UNIT III

Cloud Platform Architecture:

Cloud Computing and service Models, Architectural Design of Compute and Storage Clouds, Public Cloud Platforms, Inter Cloud Resource Management, Cloud Security and Trust Management. Service Oriented Architecture, Message Oriented Middleware

4.1 CLOUD COMPUTING AND SERVICE MODELS

Users can access and deploy cloud applications from anywhere in the world at very competitive costs. Virtualized cloud platforms are often built on top of large data centers. In other words, clouds aim to power the next generation of data centers by architecting them as virtual resources over automated hardware, databases, user interfaces, and application environments. In this sense, clouds grow out of the desire to build better data centers through automated resource provisioning.

4.1.1 Public, Private, and Hybrid Clouds

The concept of cloud computing has evolved from cluster, grid, and utility computing. Cluster and grid computing leverage the use of many computers in parallel to solve problems of any size. Utility and Software as a Service (SaaS) provide computing resources as a service with the notion of pay per use.

Cloud computing leverages dynamic resources to deliver large numbers of services to end users. Cloud computing is a high-throughput computing (HTC) paradigm whereby the infrastructure provides the services through a large data center or server farms. The cloud computing model enables users to share access to resources from anywhere at any time through their connected devices. Furthermore, machine virtualization has enhanced resource utilization, increased application flexibility, and reduced the total cost of using virtualized data-center resources. The main idea is to move desktop computing to a service-oriented platform using server clusters and huge databases at data centers. Cloud computing leverages its low cost and simplicity to both providers and users.

4.1.1.1 Centralized versus Distributed Computing

Some people argue that cloud computing is centralized computing at data centers. Others claim that cloud computing is the practice of distributed parallel computing over data-center resources. All computations in cloud applications are distributed to servers in a data center. These are mainly virtual machines (VMs) in virtual clusters created out of data-center resources. In this sense, cloud platforms are systems distributed through virtualization.

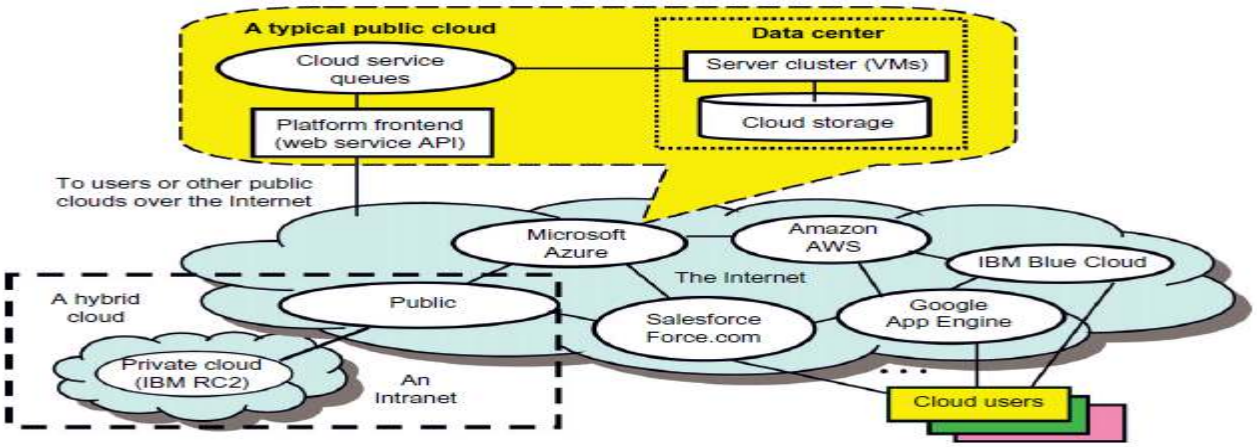

FIGURE 4.1

Public, private, and hybrid clouds illustrated by functional architecture and connectivity of representative clouds available by 2011.

As Figure 4.1 shows, both public clouds and private clouds are developed in the Internet. As many clouds are generated by commercial providers or by enterprises in a distributed manner, they will be interconnected over the Internet to achieve scalable and efficient computing services. Commercial cloud providers such as Amazon, Google, and Microsoft created their platforms to be distributed geographically. This distribution is partially attributed to fault tolerance, response latency reduction, and even legal reasons. Intranet-based private clouds are linked to public clouds to get additional resources.

4.1.1.2 Public Clouds

A public cloud is built over the Internet and can be accessed by any user who has paid for the service. Public clouds are owned by service providers and are accessible through a subscription. The callout box in top of Figure 4.1 shows the architecture of a typical public cloud. Many public clouds are available, including Google App Engine (GAE), Amazon Web Services (AWS), Microsoft Azure, IBM Blue Cloud, and Salesforce.com's Force.com. The providers of the aforementioned clouds are commercial providers that offer a publicly accessible remote interface for creating and managing VM instances within their proprietary infrastructure. A public cloud delivers a selected set of business processes.

4.1.1.3 Private Clouds

A private cloud is built within the domain of an intranet owned by a single organization. Therefore, it is client owned and managed, and its access is limited to the own clients and their partners. Private clouds give local users a flexible and alert private infrastructure to run service workloads within their administrative domains. A private cloud is supposed to deliver more efficient and convenient cloud services. It may impact the cloud standardization, while retaining greater customization and organizational control.

4.1.1.4 Hybrid Clouds

A hybrid cloud is built with both public and private clouds, as shown at the lower-left corner of Figure 4.1. Private clouds can also support a hybrid cloud model by supplementing local infrastructure with computing

capacity from an external public cloud. A hybrid cloud provides access to clients, the partner network, and third parties. In summary, public clouds promote standardization, preserve capital investment, and offer application flexibility. Private clouds attempt to achieve customization and offer higher efficiency, resiliency, security, and privacy. Hybrid clouds operate in the middle, with many compromises in terms of resource sharing.

4.1.1.5 Data-Center Networking Structure

The core of a cloud is the server cluster (or VM cluster). Cluster nodes are used as compute nodes. A few control nodes are used to manage and monitor cloud activities. The scheduling of user jobs requires that you assign work to virtual clusters created for users. The gateway nodes provide the access points of the service from the outside world. These gateway nodes can be also used for security control of the entire cloud platform. In physical clusters and traditional grids, users expect static demand of resources. Clouds are designed to handle fluctuating workloads, and thus demand variable resources dynamically. Private clouds will satisfy this demand if properly designed and managed. Data centers and supercomputers have some similarities as well as fundamental differences. In the case of data centers, scaling is a fundamental require-ment. Data-center server clusters are typically built with large number of servers, ranging from thousands to millions of servers (nodes). Data centers and supercomputers also differ in networking requirements, as illustrated in Figure 4.2 which shows a multilayer structure for accessing the Internet. The server racks are at the bottom Layer 2, and they are connected through fast switches (S) as the hardware core. The data center is connected to the Internet at Layer 3 with many access routers (ARs) and border routers (BRs). These cloud models demand different levels of performance, data protection, and security enforcement. In this case, different SLAs may be applied to satisfy both providers and paid users. Cloud computing exploits many existing technologies.

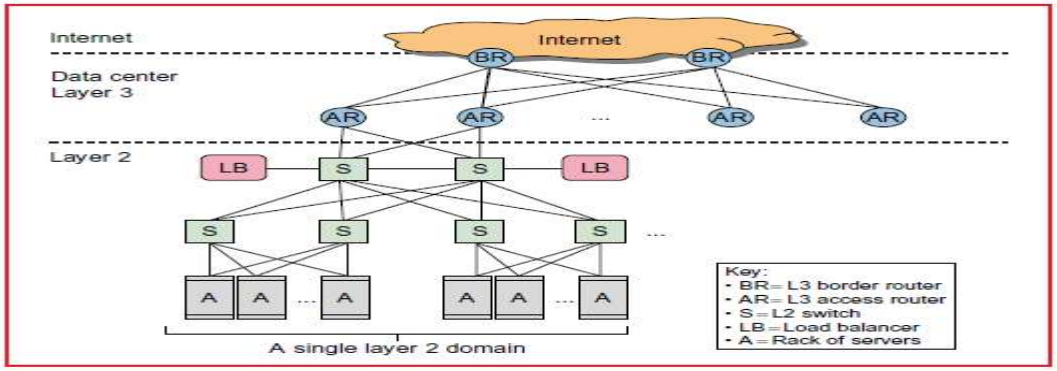

FIGURE 4.2
Standard data-center networking for the cloud to access the Internet.

4.1.1.6 Cloud Development Trends

Although most clouds built in 2010 are large public clouds, the authors believe private clouds will grow much faster than public clouds in the future. Private clouds are easier to secure and more trustworthy within a company or organization. Once private clouds become mature and better secured, they could be open or converted to public clouds. Therefore, the boundary between public and private clouds could be blurred in the future. Most likely, most future clouds will be hybrid in nature.

4.1.2 Cloud Ecosystem and Enabling Technologies

Cloud computing platforms differ from conventional computing platforms in many aspects. The traditional computing model is specified below by the process on the left, which involves buying the hardware, acquiring the necessary system software, installing the system, testing the configuration and executing the application code and management of resources. The cloud computing paradigm is shown on the right. This computing model follows a pay-as-you-go model. Therefore the cost is significantly reduced, because we simply rent computer resources without buying the computer in advance. All hardware and software resources are leased from the cloud provider without capital investment on the part of the users. Only the execution phase costs some money.

Classical Computing	**Cloud Computing**
(Repeat the following cycle every 18 months)	(Pay as you go per each service provided)
Buy and own	**Subscribe**
Hardware, system software, applications to meet peak needs	- - - -
Install, configure, test, verify, evaluate, manage	**Use** (Save about 80-95% of the total cost)
- - - -	- - - -
Use	(Finally)
- - - -	**$ - Pay for what you use**
Pay $$$$$ (High cost)	based on the QoS

4.1.2.1 Cloud Design Objectives

The following list highlights six design objectives for cloud computing:

• Shifting computing from desktops to data centers Computer processing, storage, and software delivery is shifted away from desktops and local servers and toward data centers over the Internet.

• Service provisioning and cloud economics Providers supply cloud services by signing SLAs with consumers and end users. The services must be efficient in terms of computing, storage, and power consumption. Pricing is based on a pay-as-you-go policy.

• Scalability in performance The cloud platforms and software and infrastructure services must be able to scale in performance as the number of users increases.

• Data privacy protection Can you trust data centers to handle your private data and records? This concern must be addressed to make clouds successful as trusted services.

• High quality of cloud services The QoS of cloud computing must be standardized to make clouds interoperable among multiple providers.

• New standards and interfaces This refers to solving the data lock-in problem associated with data centers or cloud providers. Universally accepted APIs and access protocols are needed to provide high portability and flexibility of virtualized applications.

