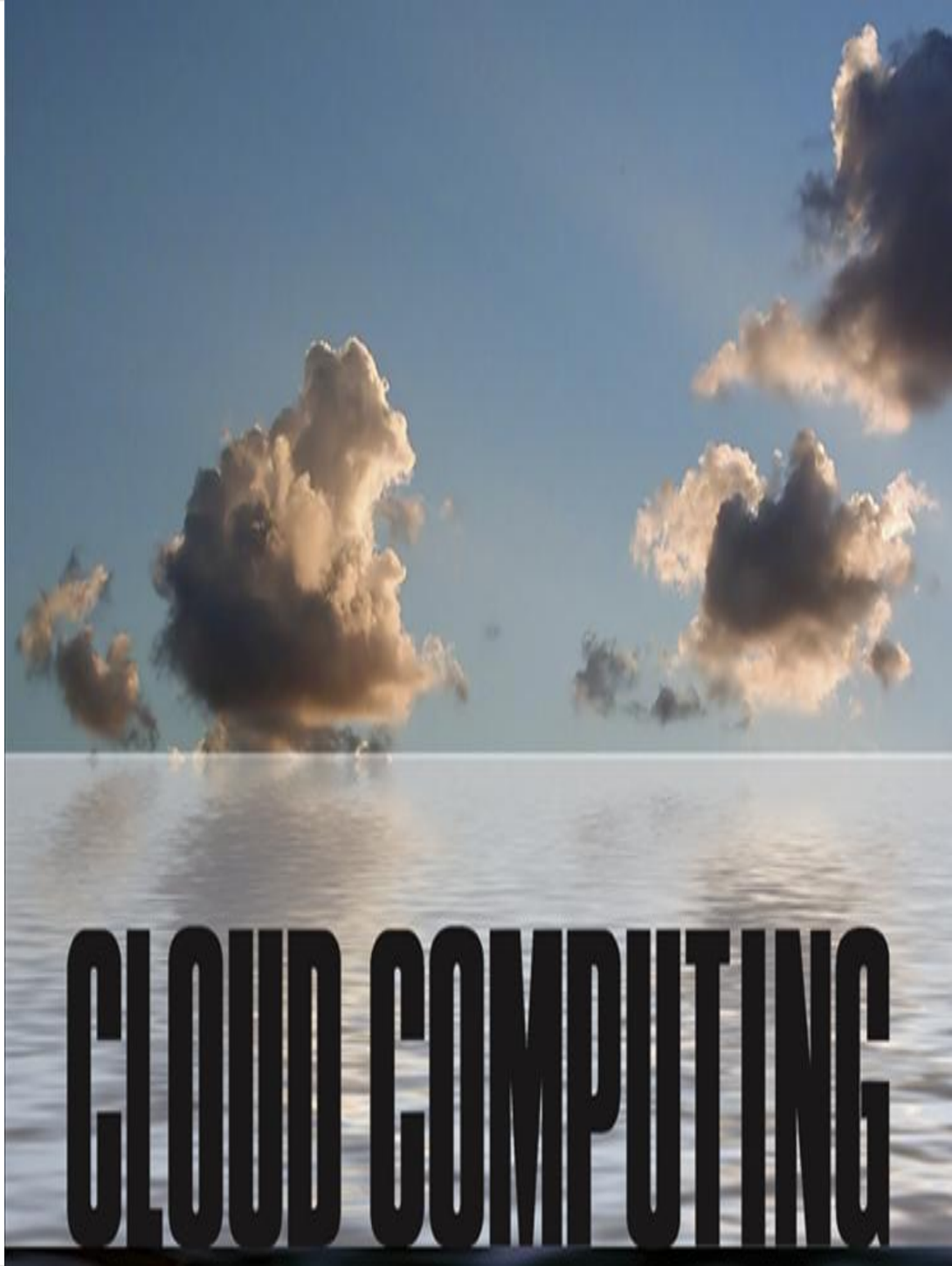
CLOUD COMPUTING AND SECURITY (BCS601)

Module-5



	CLOUD COMPUTING		Semester	6
	Course Code	BCS601	CIE Marks	50
	Teaching Hours/Week (L: T:P: S)	3:0:2:0	SEE Marks	50
	Total Hours of Pedagogy	40	Total Marks	100
	Credits	04	Exam Hou3rs	3
	Examination type (SEE)	Theory/Practical		

Course objectives:

- Introduce the rationale behind the cloud computing revolution and the business drivers
- Understand various models, types and challenges of cloud computing
- Understand the design of cloud native applications, the necessary tools and the design tradeoffs.
- Realize the importance of Cloud Virtualization, Abstraction`s, Enabling Technologies and cloud security

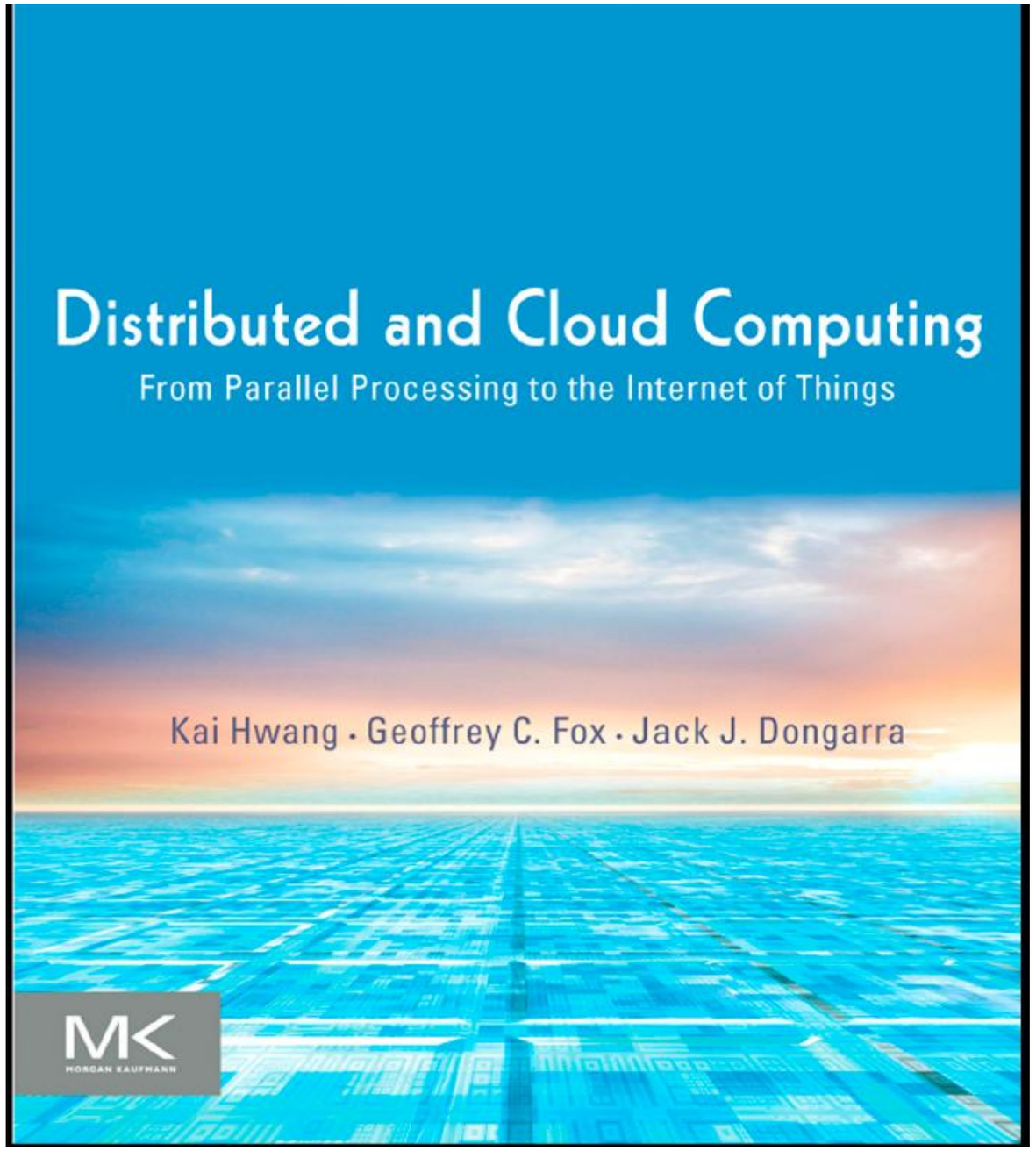
Course outcome (Course Skill Set)

