive or data-intensive, static or dynamic, homogeneous or heterogeneous.
MTC applications includes loosely coupled applications that are communication-intensive but not naturally expressed using the message-passing interface.
It aims to bridge the gap between HPC and HTC. MTC is similar to HTC, but it concentrates on the use of many computing resources over a short period of time to accomplish many computational tasks.

Q. 6 a) Explain the MPI reference scenario and MPI programming structure with the required diagram.

MPI applications

Message Passing Interface (MPI) is a specification for developing parallel programs that communicate by exchanging messages.
MPI has originated as an attempt to create common ground from the several distributed shared memory and message-passing infrastructures available for distributed computing. Now a days, MPI has become a de facto standard for developing portable and efficient message-passing HPC applications.

MPI provides developers with a set of routines that:

- Manage the distributed environment where MPI programs are executed
- Provide facilities for point-to-point communication
- Provide facilities for group communication
- Provide support for data structure definition and memory allocation
- Provide basic support for synchronization with blocking calls

The general reference architecture is depicted in Figure 7.4. A distributed application in MPI is composed of a collection of MPI processes that are executed in parallel in a distributed infrastructure that supports MPI.

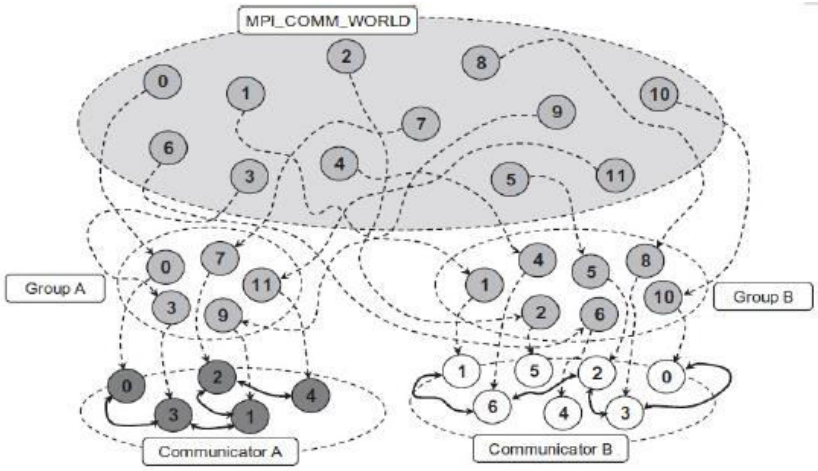


FIGURE 7.4

MPI reference scenario.

MPI applications that share the same MPI runtime are by default as part of a global group called MPI_COMM_WORLD. Within this group, all the distributed processes have a unique identifier that allows the MPI runtime to localize and address them.

Each MPI process is assigned a rank within the group.

The rank is a unique identifier that allows processes to communicate with each other within a group.

To create an MPI application it is necessary to define the code for the MPI process that will be executed in parallel. This program has, in general, the structure described in **Figure 7.5**.

The section of code that is executed in parallel is clearly identified by two operations that set up the MPI environment and shut it down, respectively.

In the code section, it is possible to use all the MPI functions to send or receive messages in either asynchronous or synchronous mode.

The diagram in **Figure 7.5** might suggest that the MPI might allow the definition of completely symmetrical applications, since the portion of code executed in each node is the same.

A common model used in MPI is the master-worker model, where by one MPI process coordinates the execution of others that perform the same task.

Once the program has been defined in one of the available MPI implementations, it is compiled with a modified version of the compiler for the language.

The output of the compilation process can be run as a distributed application by using a specific tool provided with the MPI implementation.

One of the most popular MPI software environments is developed by the Argonne National Laboratory in the United States.