4.1.2.2 Cost Model

In traditional IT computing, users must acquire their own computer and peripheral equipment as capital expenses. In addition, they have to face operational expenditures in operating and maintaining the computer systems, including personnel and service costs.

Figure 4.3(a) shows the addition of variable operational costs on top of fixed capital investments in traditional IT. Note that the fixed cost is the main cost, and that it could be reduced slightly as the number of users increases. However, the operational costs may increase sharply with a larger number of users. Therefore, the total cost escalates quickly with massive numbers of users. On the other hand, cloud computing applies a pay-per-use business model, in which user jobs are outsourced to data centers. To use the cloud, one has no up-front cost in hardware acquisitions. Only variable costs are experienced by cloud users, as demonstrated in Figure 4.3(b).

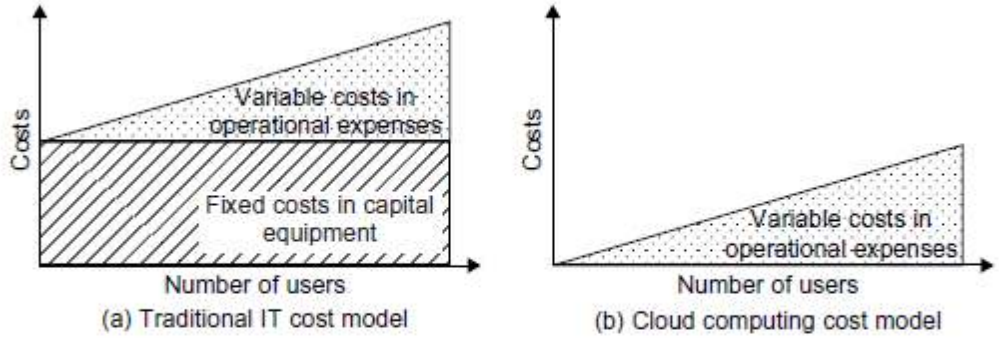

FIGURE 4.3

Computing economics between traditional IT users and cloud users.

4.1.2.3 Cloud Ecosystems

With the emergence of various Internet clouds, an ecosystem of providers, users, and technologies has appeared. This ecosystem has evolved around public clouds. Strong interest is growing in open source cloud computing tools that let organizations build their own IaaS clouds using their internal infrastructures. Private and hybrid clouds are not exclusive, since public clouds are involved in both cloud types. A private/hybrid cloud allows remote access to its resources over the Internet using remote web service interfaces such as that used in Amazon EC2.

They suggested four levels of ecosystem development in a private cloud. At the user end, consumers demand a flexible platform. At the cloud management level, the cloud manager provides virtualized resources over an IaaS platform. At the virtual infrastructure (VI) management level, the manager allocates VMs over multiple server clusters. Finally, at the VM management level, the VM managers handle VMs installed on individual host machines. An ecosystem of cloud tools attempts to span both cloud management and VI management. Integrating these two layers is complicated by the lack of open and standard interfaces between them.

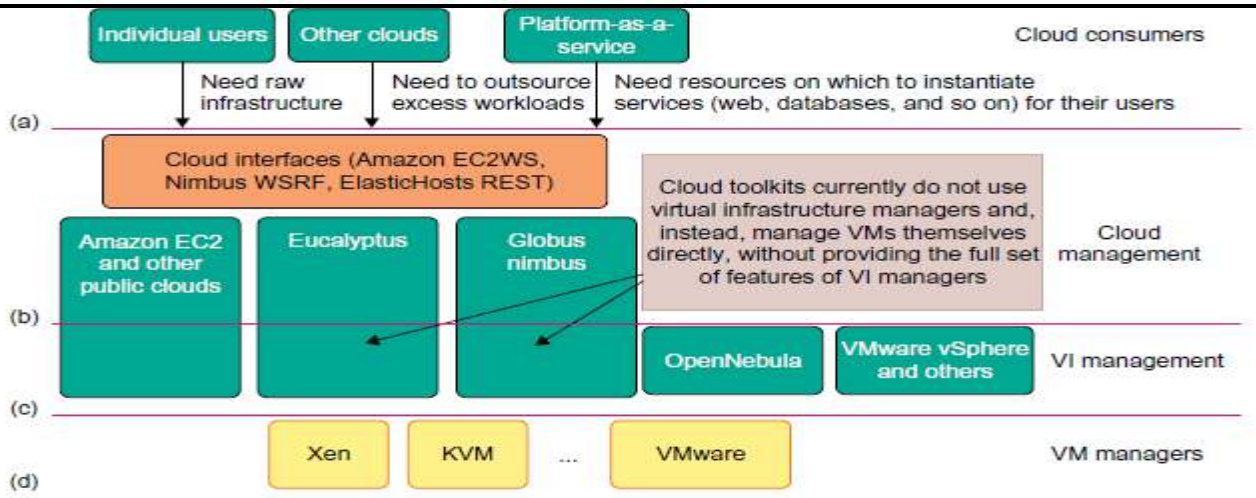

FIGURE 4.4

Cloud ecosystem for building private clouds: (a) Consumers demand a flexible platform; (b) Cloud manager provides virtualized resources over an IaaS platform; (c) VI manager allocates VMs; (d) VM managers handle VMs installed on servers.

An increasing number of startup companies are now basing their IT strategies on cloud resources, spending little or no capital to manage their own IT infrastructures. We desire a flexible and open architecture that enables organizations to build private/hybrid clouds. VI management is aimed at this goal.

4.1.3 Infrastructure-as-a-Service (IaaS)

Cloud computing delivers infrastructure, platform, and software (application) as services, which are made available as subscription-based services in a pay-as-you-go model to consumers. The services provided over the cloud can be generally categorized into three different service models: namely IaaS, Platform as a Service (PaaS), and Software as a Service (SaaS). These form the three pillars on top of which cloud computing solutions are delivered to end users. All three models allow users to access services over the Internet, relying entirely on the infrastructures of cloud service providers. These models are offered based on various SLAs between providers and users. In a broad sense, the SLA for cloud computing is addressed in terms of service availability, performance, and data protection and security.

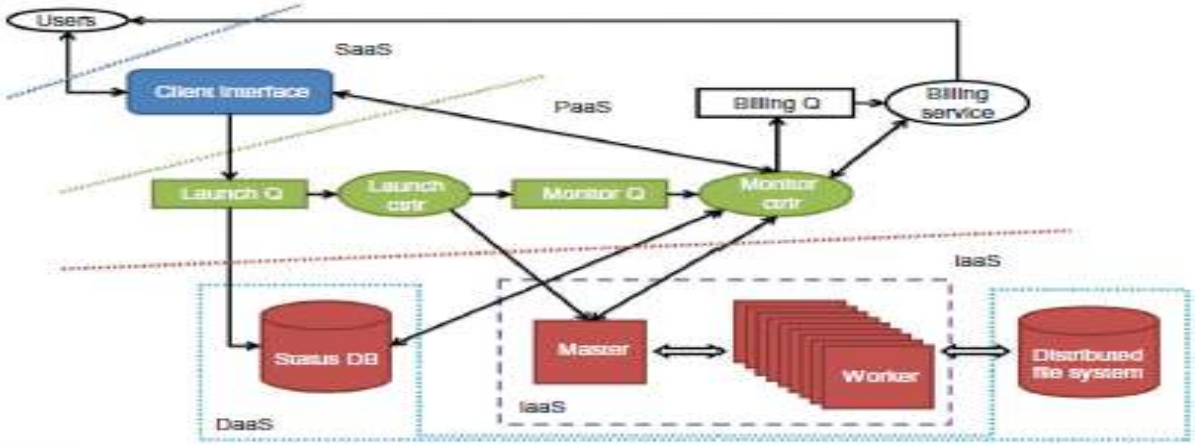

FIGURE 4.5

The IaaS, PaaS, and SaaS cloud service models at different service levels.

Figure 4.5 illustrates three cloud models at different service levels of the cloud. SaaS is applied at the application end using special interfaces by users or clients. At the PaaS layer, the cloud platform must perform billing services and handle job queuing, launching, and monitoring services. At the bottom layer of the IaaS services, databases, compute instances, the file system, and storage must be provisioned to satisfy user demands.

4.1.3.1 Infrastructure as a Service

This model allows users to use virtualized IT resources for computing, storage, and networking. In short, the service is performed by rented cloud infrastructure. The user can deploy and run his applications over his chosen OS environment. The user does not manage or control the underlying cloud infrastructure, but has control over the OS, storage, deployed applications, and possibly select networking components. This IaaS model encompasses storage as a service, compute instances as a service, and communication as a service.

4.1.4 Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS)

In this section, we will introduce the PaaS and SaaS models for cloud computing. SaaS is often built on top of the PaaS, which is in turn built on top of the IaaS.

4.1.4.1 Platform as a Service (PaaS)

To be able to develop, deploy, and manage the execution of applications using provisioned resources demands a cloud platform with the proper software environment. Such a platform includes operating system and runtime library support. This has triggered the creation of the PaaS model to enable users to develop and deploy their user applications. Table 4.2 highlights cloud platform services offered by five PaaS services.

Table 4.2 Five Public Cloud Offerings of PaaS [10,18]

Cloud Name	Languages and Developer Tools	Programming Models Supported by Provider	Target Applications and Storage Option
Google App Engine	Python, Java, and Eclipse-based IDE	MapReduce, web programming on demand	Web applications and BigTable storage
Salesforce.com's Force.com	Apex, Eclipse-based IDE, web-based Wizard	Workflow, Excel-like formula, Web programming on demand	Business applications such as CRM
Microsoft Azure	.NET, Azure tools for MS Visual Studio	Unrestricted model	Enterprise and web applications
Amazon Elastic MapReduce	Hive, Pig, Cascading, Java, Ruby, Perl, Python, PHP, R, C++	MapReduce	Data processing and e-commerce
Aneka	.NET, stand-alone SDK	Threads, task, MapReduce	.NET enterprise applications, HPC

The platform cloud is an integrated computer system consisting of both hardware and software infra-structure. The user application can be developed on this virtualized cloud platform using some programming languages and software tools supported by the provider (e.g., Java, Python, .NET). The user does not manage the underlying cloud infrastructure. The cloud provider supports user application development and testing on a well-defined service platform. This PaaS model enables a collaborated software development platform for users

from different parts of the world. This model also encourages third parties to provide software management, integration, and service monitoring solutions.

4.1.4.2 Software as a Service (SaaS)

This refers to browser-initiated application software over thousands of cloud customers. Services and tools offered by PaaS are utilized in construction of applications and management of their deployment on resources offered by IaaS providers. The SaaS model provides software applications as a service. As a result, on the customer side, there is no upfront investment in servers or software licensing. On the provider side, costs are kept rather low, compared with conventional hosting of user applications. Customer data is stored in the cloud that is either vendor proprietary or publicly hosted to support PaaS and IaaS.