At the end of the course, the student will be able to:

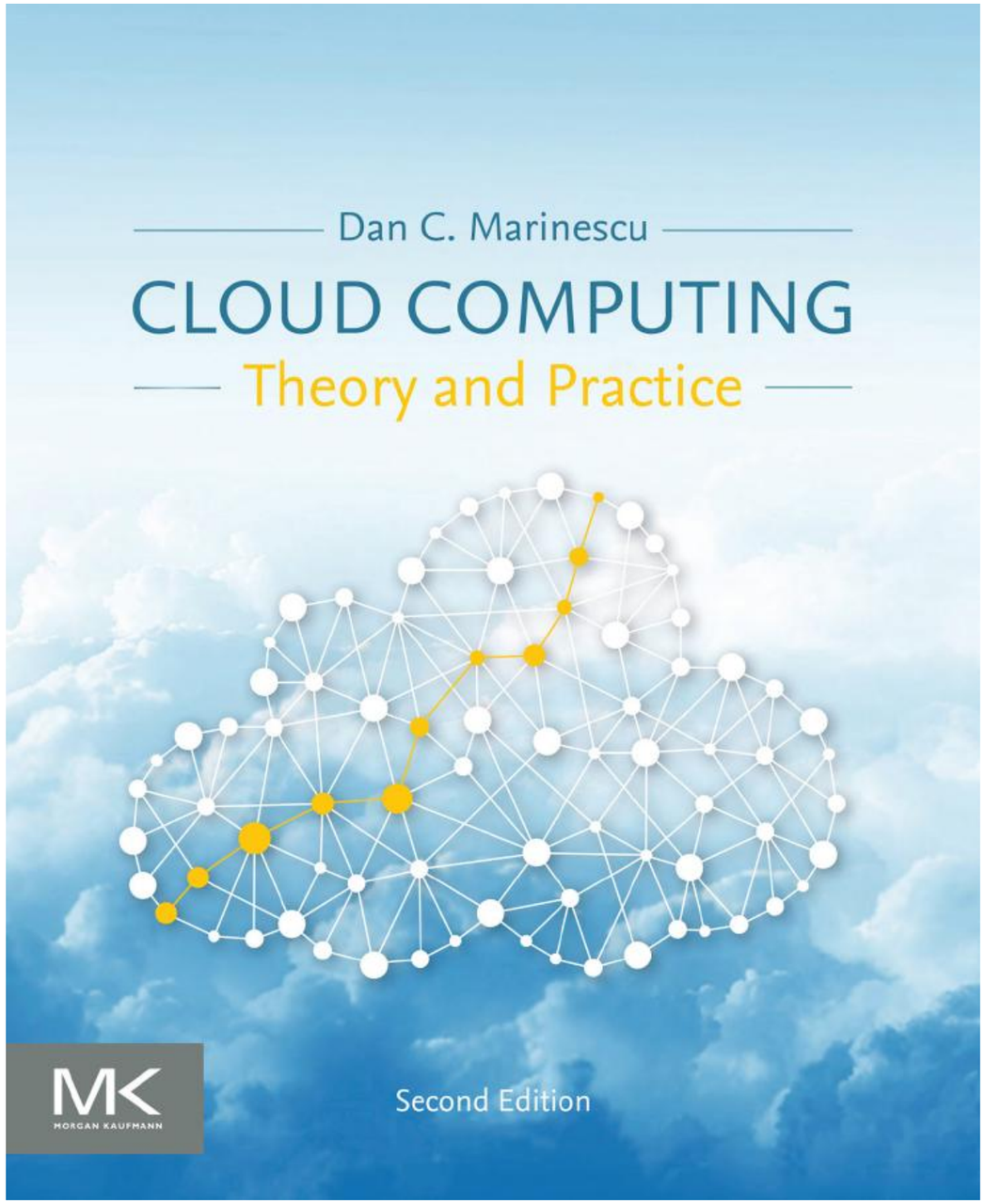
1. Describe various cloud computing platforms and service providers.
2. Illustrate the significance of various types of virtualization.
3. Identify the architecture, delivery models and industrial platforms for cloud computing based applications.
4. Analyze the role of security aspects in cloud computing.
5. Demonstrate cloud applications in various fields using suitable cloud platforms.

TEXT BOOKS

MODULE 1 to MODULE 5



MODULE 4



	Module-5
	Cloud Programming and Software Environments: Features of Cloud and Grid Platforms, Parallel and Distributed Computing Paradigms, Programming Support for Google App Engine, Programming on Amazon AWS and Microsoft, Emerging Cloud Software Environments. **Textbook 1: Chapter 6: 6.1 to 6.5**

MODULE-5 :Cloud Programming and Software Environments

Features of Cloud and Grid Platforms

- **Parallel and Distributed Programming Paradigms**

Introduction

• Cloud computing is a technology that enables us to create, configure, and customize applications through an internet connection

.• Software environment is a collection of programs, libraries, and utilities that allow users to perform specific tasks

• Software environments are often used by programmers to develop applications or run existing ones.

• A software environment for a particular application could include the operating system, the database system, specific development tools, or compilers.

Features of Cloud and Grid Platforms

• Important features in real cloud and grid platforms!!

• In four tables, we cover the capabilities, traditional features, data features, and features for programmers and runtime systems to use

• The entries in these tables are source references for anyone who wants to program the cloud efficiently

Grid Computing

• Grid computing (sometimes referred to as virtual supercomputing) is a group of networked computers that work together as a virtual supercomputer to perform large tasks, such as analyzing huge sets of data or weather modeling.

• Grid computing is extensively used in scientific research and high- performance computing to solve complex scientific problems.

• For example, grid computing can be used to simulate the behavior of a nuclear explosion, model the human genome, or analyze massive amounts of data generated from particle accelerators.

Advantages:

• Can solve larger, more complex problems in a shorter time

• Easier to collaborate with other organizations

• Make better use of existing hardware

Cloud platform capabilities

Table 6.1 Important Cloud Platform Capabilities

Capability	Description
Physical or virtual computing platform	The cloud environment consists of some physical or virtual platforms. Virtual platforms have unique capabilities to provide isolated environments for different applications and users.
Massive data storage service, distributed file system	With large data sets, cloud data storage services provide large disk capacity and the service interfaces that allow users to put and get data. The distributed file system offers massive data storage service. It can provide similar interfaces as local file systems.
Massive database storage service	Some distributed file systems are sufficient to provide the underlying storage service application developers need to save data in a more semantic way. Just like DBMS in the traditional software stack, massive database storage services are needed in the cloud.
Massive data processing method and programming model	Cloud infrastructure provides thousands of computing nodes for even a very simple application. Programmers need to be able to harness the power of these machines without considering tedious infrastructure management issues such as handling network failure or scaling the running code to use all the computing facilities provided by the platforms.