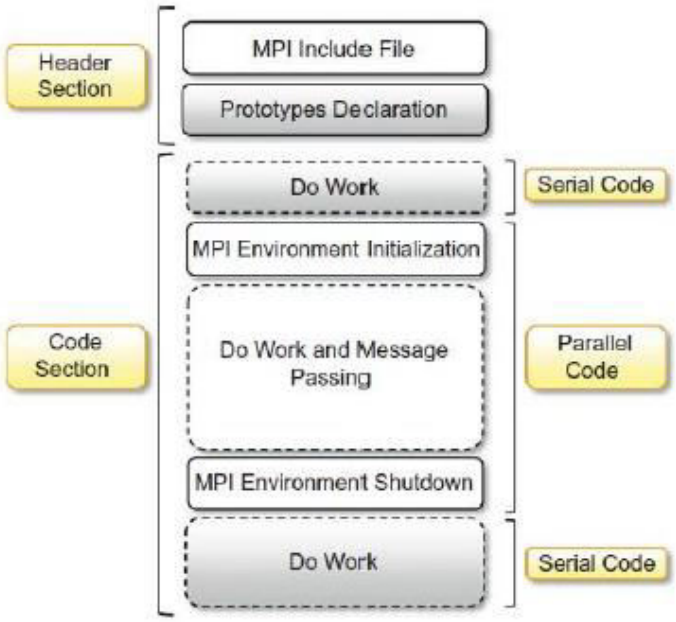


FIGURE 7.5

MPI program structure.

Q. 6 b) Explain the Task Programming Model with neat diagram.

Task programming model

The Task Programming Model provides a very intuitive abstraction for quickly developing distributed applications on top of Aneka.

It provides a minimum set of APIs that are mostly centered on the Aneka.Tasks.ITask interface.

Figure 7.8 provides an overall view of the components of the Task Programming Model and their roles during application execution.

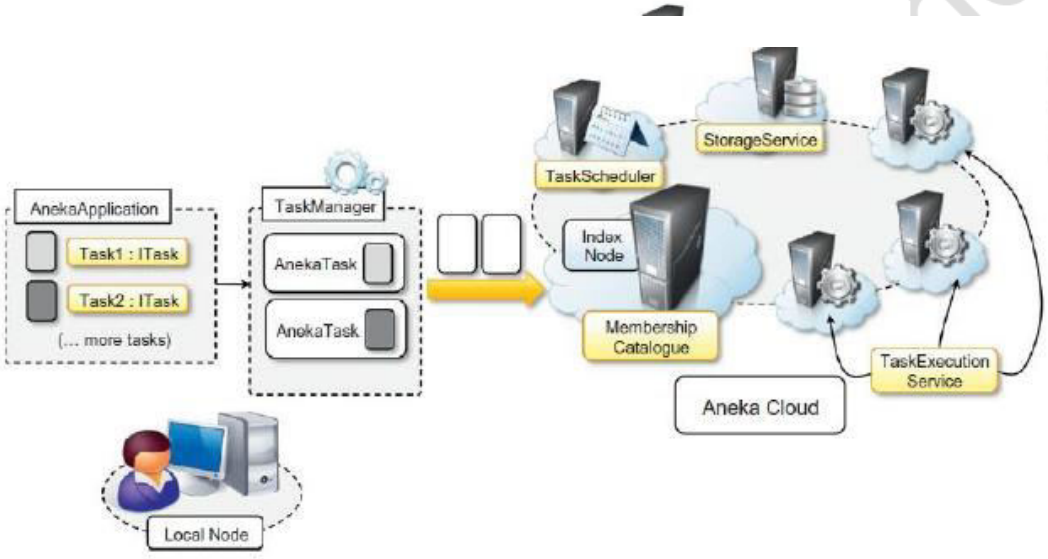


FIGURE 7.8

Task programming model scenario.

Developers create distributed applications in terms of ITask instances, the collective execution of which describes a running application.

These tasks, together with all the required dependencies (data files and libraries), are grouped and managed through the Aneka Application class, which is specialized to support the execution of tasks.

Two other components, AnekaTask and TaskManager, constitute the client-side view of a task-based application. The former constitutes the runtime wrapper Aneka uses to represent a task within the middleware; the latter is the underlying component that interacts with Aneka, submits the tasks, monitors their execution, and collects the results.

In the middleware, four services coordinate their activities inorder to execute task-based applications. These are MembershipCatalogue, TaskScheduler, ExecutionService, and StorageService.

MembershipCatalogue constitutes the main access point of the cloud and acts as a service directory to locate the TaskScheduler service that is incharge of managing the execution of task-based applications.

Its main responsibility is to allocate task instances to resources featuring the Execution Service for task execution and for monitoring task state.