4.1.4.3 Mashup of Cloud Services

At the time of this writing, public clouds are in use by a growing number of users. Due to the lack of trust in leaking sensitive data in the business world, more and more enterprises, organizations, and communities are developing private clouds that demand deep customization. An enterprise cloud is used by multiple users within an organization. Each user may build some strategic applications on the cloud, and demands customized partitioning of the data, logic, and database in the metadata representation. Cloud mashups have resulted from the need to use multiple clouds simultaneously or in sequence. Some public repository provides thousands of service APIs and mashups for web commerce services. Popular APIs are provided by Google Maps, Twitter, YouTube, Amazon e-Commerce, Salesforce.com, etc.

4.3 ARCHITECTURAL DESIGN OF COMPUTE AND STORAGE CLOUDS

4.3.1 A Generic Cloud Architecture Design

An Internet cloud is envisioned as a public cluster of servers provisioned on demand to perform collective web services or distributed applications using data-center resources.

4.3.1.1 Cloud Platform Design Goals

Scalability, virtualization, efficiency, and reliability are four major design goals of a cloud computing platform. Clouds support Web 2.0 applications. Cloud management receives the user request, finds the correct resources, and then calls the provisioning services which invoke the resources in the cloud. The cloud management software needs to support both physical and virtual machines. Security in shared resources and shared access of data centers also pose another design challenge.

4.3.1.2 Enabling Technologies for Clouds

The key driving forces behind cloud computing is the ubiquity of broadband and wireless networking, falling storage costs, and progressive improvements in Internet computing software. Cloud users are able to demand more capacity at peak demand, reduce costs, experiment with new services, and remove unneeded capacity, whereas service providers can increase system utilization via multiplexing, virtualization, and dynamic resource provisioning.

The technologies play instrumental roles in making cloud computing a reality. Most of the technologies are mature today to meet increasing demand. In the hardware area, the rapid progress in multicore CPUs, memory chips, and disk arrays has made it possible to build faster data centers with huge amounts of storage space. Resource virtualization enables rapid cloud deployment and disaster recovery. Service-oriented architecture (SOA) also plays a vital role. Progress in providing SaaS, Web 2.0 standards and Internet performance have all contributed to the emergence of cloud services. Today's clouds are designed to serve a large number of tenants over massive volumes of data.

4.3.1.3 A Generic Cloud Architecture

Figure 4.14 shows security-aware cloud architecture. The Internet cloud is envisioned as a massive cluster of servers. These servers are provisioned on demand to perform collective web services or distributed applications using data-center resources. The cloud platform is formed dynamically by provisioning or de-provisioning servers, software, and database resources. Servers in the cloud can be physical machines or VMs. User interfaces are applied to request services. In addition to building the server cluster, the cloud platform demands distributed storage and accompanying services. The cloud computing resources are built into the data centers, which are typically owned and operated by a third-party provider. Consumers do not need to know the underlying technologies. In a cloud, software becomes a service. The cloud demands a high degree of trust of massive amounts of data retrieved from large data centers. We need to build a framework to process large-scale data stored in the storage system. This demands a distributed file system over the database system. Other cloud resources are added into a cloud platform, including storage area networks (SANs), database systems, firewalls, and security devices. Web service providers offer special APIs that enable developers to exploit Internet clouds. Monitoring and metering units are used to track the usage and performance of provisioned resources

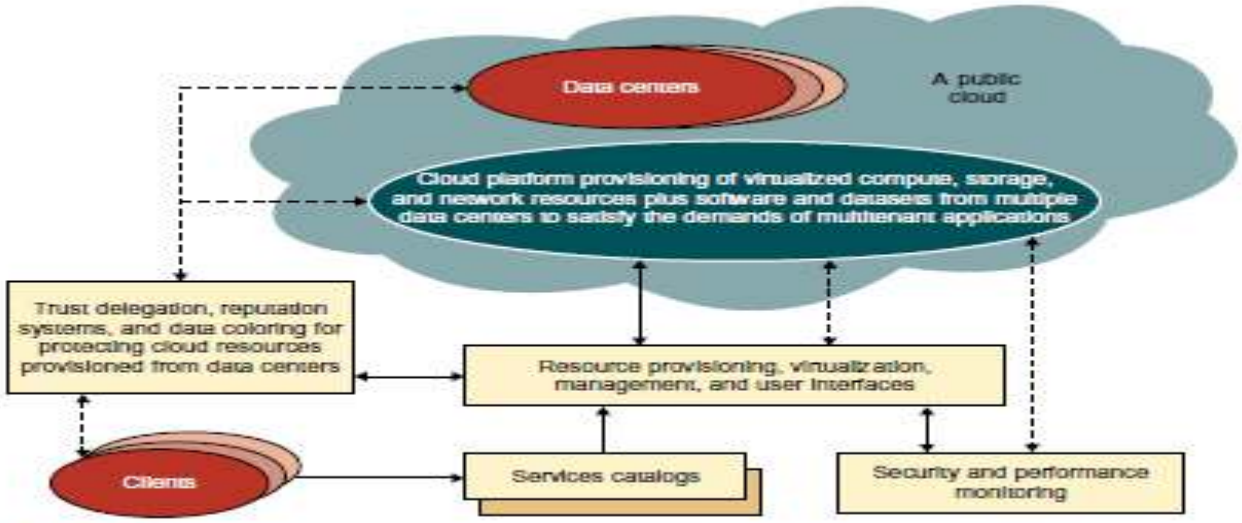

FIGURE 4.14

A security-aware cloud platform built with a virtual cluster of VMs, storage, and networking resources over the data-center servers operated by providers.

The software infrastructure of a cloud platform must handle all resource management and do most of the maintenance automatically. Software must detect the status of each node server joining and leaving, and

perform relevant tasks accordingly. The cloud physical platform builder is more concerned about the performance/price ratio and reliability issues than shear speed performance. In general, private clouds are easier to manage, and public clouds are easier to access. Security becomes a critical issue in safeguarding the operation of all cloud types.

4.3.2 Layered Cloud Architectural Development

The architecture of a cloud is developed at three layers: infrastructure, platform, and application, as demonstrated in Figure 4.15. These three development layers are implemented with virtualization and standardization of hardware and software resources provisioned in the cloud. The services to public, private, and hybrid clouds are conveyed to users through networking support over the Internet and intranets involved. Different types of cloud services demand application of these resources separately.

The infrastructure layer is built with virtualized compute, storage, and network resources. The abstraction of these hardware resources is meant to provide the flexibility demanded by users. Internally, virtualization realizes automated provisioning of resources and optimizes the infrastructure management process. The platform layer is for general-purpose and repeated usage of the collection of software resources. This layer provides users with an environment to develop their applications, to test operation flows, and to monitor execution results and performance. The platform should be able to assure users that they have scalability, dependability, and security protection. The application layer is formed with a collection of all needed software modules for SaaS applications. Service applications in this layer include daily office management work, such as information retrieval, document processing, and calendar and authentication services. It should be noted that not all cloud services are restricted to a single layer. Many applications may apply resources at mixed layers. From the provider's perspective, the services at various layers demand different amounts of functionality support and resource management by providers. In general, SaaS demands the most work from the provider, PaaS is in the middle, and IaaS demands the least.

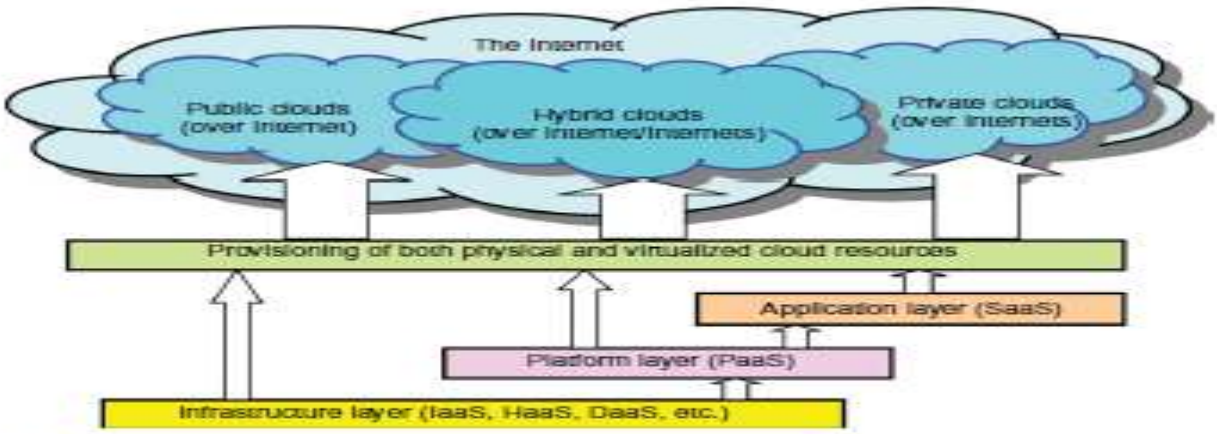

FIGURE 4.15
Layered architectural development of the cloud platform for IaaS, PaaS, and SaaS applications over the Internet.

4.3.2.1 Market-Oriented Cloud Architecture

As consumers rely on cloud providers to meet more of their computing needs, they will require a specific level of QoS to be maintained by their providers, in order to meet their objectives and sustain their operations. Cloud providers consider and meet the different QoS parameters of each individual consumer as negotiated in specific SLAs. To achieve this, market-oriented resource management is necessary to regulate the supply and demand of cloud resources to achieve market equilibrium between supply and demand.

The designer needs to provide feedback on economic incentives for both consumers and providers to promote QoS-based resource allocation mechanisms. Figure 4.16 shows the high-level architecture for supporting market-oriented resource allocation in a cloud computing environment. This cloud is basically built with the following entities:

Users or brokers acting on user's behalf submit service requests from anywhere in the world to the data center and cloud to be processed. The SLA resource allocator acts as the interface between the data center /cloud service provider and external users/brokers. It requires the interaction of the following mechanisms to support SLA-oriented resource management. When a service request is first submitted the service request examiner interprets the submitted request for QoS requirements before determining whether to accept or reject the request.