Cont..

	provided by the platforms.
Workflow and data query language support	The programming model offers abstraction of the cloud infrastructure. Similar to the SQL language used for database systems, in cloud computing, providers have built some workflow language as well as data query language to support better application logic.
Programming interface and service deployment	Web interfaces or special APIs are required for cloud applications: J2EE, PHP, ASP, or Rails. Cloud applications can use Ajax technologies to improve the user experience while using web browsers to access the functions provided. Each cloud provider opens its programming interface for accessing the data stored in massive storage.
Runtime support	Runtime support is transparent to users and their applications. Support includes distributed monitoring services, a distributed task scheduler, as well as distributed locking and other services. They are critical in running cloud applications.
Support services	Important support services include data and computing services. For example, clouds offer rich data services and interesting data parallel execution models like MapReduce.

Infrastructure Cloud Features

Table 6.2 Infrastructure Cloud Features

Accounting: Includes economies; clearly an active area for commercial clouds

Appliances: Preconfigured virtual machine (VM) image supporting multifaceted tasks such as message-passing interface (MPI) clusters

Authentication and authorization: Could need single sign-on to multiple systems

Data transport: Transports data between job components both between and within grids and clouds; exploits custom storage patterns as in BitTorrent

Operating systems: Apple, Android, Linux, Windows

Program library: Stores images and other program material

Registry: Information resource for system (system version of metadata management)

Security: Security features other than basic authentication and authorization; includes higher level concepts such as trust

Cont..

such as trust

Scheduling: Basic staple of Condor, Platform, Oracle Grid Engine, etc.; clouds have this implicitly as is especially clear with Azure Worker Role

Gang scheduling: Assigns multiple (data-parallel) tasks in a scalable fashion; note that this is provided automatically by MapReduce

Software as a Service (SaaS): Shared between clouds and grids, and can be supported without special attention; Note use of services and corresponding service oriented architectures are very successful and are used in clouds very similarly to previous distributed systems.

Virtualization: Basic feature of clouds supporting "elastic" feature highlighted by Berkeley as characteristic of what defines a (public) cloud; includes virtual networking as in ViNe from University of Florida

Traditional Features in Cluster, Grid and Paralel computing Environments

Table 6.3 Traditional Features in Cluster, Grid, and Parallel Computing Environments

Cluster management: ROCKS and packages offering a range of tools to make it easy to bring up clusters

Data management: Included metadata support such as RDF triple stores (Semantic web success and can be built on MapReduce as in SHARD); SQL and NOSQL included in

Grid programming environment: Varies from link-together services as in Open Grid Services Architecture (OGSA) to GridRPC (Ninf, GridSolve) and SAGA

OpenMP/threading: Can include parallel compilers such as Cilk; roughly shared memory technologies. Even transactional memory and fine-grained data flow come here

Portals: Can be called (science) gateways and see an interesting change in technology from portlets to HUBzero and now in the cloud: Azure Web Roles and GAE

Scalable parallel computing environments: MPI and associated higher level concepts including ill-fated HP FORTRAN, PGAS (not successful but not disgraced), HPCS languages (X-10, Fortress, Chapel), patterns (including Berkeley dwarves), and functional languages such as F# for distributed memory

Virtual organizations: From specialized grid solutions to popular Web 2.0 capabilities such as Facebook

Workflow: Supports workflows that link job components either within or between grids and clouds; relate to LIMS Laboratory Information Management Systems.

Platform Features supported by Clouds and Grids

Table 6.4 Platform Features Supported by Clouds and (Sometimes) Grids

Blob: Basic storage concept typified by Azure Blob and Amazon S3

DPFS: Support of file systems such as Google (MapReduce), HDFS (Hadoop), and Cosmos (Dryad) with compute-data affinity optimized for data processing

Fault tolerance: As reviewed in [1] this was largely ignored in grids, but is a major feature of clouds

MapReduce: Support MapReduce programming model including Hadoop on Linux, Dryad on Windows HPCS, and Twister on Windows and Linux. Include new associated languages such as Sawzall, Pregel, Pig Latin, and LINQ

Monitoring: Many grid solutions such as Inca. Can be based on publish-subscribe

Notification: Basic function of publish-subscribe systems

Programming model: Cloud programming models are built with other platform features and are related to familiar web and grid models

Queues: Queuing system possibly based on publish-subscribe

Cont..

Queues: Queuing system possibly based on publish-subscribe

Scalable synchronization: Apache Zookeeper or Google Chubby. Supports distributed locks and used by BigTable. Not clear if (effectively) used in Azure Table or Amazon SimpleDB

SQL: Relational database

Table: Support of table data structures modeled on Apache Hbase or Amazon SimpleDB/Azure Table. Part of NOSQL movement

Web role: Used in Azure to describe important link to user and can be supported otherwise with a portal framework. This is the main purpose of GAE

Worker role: Implicitly used in both Amazon and grids but was first introduced as a high-level construct by Azure

Data and software protection techniques

6.1.2.3 Security, Privacy, and Availability

The following techniques are related to security, privacy, and availability requirements for developing a healthy and dependable cloud programming environment. We summarize these techniques here. Some of these issues have been discussed in Section 4.4.6 with plausible solutions.

- Use virtual clustering to achieve dynamic resource provisioning with minimum overhead cost.
- Use stable and persistent data storage with fast queries for information retrieval.
- Use special APIs for authenticating users and sending e-mail using commercial accounts.
- Cloud resources are accessed with security protocols such as HTTPS and SSL.
- Fine-grained access control is desired to protect data integrity and deter intruders or hackers.
- Shared data sets are protected from malicious alteration, deletion, or copyright violations.
- Features are included for availability enhancement and disaster recovery with life migration of VMs.
- Use a reputation system to protect data centers. This system only authorizes trusted clients and stops pirates.

System issues for running a typical parallel program

6.2.1 Parallel Computing and Programming Paradigms

Consider a distributed computing system consisting of a set of networked nodes or workers. The system issues for running a typical parallel program in either a parallel or a distributed manner would include the following [36–39]:

- **Partitioning** This is applicable to both computation and data as follows:
- **Computation partitioning** This splits a given job or a program into smaller tasks. Partitioning greatly depends on correctly identifying portions of the job or program that can be performed concurrently. In other words, upon identifying parallelism in the structure of the program, it can be divided into parts to be run on different workers. Different parts may process different data or a copy of the same data.
- **Data partitioning** This splits the input or intermediate data into smaller pieces. Similarly, upon identification of parallelism in the input data, it can also be divided into pieces to be processed on different workers. Data pieces may be processed by different parts of a program or a copy of the same program.
- **Mapping** This assigns the either smaller parts of a program or the smaller pieces of data to underlying resources. This process aims to appropriately assign such parts or pieces to be run simultaneously on different workers and is usually handled by resource allocators in the system.

Cont..

- **Synchronization** Because different workers may perform different tasks, synchronization and coordination among workers is necessary so that race conditions are prevented and data dependency among different workers is properly managed. Multiple accesses to a shared resource by different workers may raise race conditions, whereas data dependency happens when a worker needs the processed data of other workers.
- **Communication** Because data dependency is one of the main reasons for communication among workers, communication is always triggered when the intermediate data is sent to workers.
- **Scheduling** For a job or program, when the number of computation parts (tasks) or data pieces is more than the number of available workers, a scheduler selects a sequence of tasks or data pieces to be assigned to the workers. It is worth noting that the resource allocator performs the actual mapping of the computation or data pieces to workers, while the scheduler only picks the next part from the queue of unassigned tasks based on a set of rules called the scheduling policy. For multiple jobs or programs, a scheduler selects a sequence of jobs or programs to be run on the distributed computing system. In this case, scheduling is also necessary when system resources are not sufficient to simultaneously run multiple jobs or programs.