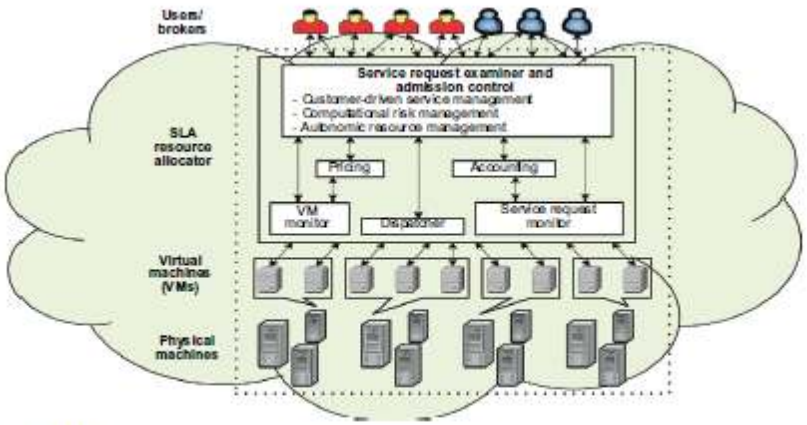

FIGURE 4.16
Market-oriented cloud architecture to expand/shrink leasing of resources with variation in QoS/demand from users.

The request examiner ensures that there is no overloading of resources whereby many service requests cannot be fulfilled successfully due to limited resources. It also needs the latest status information regarding resource availability (from the VM Monitor mechanism) and workload processing (from the Service Request Monitor mechanism) in order to make resource allocation decisions effectively. Then it assigns requests to VMs and determines resource entitlements for allocated VMs.

The VM Monitor mechanism keeps track of the availability of VMs and their resource entitlements. The Dispatcher mechanism starts the execution of accepted service requests on allocated VMs. The Service Request Monitor mechanism keeps track of the execution progress of service requests. Multiple VMs can be started and stopped on demand on a single physical machine to meet accepted service requests, hence providing maximum flexibility to configure various partitions of resources on the same physical machine to different specific

requirements of service requests. In addition, multiple VMs can concurrently run applications based on different operating system environments on a single physical machine since the VMs are isolated from one another on the same physical machine.

4.3.2.2 Quality of Service Factors

The data center comprises multiple computing servers that provide resources to meet service demands. In the case of a cloud as a commercial offering to enable crucial business operations of companies, there are critical QoS parameters to consider in a service request, such as time, cost, reliability, and trust/security. In particular, QoS requirements cannot be static and may change over time due to continuing changes in business operations and operating environments. In short, there should be greater importance on customers since they pay to access services in clouds. In addition, the state of the art in cloud computing has no or limited support for dynamic negotiation of SLAs between participants and mechanisms for automatic allocation of resources to multiple competing requests. Negotiation mechanisms are needed to respond to alternate offers protocol for establishing SLAs.

4.3.3 Virtualization Support and Disaster Recovery

Virtualization of servers on a shared cluster can consolidate web services. As the VMs are the containers of cloud services, the provisioning tools will first find the corresponding physical machines and deploy the VMs to those nodes before scheduling the service to run on the virtual nodes. In addition, in cloud computing, virtualization also means the resources and fundamental infrastructure are virtualized. The user will not care about the computing resources that are used for providing the services. Cloud users do not need to know and have no way to discover physical resources that are involved while processing a service request. Also, application developers do not care about some infrastructure issues such as scalability and fault tolerance (i.e., they are virtualized). Application developers focus on service logic. Figure 4.17 shows the infrastructure needed to virtualize the servers in a data center for implementing specific cloud applications.

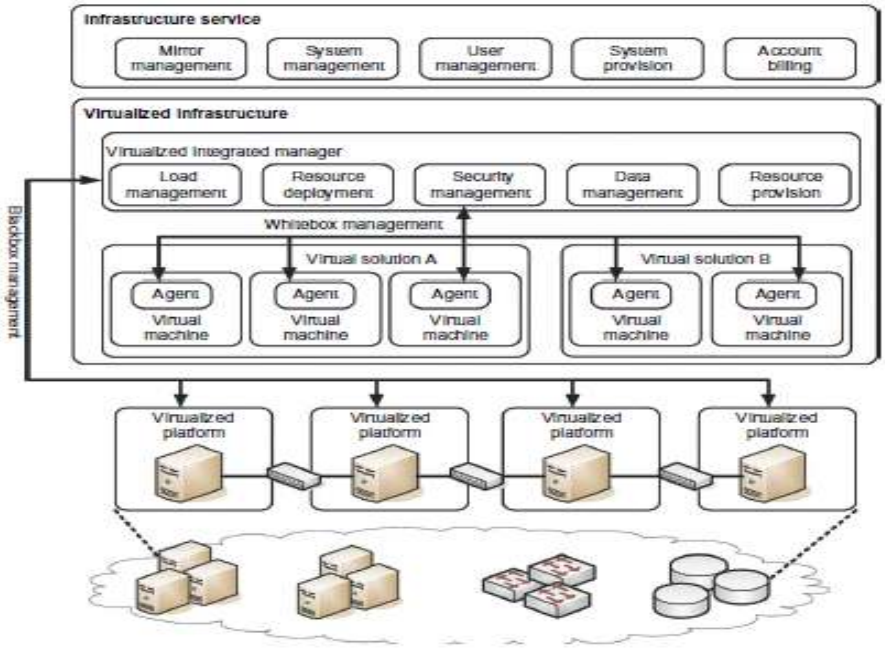

FIGURE 4.17
Virtualized servers, storage, and network for cloud platform construction.

4.3.3.1 Hardware Virtualization

In many cloud computing systems, virtualization software is used to virtualize the hardware. System virtualization software is a special kind of software which simulates the execution of hardware and runs even unmodified operating systems. Cloud computing systems use virtualization software as the running environment for legacy software such as old operating systems and unusual applications. Virtualization software is also used as the platform for developing new cloud applications that enable developers to use any operating systems and programming environments they like. The development environment and deployment environment can now be the same, which eliminates some runtime problems. Some cloud computing providers have used virtualization technology to provide this service for developers. As mentioned before, system virtualization software is considered the hardware analog mechanism to run an unmodified operating system, usually on bare hardware directly, on top of software. Table 4.4 lists some of the system virtualization software in wide use at the time of this writing. Currently, the VMs installed on a cloud computing platform are mainly used for hosting third-party programs. VMs provide flexible runtime services to free users from worrying about the system environment.

Using VMs in a cloud computing platform ensures extreme flexibility for users. As the computing resources are shared by many users, a method is required to maximize the users' privileges and still keep them separated safely. Traditional sharing of cluster resources depends on the user and group mechanism on a system. Such sharing is not flexible. Users cannot customize the system for their special purposes. Operating systems cannot be changed. The separation is not complete.

Table 4.4 Virtualized Resources in Compute, Storage, and Network Clouds [4]

Provider	AWS	Microsoft Azure	GAE
Compute cloud with virtual cluster of servers	x86 instruction set, Xen VMs, resource elasticity allows scalability through virtual cluster, or a third party such as RightScale must provide the cluster	Common language runtime VMs provisioned by declarative descriptions	Predefined application framework handlers written in Python, automatic scaling up and down, server failover inconsistent with the web applications
Storage cloud with virtual storage	Models for block store (EBS) and augmented key/blob store (SimpleDB), automatic scaling varies from EBS to fully automatic (SimpleDB, S3)	SQL Data Services (restricted view of SQL Server), Azure storage service	MegaStore/BigTable
Network cloud services	Declarative IP-level topology; placement details hidden, security groups restricting communication, availability zones isolate network failure, elastic IP applied	Automatic with user's declarative descriptions or roles of app. components	Fixed topology to accommodate three-tier web app. structure, scaling up and down is automatic and programmer-invisible

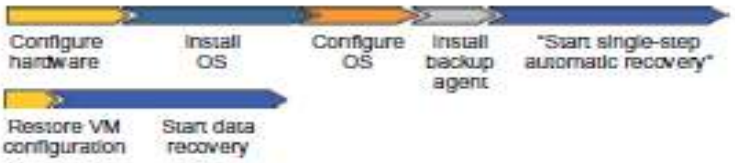

FIGURE 4.18

Recovery overhead of a conventional disaster recovery scheme, compared with that required to recover from live migration of VMs.

An environment that meets one user's requirements often cannot satisfy another user. Virtualization allows users to have full privileges while keeping them separate.

Users have full access to their own VMs, which are completely separate from other users' VMs. Multiple VMs can be mounted on the same physical server. Different VMs may run with different OSes. We also need to establish the virtual disk storage and virtual networks needed by the VMs. The virtualized resources form a resource pool. The virtualization is carried out by special servers dedicated to generating the virtualized resource pool. The virtualized infrastructure (black box in the middle) is built with many virtualizing integration managers. These managers handle loads, resources, security, data, and provisioning functions. Figure 4.18 shows two VM platforms. Each platform carries out a virtual solution to a user job. All cloud services are managed in the boxes at the top.

4.3.3.4 Virtualization for IaaS

VM technology has increased in ubiquity. This has enabled users to create customized environments atop physical infrastructure for cloud computing. Use of VMs in clouds has the following distinct benefits: (1) System administrators consolidate workloads of underutilized servers in fewer servers; (2) VMs have the ability to run legacy code without interfering with other APIs; (3) VMs can be used to improve security through creation of sandboxes for running applications with questionable reliability; And (4) virtualized cloud platforms can apply performance isolation, letting providers offer some guarantees and better QoS to customer applications.

4.3.4 Architectural Design Challenges

4.3.4.1 Challenge 1—Service Availability and Data Lock-in Problem

To achieve HA, one can consider using multiple cloud providers . Even if a company has multiple data centers located in different geographic regions, it may have common software infrastructure and accounting systems. Therefore, using multiple cloud providers may provide more protection from failures. Another availability obstacle is distributed denial of service (DDoS) attacks. Criminals threaten to cut off the incomes of SaaS providers by making their services unavailable. Software stacks have improved interoperability among different cloud platforms, but the APIs itself are still proprietary. Thus, customers cannot easily extract their data and programs from one site to run on another. The obvious solution is to standardize the APIs so that a SaaS developer can deploy services and data across multiple cloud providers. This will rescue the loss of all data due to the failure of a single company.

4.3.4.2 Challenge 2—Data Privacy and Security Concerns

Current cloud offerings are essentially public (rather than private) networks, exposing the system to more attacks. Many obstacles can be overcome immediately with well-understood technologies such as encrypted storage, virtual LANs, and network middle boxes (e.g., firewalls, packet filters). In a cloud environment, newer attacks may result from hypervisor malware, guest hopping and hijacking, or VM root kits. Another type of attack is the man-in-the-middle attack for VM migrations. In general, passive attacks steal sensitive data or passwords. Active attacks may manipulate kernel data structures which will cause major damage to cloud servers.

4.3.4.3 Challenge 3—Unpredictable Performance and Bottlenecks

Multiple VMs can share CPUs and main memory in cloud computing, but I/O sharing is problematic. Internet applications continue to become more data-intensive. If we assume applications to be “pulled apart” across the boundaries of clouds, this may complicate data placement and transport. Cloud users and providers have to think about the implications of placement and traffic at every level of the system, if they want to minimize costs. This kind of reasoning can be seen in Amazon’s development of its new CloudFront service. Therefore, data transfer bottlenecks must be removed, bottleneck links must be widened, and weak servers should be removed.

4.3.4.4 Challenge 4—Distributed Storage and Widespread Software Bugs

The database is always growing in cloud applications. The opportunity is to create a storage system that will not only meet this growth, but also combine it with the cloud advantage of scaling arbitrarily up and down on demand. This demands the design of efficient distributed SANs. Data consistence checking in SAN-connected data centers is a major challenge in cloud computing. Large-scale distributed bugs cannot be reproduced, so the debugging must occur at a scale in the production data centers. No data center will provide such a convenience. One solution may be a trust on using VMs in cloud computing. The level of virtualization may make it possible to capture valuable information in ways that are impossible without using VMs.

4.3.4.5 Challenge 5—Cloud Scalability, Interoperability, and Standardization

The pay-as-you-go model applies to storage and network bandwidth; both are counted in terms of the number of bytes used. Computation is different depending on virtualization level. Open Virtualization Format (OVF) describes an open, secure, portable, efficient, and extensible format for the packaging and distribution of VMs. It also defines a format for distributing software to be deployed in VMs. This VM format does not rely on the use of a specific host platform, virtualization platform, or guest operating system. The approach is to address virtual platform-agnostic packaging with certification and integrity of packaged software. The package supports virtual appliances to span more than one VM. OVF also defines a transport mechanism for VM templates, and can apply to different virtualization platforms with different levels of virtualization. In terms of cloud standardization, we suggest the ability for virtual appliances to run on any virtual platform. We also need to enable VMs to run on heterogeneous hardware platform hypervisors. This requires hypervisor-agnostic VMs.

4.3.4.6 Challenge 6—Software Licensing and Reputation Sharing

Many cloud computing providers originally relied on open source software because the licensing model for commercial software is not ideal for utility computing. The primary opportunity is either for open source to remain popular or simply for commercial software companies to change their licensing structure to better fit cloud computing. One can consider using both pay-for-use and bulk-use licensing schemes to widen the business coverage. Another legal issue concerns the transfer of legal liability. Cloud providers want legal liability to remain with the customer, and vice versa. This problem must be solved at the SLA level.

4.4 PUBLIC CLOUD PLATFORMS: GAE, AWS, AND AZURE

4.4.1 Public Clouds and Service Offerings

Cloud services are demanded by computing and IT administrators, software vendors, and end users. Figure 4.19 introduces five levels of cloud players. At the top level, individual users and organizational users demand very different services. The application providers at the SaaS level serve mainly individual users. Most business organizations are serviced by IaaS and PaaS providers. The infrastructure services (IaaS) provide compute, storage, and communication resources to both applications and organizational users. The cloud environment is defined by the PaaS or platform providers. Note that the platform providers support both infrastructure services and organizational users directly.

Cloud services rely on new advances in machine virtualization, SOA, grid infrastructure management, and power efficiency. Consumers purchase such services in the form of IaaS, PaaS, or SaaS as described earlier. Also, many cloud entrepreneurs are selling value-added utility services to massive numbers of users. The cloud industry leverages the growing demand by many enterprises and business users to outsource their computing and storage jobs to professional providers. The provider service charges are often much lower than the cost for users to replace their obsolete servers frequently.

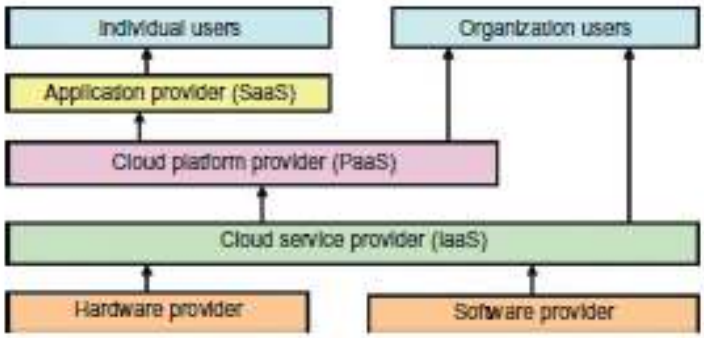

FIGURE 4.19

Roles of individual and organizational users and their interaction with cloud providers under various cloud service models.

4.4.2 Google App Engine (GAE)

Google has the world's largest search engine facilities. The company has extensive experience in massive data processing that has led to new insights into data-center design and novel programming models that scale to incredible sizes. The Google platform is based on its search engine expertise with MapReduce, this which is applicable to many other areas.

4.4.2.1 Google Cloud Infrastructure

Google has pioneered cloud development by leveraging the large number of data centers it operates. For example, Google pioneered cloud services in Gmail, Google Docs, and Google Earth, among other applications. These applications can support a large number of users simultaneously with HA. Notable technology achievements include the Google File System (GFS), MapReduce, BigTable, and Chubby. In 2008, Google announced the GAE web application platform which is becoming a common platform for many small cloud service providers. This platform specializes in supporting scalable web applications. GAE enables users to run their applications on a large number of data centers associated with Google's search engine operations.

4.4.2.2 GAE Architecture

Figure 4.20 shows the major building blocks of the Google cloud platform which has been used to deliver the cloud services highlighted earlier. GFS is used for storing large amounts of data. MapReduce is for use in application program development. Chubby is used for distributed application lock services. BigTable offers a storage service for accessing structured data.

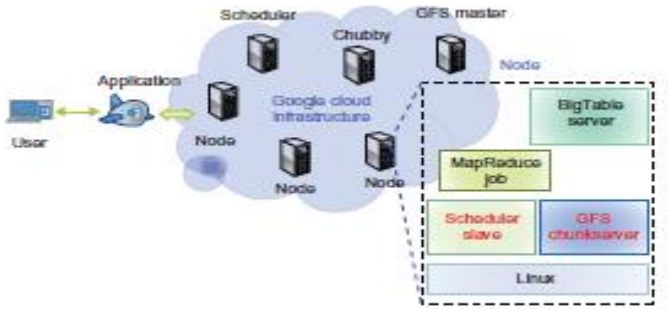

FIGURE 4.20

Google cloud platform and major building blocks, the blocks shown are large clusters of low-cost servers.

Users can interact with Google applications via the web interface provided by each application. Third-party application providers can use GAE to build cloud applications for providing services. The applications all run in data centers under tight management by Google engineers. Inside each data center, there are thousands of servers forming different clusters. The above figure shows the overall architecture of the Google cloud infrastructure. A typical cluster configuration can run the Google File System, MapReduce jobs, and BigTable servers for structure data. Extra services such as Chubby for distributed locks can also run in the clusters. GAE runs the user program on Google's infrastructure. As it is a platform running third-party programs, application developers now do not need to worry about the maintenance of servers. GAE can be thought of as the combination of several software components. The frontend is an application framework which is similar to other web application frameworks such as ASP, J2EE, and JSP. At the time of this writing, GAE supports Python and Java programming environments. The applications can run similar to web application containers. The frontend can be used as the dynamic web serving infrastructure which can provide the full support of common technologies.

4.4.2.3 Functional Modules of GAE

The GAE platform comprises the following five major components. The GAE is not an infrastructure platform, but rather an application development platform for users.

a. The datastore offers object-oriented, distributed, structured data storage services based on BigTable techniques. The datastore secures data management operations.

b. The application runtime environment offers a platform for scalable web programming and execution. It supports two development languages: Python and Java.

c. The software development kit (SDK) is used for local application development. The SDK allows users to execute test runs of local applications and upload application code.

d. The administration console is used for easy management of user application development cycles, instead of for physical resource management.

e. The GAE web service infrastructure provides special interfaces to guarantee flexible use and management of storage and network resources by GAE.

4.4.2.4 GAE Applications

Well-known GAE applications include the Google Search Engine, Google Docs, Google Earth, and Gmail. These applications can support large numbers of users simultaneously. Users can interact with Google applications via the web interface provided by each application. Third-party application providers can use GAE to build cloud applications for providing services. The applications are all run in the Google data centers. Inside each data center, there might be thousands of server nodes to form different clusters.

4.4.3 Amazon Web Services (AWS)

VMs can be used to share computing resources both flexibly and safely. Amazon has been a leader in providing public cloud services. Amazon applies the IaaS model in providing its services. Figure 4.21 shows the AWS architecture. EC2 provides the virtualized platforms to the host VMs where the cloud application can run. S3 (Simple Storage Service) provides the object-oriented storage service for users.

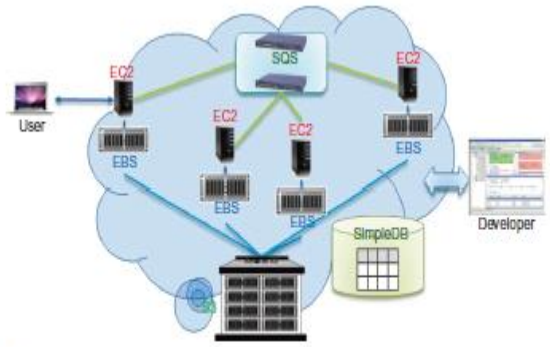

FIGURE 4.21
Amazon cloud computing infrastructure (Key services are identified here; many more are listed in Table 4.6).

Table 4.6 AWS Offerings in 2011

Service Area	Service Modules and Abbreviated Names
Compute	Elastic Compute Cloud (EC2), Elastic MapReduce, Auto Scaling
Messaging	Simple Queue Service (SQS), Simple Notification Service (SNS)
Storage	Simple Storage Service (S3), Elastic Block Storage (EBS), AWS Import/Export
Content Delivery	Amazon CloudFront
Monitoring	Amazon CloudWatch
Support	AWS Premium Support
Database	Amazon SimpleDB, Relational Database Service (RDS)
Networking	Virtual Private Cloud (VPC) (Example 4.1, Figure 4.6), Elastic Load Balancing
Web Traffic	Alexa Web Information Service, Alexa Web Sites
E-Commerce	Fulfillment Web Service (FWS)
Payments and Billing	Flexible Payments Service (FPS), Amazon DevPay
Workforce	Amazon Mechanical Turk

EBS (Elastic Block Service) provides the block storage interface which can be used to support traditional applications. SQS stands for Simple Queue Service, and its job is to ensure a reliable message service between two processes. The message can be kept reliably even when the receiver processes are not running. Users can access their objects through SOAP with either browsers or other client programs which support the SOAP standard. Table 4.6 summarizes the service offerings by AWS in 12 application tracks.