Map Reduce framework

6.2.2 MapReduce, Twister, and Iterative MapReduce

MapReduce, as introduced in Section 6.1.4, is a software framework which supports parallel and distributed computing on large data sets [27,37,45,46]. This software framework abstracts the data flow of running a parallel program on a distributed computing system by providing users with two interfaces in the form of two functions: *Map* and *Reduce*. Users can override these two functions to interact with and manipulate the data flow of running their programs. Figure 6.1 illustrates the logical data flow from the *Map* to the *Reduce* function in MapReduce frameworks. In this framework,

Cont..

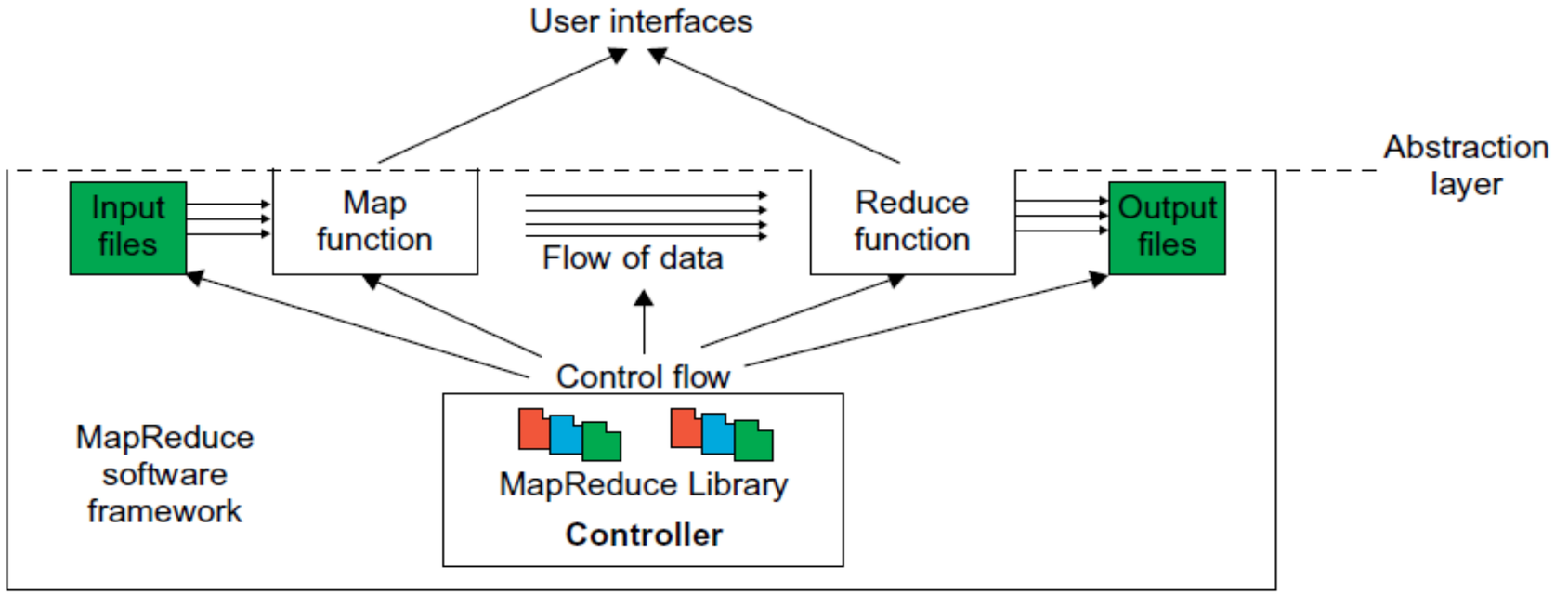


FIGURE 6.1

MapReduce framework: Input data flows through the Map and Reduce functions to generate the output result under the control flow using MapReduce software library. Special user interfaces are used to access the Map and Reduce resources.

Cont..

6.2.2.1 Formal Definition of MapReduce

The MapReduce software framework provides an abstraction layer with the data flow and flow of control to users, and hides the implementation of all data flow steps such as data partitioning, mapping, synchronization, communication, and scheduling. Here, although the data flow in such frameworks is predefined, the abstraction layer provides two well-defined interfaces in the form of two functions: *Map* and *Reduce* [47]. These two main functions can be overridden by the user to achieve specific objectives. Figure 6.1 shows the MapReduce framework with data flow and control flow.

Therefore, the user overrides the *Map* and *Reduce* functions first and then invokes the provided *MapReduce* (*Spec, & Results*) function from the library to start the flow of data. The MapReduce function, *MapReduce* (*Spec, & Results*), takes an important parameter which is a specification object, the *Spec*. This object is first initialized inside the user's program, and then the user writes code to fill it with the names of input and output files, as well as other optional tuning parameters. This object is also filled with the name of the *Map* and *Reduce* functions to identify these user-defined functions to the MapReduce library.

Cont..

The overall structure of a user's program containing the Map, Reduce, and the Main functions is given below. The Map and Reduce are two major subroutines. They will be called to implement the desired function performed in the main program.

```
Map Function ( . . . . )
  {
    . . . . . .
  }
Reduce Function ( . . . . )
  {
    . . . . . .
  }
Main Function ( . . . . )
  {
    Initialize Spec object
    . . . . . .
    MapReduce (Spec, & Results)
  }
```

Cont..

6.2.2.2 MapReduce Logical Data Flow

The input data to both the *Map* and the *Reduce* functions has a particular structure. This also pertains for the output data. The input data to the *Map* function is in the form of a (key, value) pair. For example, the key is the line offset within the input file and the value is the content of the line. The output data from the *Map* function is structured as (key, value) pairs called intermediate (key, value) pairs. In other words, the user-defined *Map* function processes each input (key, value) pair and produces a number of (zero, one, or more) intermediate (key, value) pairs. Here, the goal is to process all input (key, value) pairs to the *Map* function in parallel (Figure 6.2).

In turn, the *Reduce* function receives the intermediate (key, value) pairs in the form of a group of intermediate values associated with one intermediate key, *(key, [set of values])*. In fact, the

Cont..

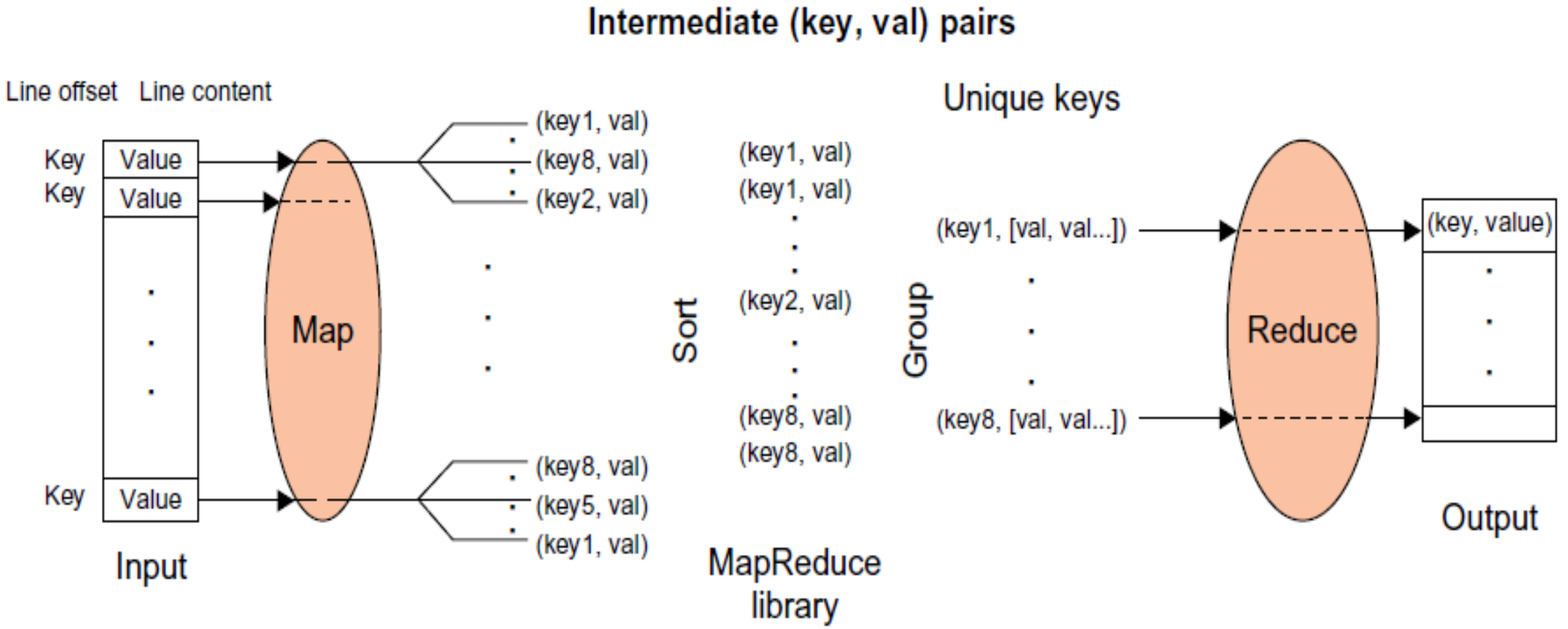


FIGURE 6.2

MapReduce logical data flow in 5 processing stages over successive (key, value) pairs.

Cont..

6.2.2.3 Formal Notation of MapReduce Data Flow

The *Map* function is applied in parallel to every input (key, value) pair, and produces new set of intermediate (key, value) pairs [37] as follows:

$$(key_1, val_1) \xrightarrow{Map\ Function} \text{List}\,(key_2, val_2) \tag{6.1}$$

Then the MapReduce library collects all the produced intermediate (key, value) pairs from all input (key, value) pairs, and sorts them based on the "key" part. It then groups the values of all occurrences of the same key. Finally, the *Reduce* function is applied in parallel to each group producing the collection of values as output as illustrated here:

$$(key_2, \text{List}\,(val_2)) \xrightarrow{Reduce\ Function} \text{List}\,(val_2) \tag{6.2}$$

Cont..

6.2.2.4 Strategy to Solve MapReduce Problems

As mentioned earlier, after grouping all the intermediate data, the values of all occurrences of the same key are sorted and grouped together. As a result, after grouping, each key becomes unique in all intermediate data. Therefore, finding unique keys is the starting point to solving a typical MapReduce problem. Then the intermediate (key, value) pairs as the output of the *Map* function will be automatically found. The following three examples explain how to define keys and values in such problems:

Problem 1: Counting the number of occurrences of each word in a collection of documents
Solution: unique "key": each word, intermediate "value": number of occurrences
Problem 2: Counting the number of occurrences of words having the same size, or the same number of letters, in a collection of documents
Solution: unique "key": each word, intermediate "value": size of the word
Problem 3: Counting the number of occurrences of anagrams in a collection of documents. Anagrams are words with the same set of letters but in a different order (e.g., the words "listen" and "silent").
Solution: unique "key": alphabetically sorted sequence of letters for each word (e.g., "eilnst"), intermediate "value": number of occurrences

Programming Support for Google App Engine

6.3.1 Programming the Google App Engine

Several web resources (e.g., http://code.google.com/appengine/) and specific books and articles (e.g., www.byteonic.com/2009/overview-of-java-support-in-google-app-engine/) discuss how to program GAE. Figure 6.17 summarizes some key features of GAE programming model for two supported languages: Java and Python. A client environment that includes an *Eclipse* plug-in for Java allows you to debug your GAE on your local machine. Also, the GWT Google Web Toolkit is available for Java web application developers. Developers can use this, or any other language using a JVM-based interpreter or compiler, such as JavaScript or Ruby. Python is often used with frameworks such as Django and CherryPy, but Google also supplies a built in *webapp* Python environment.

There are several powerful constructs for storing and accessing data. The data store is a NOSQL data management system for entities that can be, at most, 1 MB in size and are labeled by a set of schema-less properties. Queries can retrieve entities of a given kind filtered and sorted by the values of the properties. Java offers Java Data Object (JDO) and Java Persistence API (JPA) interfaces implemented by the open source Data Nucleus Access platform, while Python has a SQL-like query language called GQL. The data store is strongly consistent and uses optimistic concurrency control.

Cont..

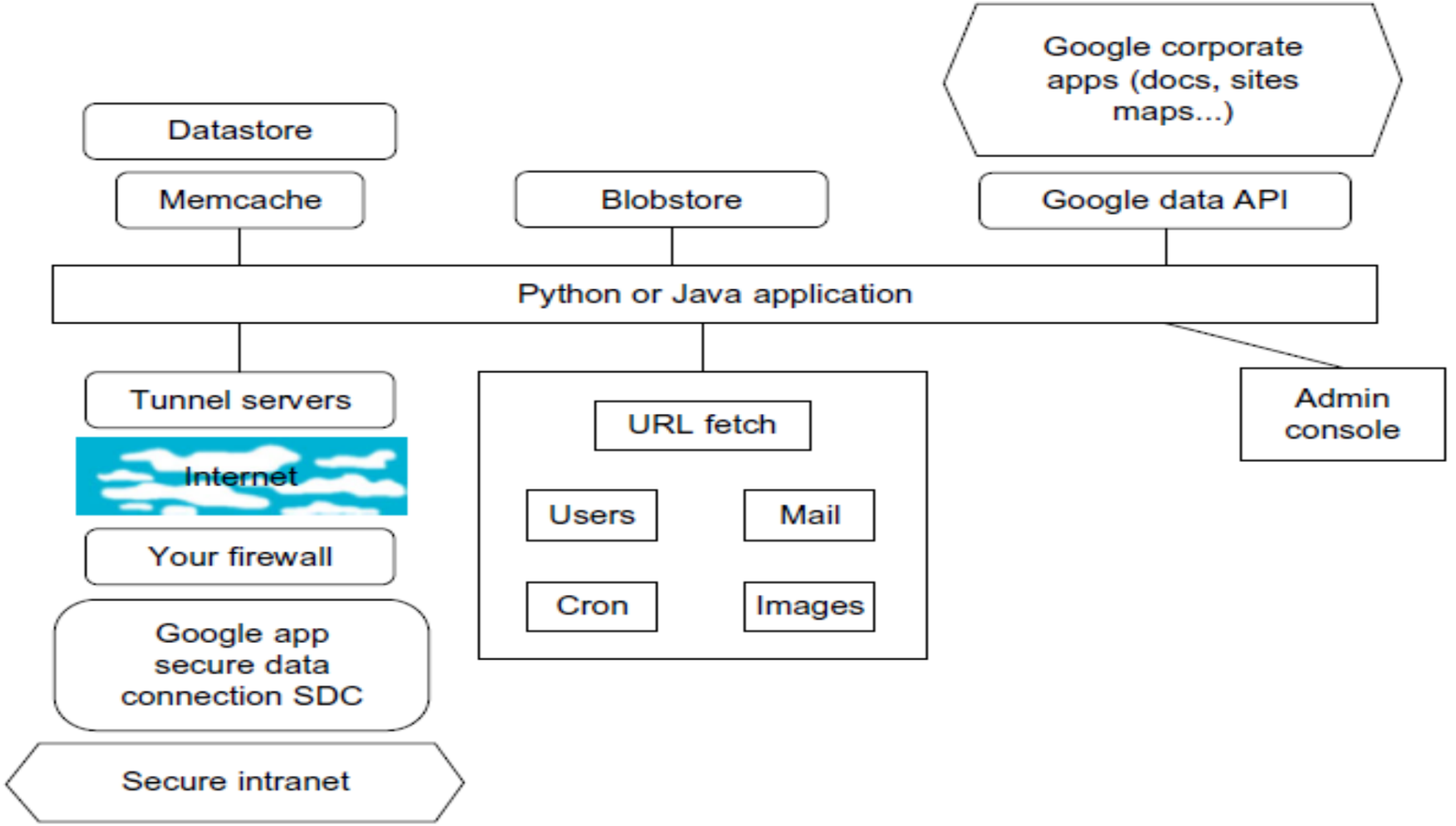


FIGURE 6.17

Programming environment for Google AppEngine.