4.4.4 Microsoft Windows Azure

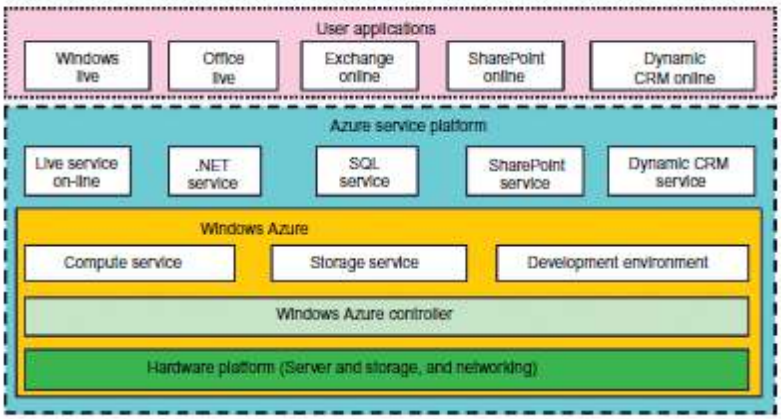

FIGURE 4.22

Microsoft Windows Azure platform for cloud computing.

In 2008, Microsoft launched a Windows Azure platform to meet the challenges in cloud computing. This platform is built over Microsoft data centers. Figure 4.22 shows the overall architecture of Microsoft's cloud platform. The platform is divided into three major component platforms. Windows Azure offers a cloud platform built on Windows OS and based on Microsoft virtualization technology. Applications are installed on VMs deployed on the data-center servers. Azure manages all servers, storage, and network resources of the data center. On top of the infrastructure are the various services for building different cloud applications. Cloud-level services provided by the Azure platform are introduced below.

• Live service Users can visit Microsoft Live applications and apply the data involved across multiple machines concurrently.

•.NET service This package supports application development on local hosts and execution on cloud machines associated with the SQL server in the cloud.

• SharePoint service This provides a scalable and manageable platform for users to develop their special business applications in upgraded web services.

• Dynamic CRM service This provides software developers a business platform in managing CRM applications in financing, marketing, and sales and promotions.

All these cloud services in Azure can interact with traditional Microsoft software applications, such as Windows Live, Office Live, Exchange online, SharePoint online, and dynamic CRM online. The Azure platform applies the standard web communication protocols SOAP and REST. The Azure service applications allow users to integrate the cloud application with other platforms or third-party clouds.

4.5 INTER-CLOUD RESOURCE MANAGEMENT

4.5.1 Extended Cloud Computing Services

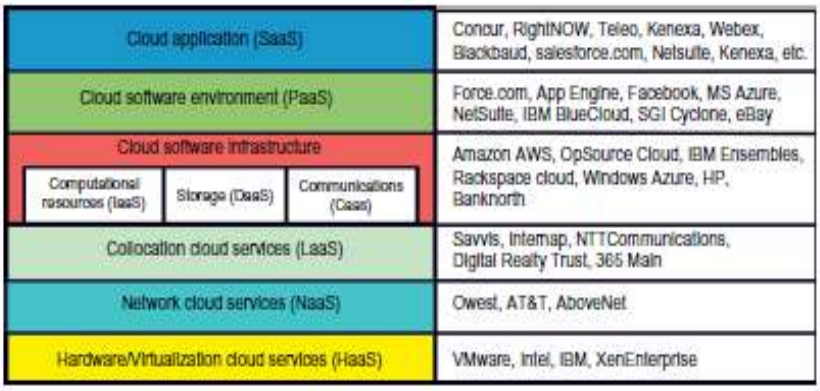

FIGURE 4.23

A stack of six layers of cloud services and their providers.

Table 4.7 Cloud Differences in Perspectives of Providers, Vendors, and Users

Cloud Players	IaaS	PaaS	SaaS
IT administrators/cloud providers	Monitor SLAs	Monitor SLAs and enable service platforms	Monitor SLAs and deploy software
Software developers (vendors)	To deploy and store data	Enabling platforms via configurators and APIs	Develop and deploy software
End users or business users	To deploy and store data	To develop and test web software	Use business software

Figure 4.23 shows six layers of cloud services, ranging from hardware, network, and collocation to infrastructure, platform, and software applications. The cloud platform provides PaaS, which sits on top of the IaaS infrastructure. The top layer offers SaaS. These must be implemented on the cloud platforms provided. Although the three basic models are dissimilar in usage, as shown in Table 4.7, they are built one on top of another. The implication is that one cannot launch SaaS applications with a cloud platform. The cloud platform cannot be built if compute and storage infrastructures are not there.

The bottom three layers are more related to physical requirements. The bottommost layer provides Hardware as a Service (HaaS). The next layer is for interconnecting all the hardware components, and is simply called Network as a Service (NaaS). Virtual LANs fall within the scope of NaaS. The next layer up offers Location as a Service (LaaS), which provides a collocation service to house, power, and secure all the physical hardware and network resources. The cloud infrastructure layer can be further subdivided as Data as a Service (DaaS) and Communication as a Service (CaaS) in addition to compute and storage in IaaS.

As shown in Table 4.7, cloud players are divided into three classes: (1) cloud service providers and IT administrators, (2) software developers or vendors, and (3) end users or business users. These cloud players vary in their roles under the IaaS, PaaS, and SaaS models. The table entries distinguish the three cloud models

as viewed by different players. From the software vendors' perspective, application performance on a given cloud platform is most important. From the providers' perspective, cloud infrastructure performance is the primary concern. From the end users' perspective, the quality of services, including security, is the most important.

4.5.1.1 Cloud Service Tasks and Trends

Cloud services are introduced in five layers. The top layer is for SaaS applications, as further subdivided into the five application areas in Figure 4.23, mostly for business applications. SaaS tools also apply to distributed collaboration, and financial and human resources management. These cloud services have been growing rapidly in recent years. Collocation services require multiple cloud providers to work together to support supply chains in manufacturing.

4.5.1.2 Software Stack for Cloud Computing

Despite the various types of nodes in the cloud computing cluster, the overall software stacks are built from scratch to meet rigorous goals. Developers have to consider how to design the system to meet critical requirements such as high throughput, HA, and fault tolerance. Even the operating system might be modified to meet the special requirement of cloud data processing. Based on the observations of some typical cloud computing instances, such as Google, Microsoft, and Yahoo!, the overall software stack structure of cloud computing software can be viewed as layers. Each layer has its own purpose and provides the interface for the upper layers just as the traditional software stack does. However, the lower layers are not completely transparent to the upper layers.

4.5.1.3 Runtime Support Services

As in a cluster environment, there are also some runtime supporting services in the cloud computing environment. Cluster monitoring is used to collect the runtime status of the entire cluster. The scheduler queues the tasks submitted to the whole cluster and assigns the tasks to the processing nodes according to node availability. The distributed scheduler for the cloud application has special characteristics that can support cloud applications, such as scheduling the programs written in MapReduce style. The runtime support system keeps the cloud cluster working properly with high efficiency.

Runtime support is software needed in browser-initiated applications applied by thousands of cloud customers. The SaaS model provides the software applications as a service, rather than letting users purchase the software. As a result, on the customer side, there is no upfront investment in servers or software licensing. On the provider side, costs are rather low, compared with conventional hosting of user applications. The customer data is stored in the cloud that is either vendor proprietary or a publicly hosted cloud supporting PaaS and IaaS.

4.5.2 Resource Provisioning and Platform Deployment

The emergence of computing clouds suggests fundamental changes in software and hardware architecture. Cloud architecture puts more emphasis on the number of processor cores or VM instances. Parallelism is exploited at the cluster node level.

4.5.2.1 Provisioning of Compute Resources (VMs)

Providers supply cloud services by signing SLAs with end users. The SLAs must commit sufficient resources such as CPU, memory, and bandwidth that the user can use for a preset period. Under provisioning of resources will lead to broken SLAs and penalties. Over provisioning of resources will lead to resource underutilization, and consequently, a decrease in revenue for the provider. Deploying an autonomous system to efficiently provision resources to users is a challenging problem. The difficulty comes from the unpredictability of consumer demand, software and hardware failures, heterogeneity of services, power management, and conflicts in signed SLAs between consumers and service providers.

Efficient VM provisioning depends on the cloud architecture and management of cloud infra-structures. Resource provisioning schemes also demand fast discovery of services and data in cloud computing infrastructures. In a virtualized cluster of servers, this demands efficient installation of VMs, live VM migration, and fast recovery from failures. To deploy VMs, users treat them as physical hosts with customized operating systems for specific applications.

The provider should offer resource-economic services. Power-efficient schemes for caching, query processing, and thermal management are mandatory due to increasing energy waste by heat dissipation from data centers. Public or private clouds promise to streamline the on-demand provisioning of software, hardware, and data as a service, achieving economies of scale in IT deployment and operation.

4.5.2.2 Resource Provisioning Methods

Figure 4.24 shows three cases of static cloud resource provisioning policies. In case (a), overprovisioning with the peak load causes heavy resource waste (shaded area). In case (b), underprovisioning (along the capacity line) of resources results in losses by both user and provider in that paid demand by the users (the shaded area above the capacity) is not served and wasted resources still exist for those demanded areas below the provisioned capacity. In case (c), the constant provisioning of resources with fixed capacity to a declining user demand could result in even worse resource waste. The user may give up the service by canceling the demand, resulting in reduced revenue for the provider. Both the user and provider may be losers in resource provisioning without elasticity. The various resource-provisioning methods are given below.

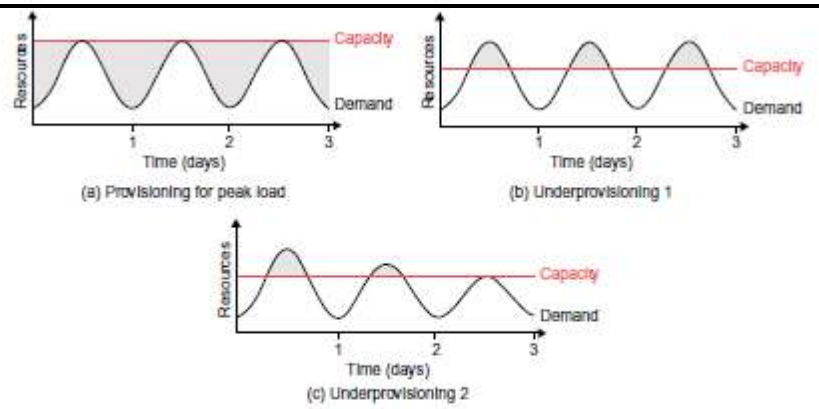

FIGURE 4.24

Three cases of cloud resource provisioning without elasticity: (a) heavy waste due to overprovisioning, (b) underprovisioning and (c) under- and then overprovisioning.