Cont..

An update of an entity occurs in a transaction that is retried a fixed number of times if other processes are trying to update the same entity simultaneously. Your application can execute multiple data store operations in a single transaction which either all succeed or all fail together. The data store implements transactions across its distributed network using "entity groups." A transaction manipulates entities within a single group. Entities of the same group are stored together for efficient execution of transactions. Your GAE application can assign entities to groups when the entities are created. The performance of the data store can be enhanced by in-memory caching using the *memcache*, which can also be used independently of the data store.

Recently, Google added the *blobstore* which is suitable for large files as its size limit is 2 GB. There are several mechanisms for incorporating external resources. The *Google SDC Secure Data Connection* can tunnel through the Internet and link your intranet to an external GAE application. The *URL Fetch* operation provides the ability for applications to fetch resources and communicate with other hosts over the Internet using HTTP and HTTPS requests. There is a specialized mail mechanism to send e-mail from your GAE application.

Applications can access resources on the Internet, such as web services or other data, using GAE's URL fetch service. The URL fetch service retrieves web resources using the same high-speed Google infrastructure that retrieves web pages for many other Google products. There are dozens of Google "corporate" facilities including maps, sites, groups, calendar, docs, and YouTube, among others. These support the *Google Data API* which can be used inside GAE.

Cont..

An application can use Google Accounts for *user* authentication. Google Accounts handles user account creation and sign-in, and a user that already has a Google account (such as a Gmail account) can use that account with your app. GAE provides the ability to manipulate image data using a dedicated *Images* service which can resize, rotate, flip, crop, and enhance images. An application can perform tasks outside of responding to web requests. Your application can perform these tasks on a schedule that you configure, such as on a daily or hourly basis using "cron jobs," handled by the *Cron* service.

Alternatively, the application can perform tasks added to a queue by the application itself, such as a background task created while handling a request. A GAE application is configured to consume resources up to certain limits or quotas. With quotas, GAE ensures that your application won't exceed your budget, and that other applications running on GAE won't impact the performance of your app. In particular, GAE use is free up to certain quotas.

Programming on AWS

6.4.1 Programming on Amazon EC2

Amazon was the first company to introduce VMs in application hosting. Customers can rent VMs instead of physical machines to run their own applications. By using VMs, customers can load any software of their choice. The elastic feature of such a service is that a customer can create, launch, and terminate server instances as needed, paying by the hour for active servers. Amazon provides several types of preinstalled VMs. Instances are often called *Amazon Machine Images (AMIs)* which are preconfigured with operating systems based on Linux or Windows, and additional software.

Table 6.12 defines three types of AMI. Figure 6.24 shows an execution environment. AMIs are the templates for instances, which are running VMs. The workflow to create a VM is

$$\text{Create an AMI} \rightarrow \text{Create Key Pair} \rightarrow \text{Configure Firewall} \rightarrow \text{Launch} \tag{6.3}$$

Cont..

Table 6.12 Three Types of AMI

Image Type	AMI Definition
Private AMI	Images created by you, which are private by default. You can grant access to other users to launch your private images.
Public AMI	Images created by users and released to the AWS community, so anyone can launch instances based on them and use them any way they like. AWS lists all public images at http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171.
Paid QAMI	You can create images providing specific functions that can be launched by anyone willing to pay you per each hour of usage on top of Amazon's charges.

Cont..

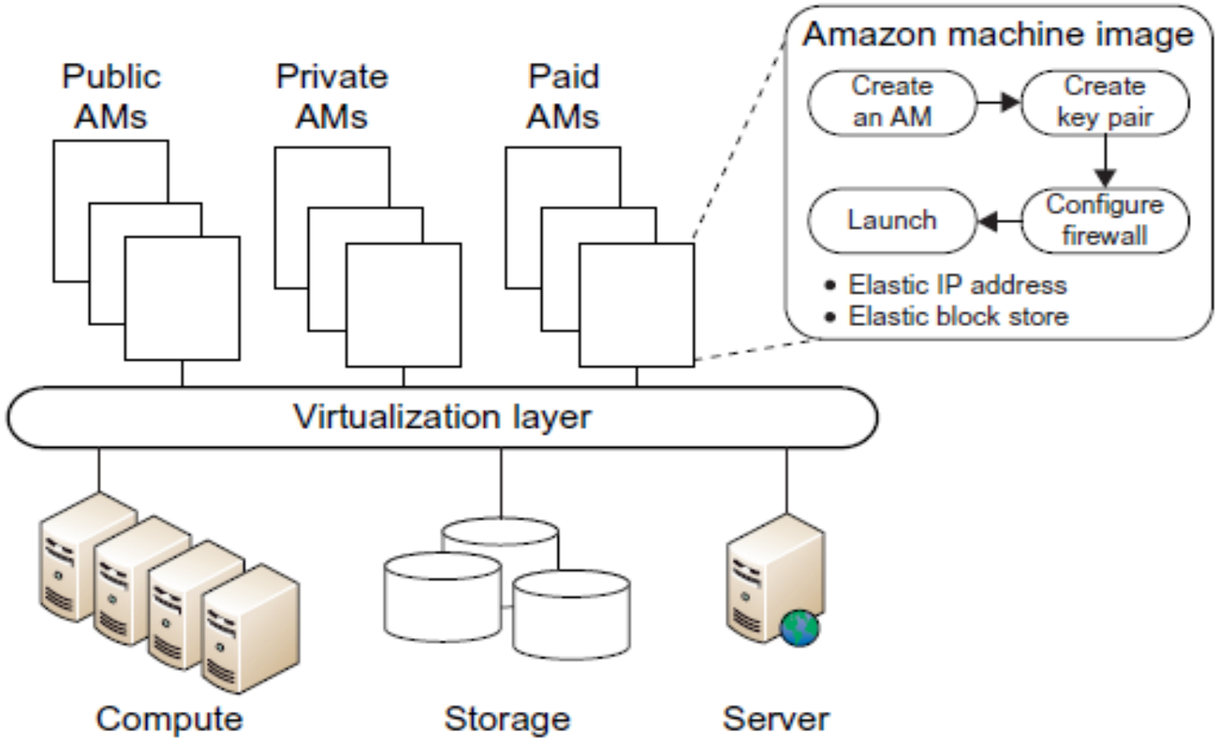


FIGURE 6.23

Amazon EC2 execution environment.

Cont..

6.4.2 Amazon Simple Storage Service (S3)

Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. S3 provides the object-oriented storage service for users. Users can access their objects through *Simple Object Access Protocol* (SOAP) with either browsers or other client programs which support SOAP. SQS is responsible for ensuring a reliable message service between two processes, even if the receiver processes are not running. Figure 6.24 shows the S3 execution environment.

Cont..