4.5.2.3 Demand-Driven Resource Provisioning

This method adds or removes computing instances based on the current utilization level of the allocated resources. The demand-driven method automatically allocates two Xeon processors for the user application, when the user was using one Xeon processor more than 60 percent of the time for an extended period. In general, when a resource has surpassed a threshold for a certain amount of time, the scheme increases that resource based on demand. When a resource is below a threshold for a certain amount of time, that resource could be decreased accordingly. Amazon implements such an auto-scale feature in its EC2 platform. This method is easy to implement. The scheme does not work out right if the workload changes abruptly.

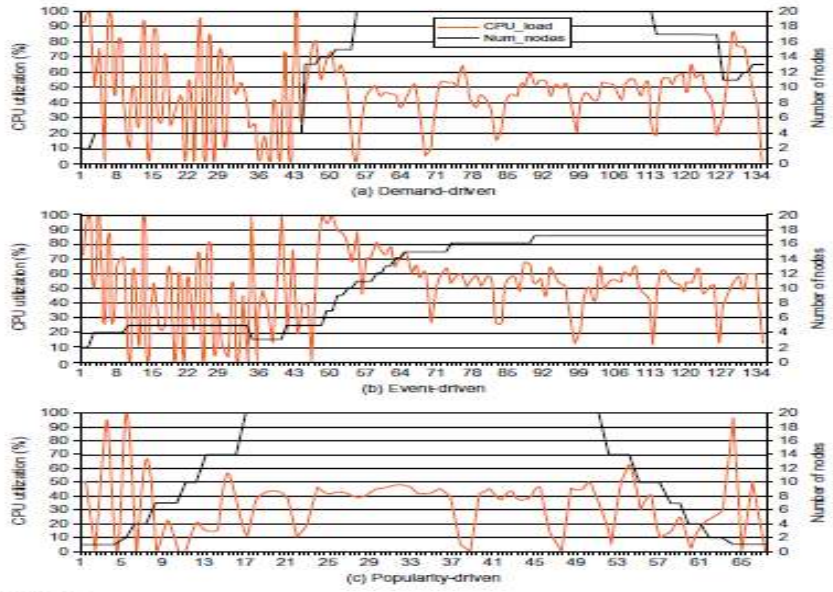

FIGURE 4.25

EC2 performance results on the AWS EC2 platform, collected from experiments at the University of Southern California using three resource provisioning methods.

The x-axis in Figure 4.25 is the time scale in milliseconds. In the beginning, heavy fluctuations of CPU load are encountered. All three methods have demanded a few VM instances initially. Gradually, the utilization rate becomes more stabilized with a maximum of 20 VMs (100 percent utilization) provided for demand-driven provisioning in Figure 4.25(a). However, the event-driven method reaches a stable peak of 17 VMs toward the end of the event and drops quickly in Figure 4.25(b). The popularity provisioning shown in Figure 4.25(c) leads to a similar fluctuation with peak VM utilization in the middle of the plot.

4.5.2.4 Event-Driven Resource Provisioning

This scheme adds or removes machine instances based on a specific time event. The scheme works better for seasonal or predicted events such as Christmastime in the West and the Lunar New Year in the East. During these events, the number of users grows before the event period and then decreases during the event period. This scheme anticipates peak traffic before it happens. The method results in a minimal loss of QoS, if the event is predicted correctly. Otherwise, wasted resources are even greater due to events that do not follow a fixed pattern.

4.5.2.5 Popularity-Driven Resource Provisioning

In this method, the Internet searches for popularity of certain applications and creates the instances by popularity demand. The scheme anticipates increased traffic with popularity. Again, the scheme has a minimal loss of QoS, if the predicted popularity is correct. Resources may be wasted if traffic does not occur as expected. In Figure 4.25(c), EC2 performance by CPU utilization rate (the dark curve with the percentage scale shown on the left) is plotted against the number of VMs provisioned (the light curves with scale shown on the right, with a maximum of 20 VMs provisioned).

4.5.3 Virtual Machine Creation and Management

In the cloud infrastructure management, we will consider the resource management of independent service jobs. Then we will consider how to execute third-party cloud applications. Figure 4.27 shows the interactions among VM managers for cloud creation and management. The managers provide a public API for users to submit and control the VMs.

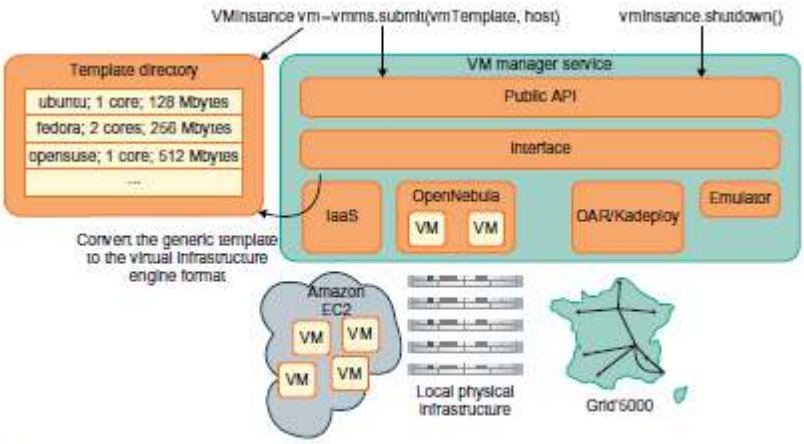

FIGURE 4.27
Interactions among VM managers for cloud creation and management; the manager provides a public API for users to submit and control the VMs.

4.5.3.1 Independent Service Management

Independent services request facilities to execute many unrelated tasks. Commonly, the APIs provided are some web services that the developer can use conveniently. In Amazon cloud computing infrastructure, SQS is constructed for providing a reliable communication service between different providers. Even the endpoint

does not run while another entity has posted a message in SQS. By using independent service providers, the cloud applications can run different services at the same time. Some other services are used for providing data other than the compute or storage services.

4.5.3.2 Running Third-Party Applications

Cloud platforms have to provide support for building applications that are constructed by third-party application providers or programmers. As current web applications are often provided by using Web 2.0 forms (interactive applications with Ajax), the programming interfaces are different from the traditional programming interfaces such as functions in runtime libraries. The APIs are often in the form of services. Web service application engines are often used by programmers for building applications. The web browsers are the user interface for end users.

4.5.3.3 Virtual Machine Manager

The VM manager is the link between the gateway and resources. The gateway doesn't share physical resources directly, but relies on virtualization technology for abstracting them. Hence, the actual resources it uses are VMs. The manager manages VMs deployed on a set of physical resources. The VM manager implementation is generic so that it can connect with different VIEs. Typically, VIEs can create and stop VMs on a physical cluster.

4.5.3.4 Virtual Machine Templates

A VM template is analogous to a computer's configuration and contains a description for a VM with the following static information:

- The number of cores or processors to be assigned to the VM
- The amount of memory the VM requires
- The kernel used to boot the VM's operating system
- The disk image containing the VM's file system
- The price per hour of using a VM

The gateway administrator provides the VM template information when the infrastructure is set up. The administrator can update, add, and delete templates at any time. In addition, each gateway in the Inter Grid network must agree on the templates to provide the same configuration on each site. To deploy an instance of a given VM, the VMM generates a descriptor from the template. This descriptor contains the same fields as the template and additional information related to a specific VM instance. Typically the additional information includes:

- The disk image that contains the VM's file system
- The address of the physical machine hosting the VM

- The VM's network configuration
- The required information for deployment on an IaaS provider

Before starting an instance, the scheduler gives the network configuration and the host's address; it then allocates MAC and IP addresses for that instance. The template specifies the disk image field. To deploy several instances of the same VM template in parallel, each instance uses a temporary copy of the disk image. Hence, the descriptor contains the path to the copied disk image. The descriptor's fields are different for deploying a VM on an IaaS provider. Network information is not needed, because Amazon EC2 automatically assigns a public IP to the instances. The IGG works with a repository of VM templates, called the VM template directory.

4.5.3.5 Distributed VM Management

Figure 4.30 illustrates the interactions between Inter Grid's components. A distributed VM manager makes requests for VMs and queries their status. This manager requests VMs from the gateway on behalf of the user application. The manager obtains the list of requested VMs from the gateway. This list contains a tuple of public IP/private IP addresses for each VM with Secure Shell (SSH) tunnels. Users must specify which VM template they want to use and the number of VM instances needed, the deadline, the wall time, and the address for an alternative gateway.

The local gateway tries to obtain resources from the underlying VIEs. When this is impossible, the local gateway starts a negotiation with any remote gateways to fulfill the request. When a gateway schedules the VMs, it sends the VM access information to the requester gateway. Finally, the manager configures the VM, sets up SSH tunnels, and executes the tasks on the VM. Under the peering policy, each gateway's scheduler uses conservative backfilling to schedule requests. When the scheduler can't start a request immediately using local resources, a redirection algorithm will be initiated.

4.5.4 Global Exchange of Cloud Resources

In order to support a large number of application service consumers from around the world, cloud infrastructure providers (i.e., IaaS providers) have established data centers in multiple geographical locations to provide redundancy and ensure reliability in case of site failures. This approach has many shortcomings. First, it is difficult for cloud customers to determine in advance the best location for hosting their services as they may not know the origin of consumers of their services. Second, SaaS providers may not be able to meet the QoS expectations of their service consumers originating from multiple geographical locations. This necessitates building mechanisms for seamless federation of data centers of a cloud provider or providers supporting dynamic scaling of applications across multiple domains in order to meet QoS targets of cloud customers. Figure 4.30 shows the high-level components of the Melbourne group's proposed InterCloud architecture. In addition, no single cloud infrastructure provider will be able to establish its data centers at all possible locations throughout the world. As a result, cloud application service (SaaS) providers will have difficulty in meeting QoS expectations for all their consumers. Hence, they would like to make use of services of multiple cloud infrastructure service providers who can provide better support for their specific consumer needs. This kind of requirement often arises in enterprises with global

operations and applications such as Internet services, media hosting, and Web 2.0 applications. This necessitates federation of cloud infrastructure service providers for seamless provisioning of services across different cloud providers.

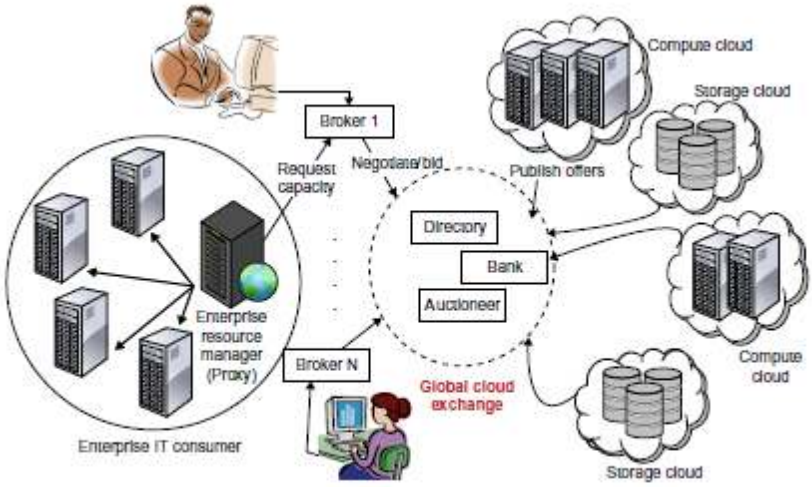

FIGURE 4.30
Inter-cloud exchange of cloud resources through brokering.