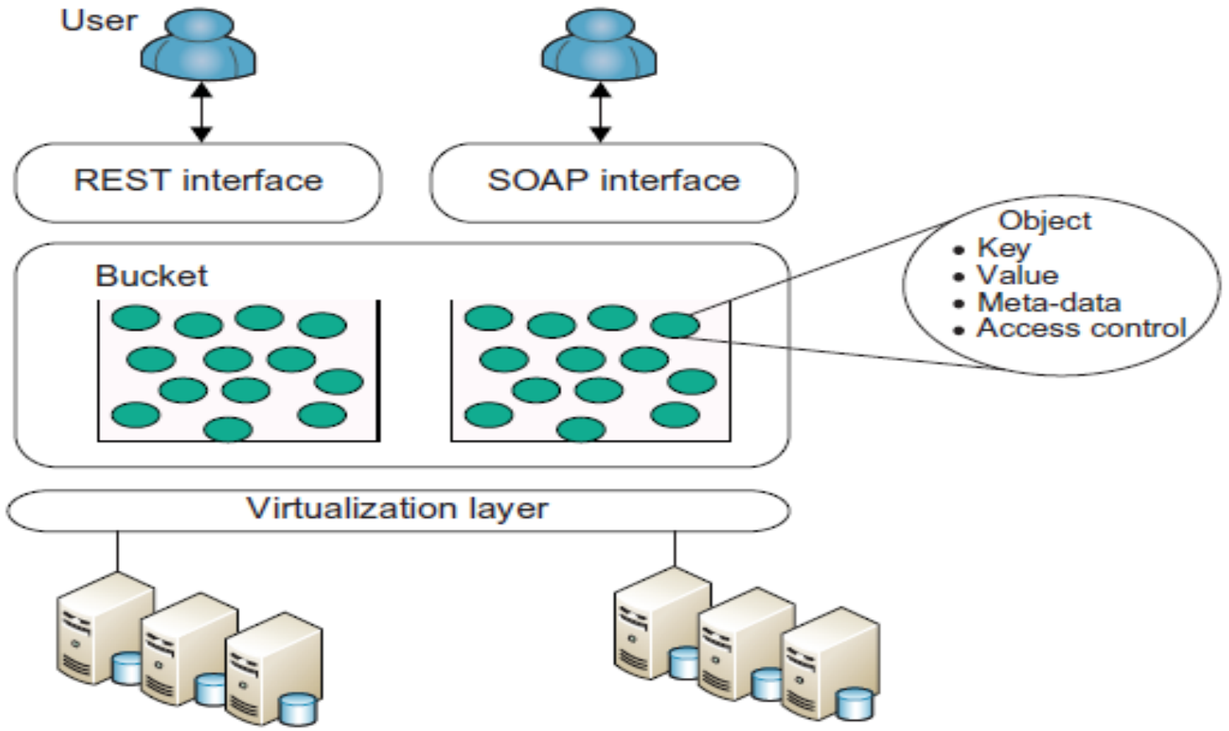


FIGURE 6.24

Amazon S3 execution environment.

Programming on Microsoft Azure

6.4.4 Microsoft Azure Programming Support

Section 4.4.4 has introduced the Azure cloud system. In this section, we describe the programming model in more detail. Some key programming components, including the client development environment, SQLAzure, and the rich storage and programming subsystems, are shown in Figure 6.25. We focus on the features of importance in developing Azure programs. First we have the underlying Azure fabric consisting of virtualized hardware together with a sophisticated control environment implementing dynamic assignment of resources and fault tolerance. This implements *domain name system* (DNS) and monitoring capabilities. Automated service management allows service models to be defined by an XML template and multiple service copies to be instantiated on request.

When the system is running, services are monitored and one can access event logs, trace/debug data, performance counters, IIS web server logs, crash dumps, and other log files. This information can be saved in Azure storage. Note that there is no debugging capability for running cloud applications, but debugging is done from a trace. One can divide the basic features into *storage* and *compute* capabilities. The Azure application is linked to the Internet through a customized compute VM called a *web role* supporting basic Microsoft web hosting. Such configured VMs are often called *appliances*. The other important compute class is the *worker role* reflecting the importance in cloud computing of a pool of compute resources that are scheduled as needed. The roles support HTTP(S) and TCP. Roles offer the following methods:

Cont..

-

- The *OnStart()* method which is called by the Fabric on startup, and allows you to perform initialization tasks. It reports a Busy status to the load balancer until you return *true*.
- The *OnStop()* method which is called when the role is to be shut down and gives a graceful exit.
- The *Run()* method which contains the main logic.

Cont..

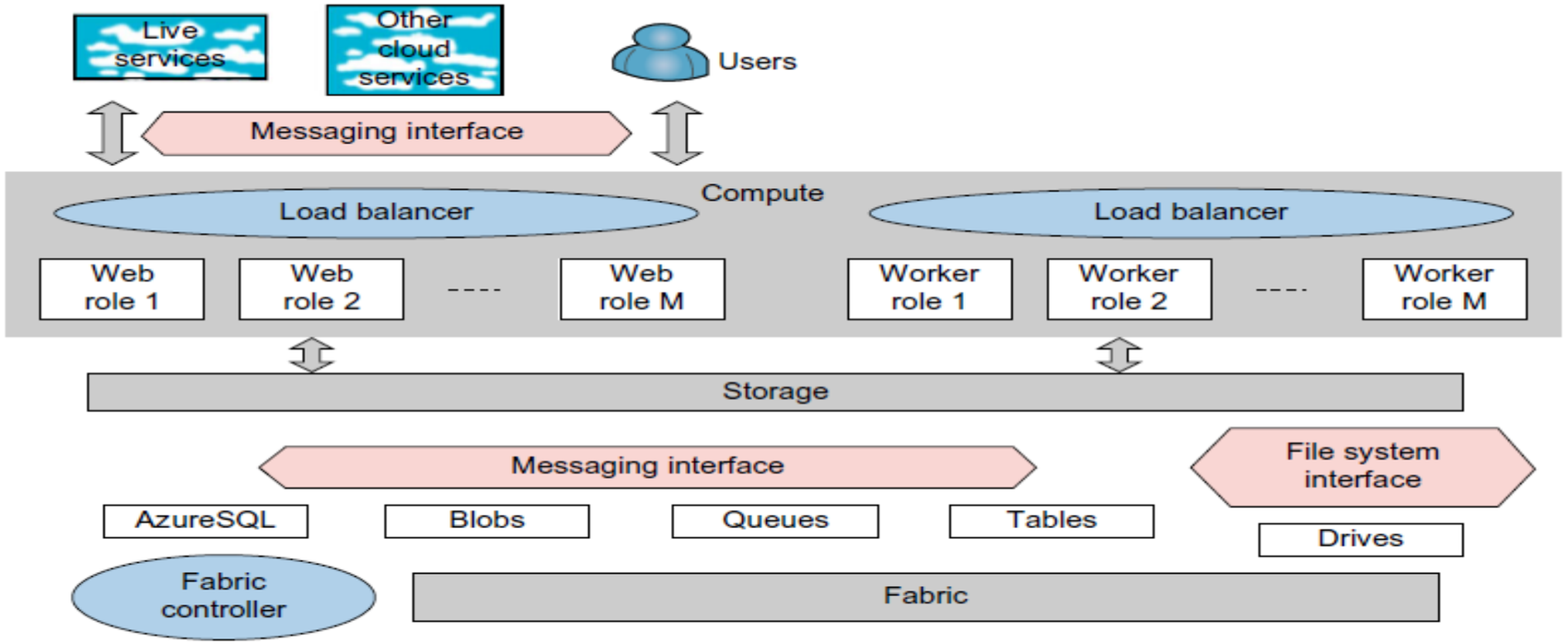


FIGURE 6.25

Features of the Azure cloud platform.

Emerging Cloud Software Environments

6.5 EMERGING CLOUD SOFTWARE ENVIRONMENTS

In this section, we will assess popular cloud operating systems and emerging software environments. We cover the open source Eucalyptus and Nimbus, then examine OpenNebula, Sector/Sphere, and Open Stack. These environments were introduced in Chapter 3 from a virtualization perspective. Here, we provide details regarding programming requirements. We will also cover the Aneka cloud programming tools recently developed at the University of Melbourne.

6.5.1 Open Source Eucalyptus and Nimbus

Eucalyptus is a product from Eucalyptus Systems (www.eucalyptus.com) that was developed out of a research project at the University of California, Santa Barbara. Eucalyptus was initially aimed at bringing the cloud computing paradigm to academic supercomputers and clusters. Eucalyptus provides an AWS-compliant EC2-based web service interface for interacting with the cloud service. Additionally, Eucalyptus provides services, such as the AWS-compliant Walrus, and a user interface for managing users and images.

Cont..

6.5.1.2 VM Image Management

Eucalyptus takes many design queues from Amazon's EC2, and its image management system is no different. Eucalyptus stores images in Walrus, the block storage system that is analogous to the Amazon S3 service. As such, any user can bundle her own root file system, and upload and then register this image and link it with a particular kernel and ramdisk image. This image is uploaded into a user-defined bucket within Walrus, and can be retrieved anytime from any availability zone. This allows users to create specialty virtual appliances (http://en.wikipedia.org/wiki/Virtual_appliance) and deploy them within Eucalyptus with ease. The Eucalyptus system is available in a commercial proprietary version, as well as the open source version we just described.

6.5.1.3 Nimbus

Nimbus [81,82] is a set of open source tools that together provide an IaaS cloud computing solution. Figure 6.27 shows the architecture of Nimbus, which allows a client to lease remote resources by deploying VMs on those resources and configuring them to represent the environment desired

Cont..

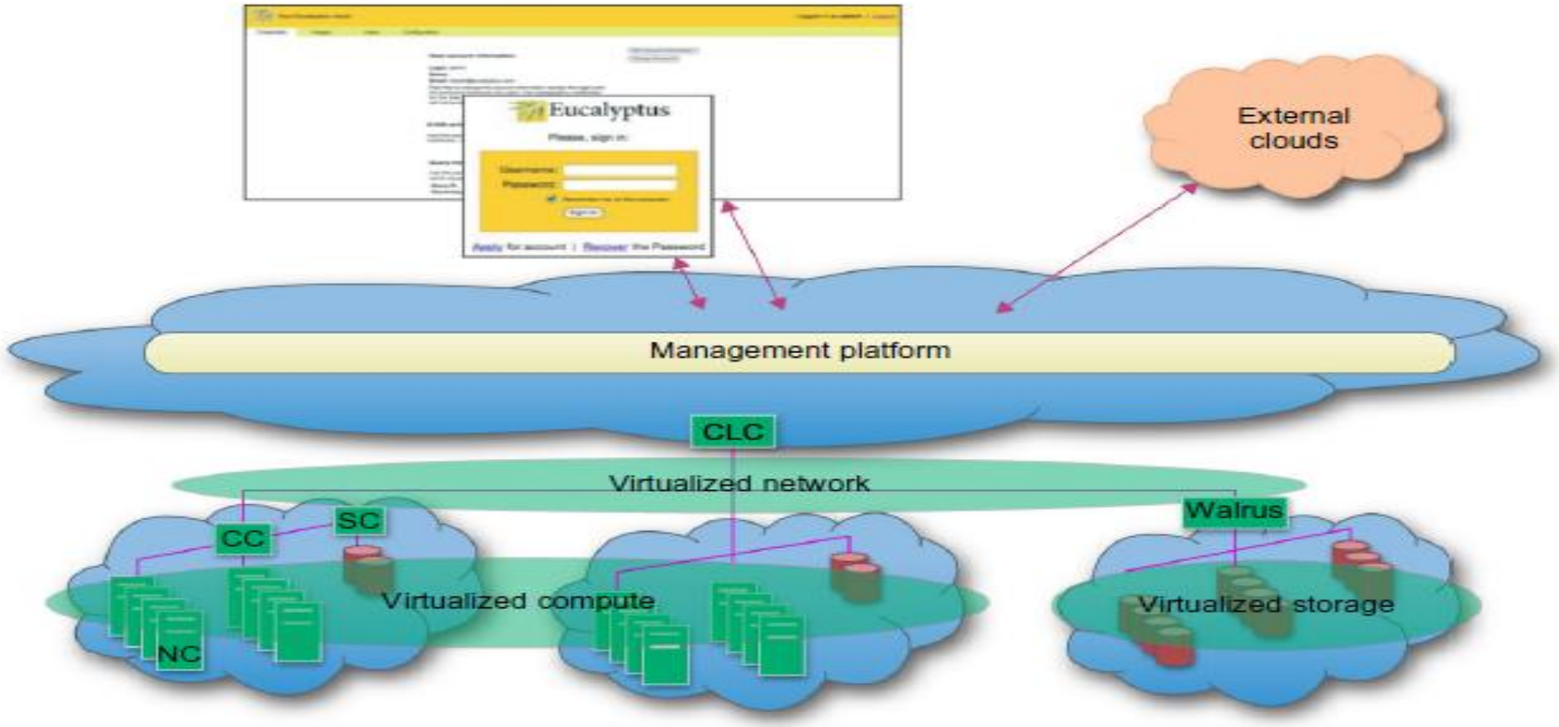


FIGURE 6.26

The Eucalyptus architecture for VM image management.

Cont..

(Courtesy of Eucalyptus LLC [81])

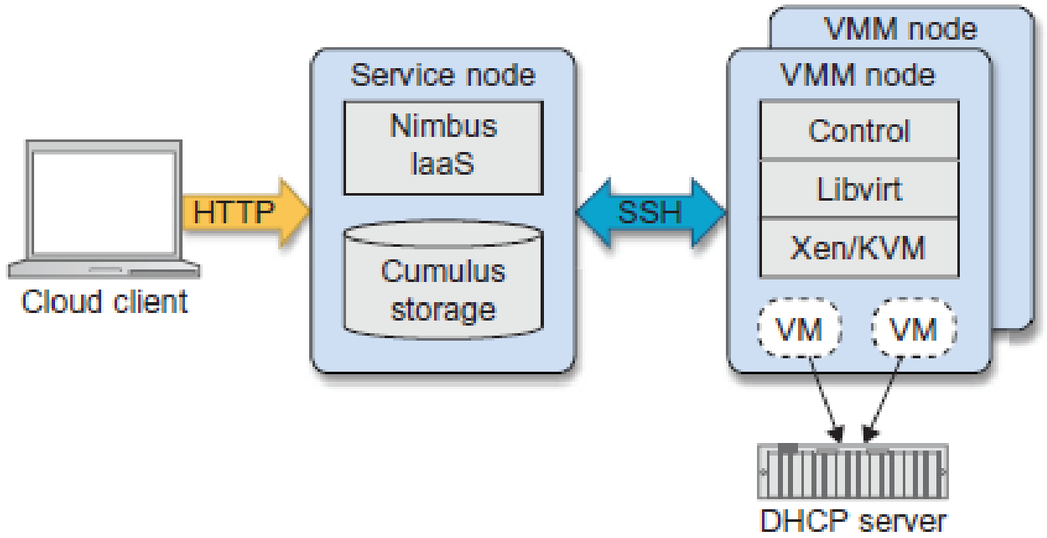


FIGURE 6.27

Nimbus cloud infrastructure.

(Courtesy of Nimbus Project [82])

Cont..

6.5.2.2 OpenStack

OpenStack [103] was been introduced by Rackspace and NASA in July 2010. The project is building an open source community spanning technologists, developers, researchers, and industry to share resources and technologies with the goal of creating a massively scalable and secure cloud infrastructure. In the tradition of other open source projects, the software is open source and limited to just open source APIs such as Amazon.