4.6 CLOUD SECURITY AND TRUST MANAGEMENT

Lacking trust between service providers and cloud users has hindered the universal acceptance of cloud computing as a service on demand. For web and cloud services, trust and security become even more demanding, because leaving user applications completely to the cloud providers has faced strong resistance by most PC and server users. Cloud platforms become worrisome to some users for lack of privacy protection, security assurance, and copyright protection. As a virtual environment, the cloud poses new security threats that are more difficult to contain than traditional client and server configurations. In many cases, one can extend the trust models for P2P networks and grid systems to protect clouds and data centers.

4.6.1 Cloud Security Defense Strategies

A healthy cloud ecosystem is desired to free users from abuses, violence, cheating, hacking, viruses, rumors, pornography, spam, and privacy and copyright violations. The security demands of three cloud service models, IaaS, PaaS, and SaaS, are described and these security models are based on various SLAs between providers and users.

4.6.1.1 Basic Cloud Security

Three basic cloud security enforcements are expected. First, facility security in data centers demands on-site security year round. Biometric readers, CCTV (close-circuit TV), motion detection, and man traps are often deployed. Also, network security demands fault-tolerant external firewalls, intrusion detection systems (IDSes), and third-party vulnerability assessment. Finally, platform security demands SSL and data decryption, strict password policies, and system trust certification. Figure 4.31 shows the mapping of cloud models, where special security measures are deployed at various cloud operating levels.

Servers in the cloud can be physical machines or VMs. User interfaces are applied to request services. The provisioning tool carves out the systems from the cloud to satisfy the requested service. A security-aware cloud architecture demands security enforcement. Malware-based attacks such as network worms, viruses, and DDoS attacks exploit system vulnerabilities. These attacks compromise system functionality or provide intruders unauthorized access to critical information. Thus, security defenses are needed to protect all cluster servers and data centers.

• Protection of servers from malicious software attacks such as worms, viruses, and malware

• Protection of hypervisors or VM monitors from software-based attacks and vulnerabilities

• Protection of VMs and monitors from service disruption and DoS attacks

• Protection of data and information from theft, corruption, and natural disasters

• Providing authenticated and authorized access to critical data and services

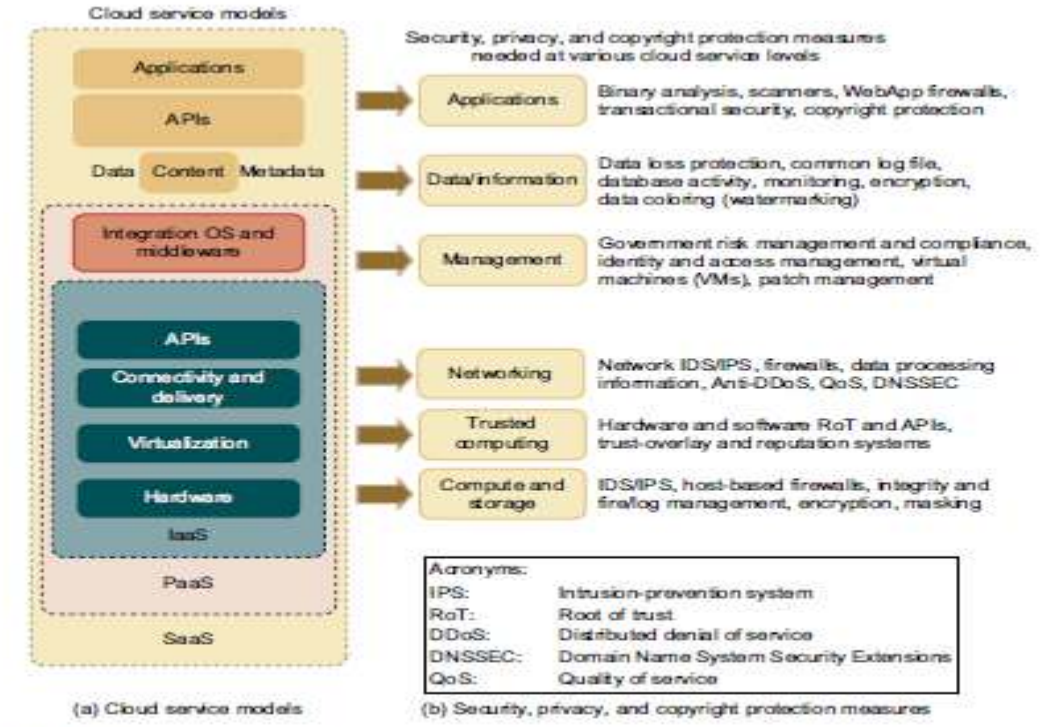

FIGURE 4.31

Cloud service models on the left (a) and corresponding security measures on the right (b); the IaaS is at the innermost level, PaaS is at the middle level, and SaaS is at the outermost level, including all hardware, software, datasets, and networking resources.

4.6.1.2 Security Challenges in VMs

The traditional network attacks include buffer overflows, DoS attacks, spyware, malware, rootkits, Trojan horses, and worms. In a cloud environment, newer attacks may result from hypervisor malware, guest hopping and hijacking, or VM rootkits. Another type of attack is the man-in-the-middle attack for

VM migrations. In general, passive attacks steal sensitive data or passwords. Active attacks may manipulate kernel data structures which will cause major damage to cloud servers. An IDS can be a NIDS or a HIDS. Program shepherding can be applied to control and verify code execution. Other defense technologies include using the RIO dynamic optimization infrastructure, or VMware's vSafe and vShield tools, security compliance for hypervisors, and Intel vPro technology. Others apply a hardened OS environment or use isolated execution and sandboxing.

4.6.1.3 Cloud Defense Methods

Virtualization enhances cloud security. But VMs add an additional layer of software that could become a single point of failure. With virtualization, a single physical machine can be divided or partitioned into multiple VMs (e.g., server consolidation). This provides each VM with better security isolation and each partition is protected from DoS attacks by other partitions. Security attacks in one VM are isolated and contained from affecting the other VMs. VM failures do not propagate to other VMs. The hypervisor provides visibility of the guest OS, with complete guest isolation. Fault containment and failure isolation of VMs provide a more secure and robust environment. Malicious intrusions may destroy valuable hosts, networks, and storage resources. Internet anomalies found in routers, gateways, and distributed hosts may stop cloud services. Trust negotiation is often done at the SLA level. Public Key Infrastructure (PKI) services could be augmented with data-center reputation systems. Worm and DDoS attacks must be contained. It is harder to establish security in the cloud because all data and software are shared by default.

4.6.1.4 Defense with Virtualization

The VM is decoupled from the physical hardware. The entire VM can be represented as a software component and can be regarded as binary or digital data. The VM can be saved, cloned, encrypted, moved, or restored with ease. VMs enable HA and faster disaster recovery. Multiple IDS VMs can be deployed at various resource sites including data centers. Security policy conflicts must be resolved at design time and updated periodically.

4.6.1.5 Privacy and Copyright Protection

The user gets a predictable configuration before actual system integration. With shared files and data sets, privacy, security, and copyright data could be compromised in a cloud computing environment. Users desire to work in a software environment that provides many useful tools to build cloud applications over large data sets. Google's platform essentially applies in-house software to protect resources. It is necessary to protect browser-initiated application software in the cloud environment. Here are several security features desired in a secure cloud:

- Dynamic web services with full support from secure web technologies
- Established trust between users and providers through SLAs and reputation systems
- Effective user identity management and data-access management
- Single sign-on and single sign-off to reduce security enforcement overhead

- Auditing and copyright compliance through proactive enforcement
- Shifting of control of data operations from the client environment to cloud providers
- Protection of sensitive and regulated information in a shared environment

4.6.2 Distributed Intrusion/Anomaly Detection

Data security is the weakest link in all cloud models. We need new cloud security standards to apply common API tools to cope with the data lock-in problem and network attacks or abuses. The IaaS model represented by Amazon is most sensitive to external attacks. Role-based interface tools alleviate the complexity of the provisioning system. For example, IBM's Blue Cloud provisions through a role-based web portal. A SaaS bureau may order secretarial services from a common cloud platform. Many IT companies are now offering cloud services with no guaranteed security. Security threats may be aimed at VMs, guest OSes, and software running on top of the cloud. IDSes attempt to stop these attacks before they take effect. Both signature matching and anomaly detection can be implemented on VMs dedicated to building IDSes. Signature-matching IDS technology is more mature, but requires frequent updates of the signature databases. Network anomaly detection reveals abnormal traffic patterns, such as unauthorized episodes of TCP connection sequences, against normal traffic patterns. Distributed IDSes are needed to combat both types of intrusions.

4.6.2.1 Distributed Defense against DDoS Flooding Attacks

A DDoS defense system must be designed to cover multiple network domains spanned by a given cloud platform. These network domains cover the edge networks where cloud resources are connected. DDoS attacks come with widespread worms. The flooding traffic is large enough to crash the victim server by buffer overflow, disk exhaustion, or connection saturation. Figure 4.33(a) shows a flooding attack pattern. Here, the hidden attacker launched the attack from many zombies toward a victim server at the bottom router R_0.

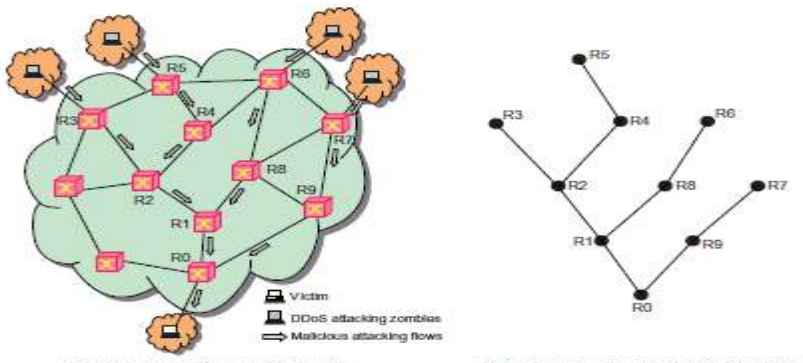

FIGURE 4.33
DDoS attacks and defense by change-point detection at all routers on the flooding tree.

The flooding traffic flows essentially with a tree pattern shown in Figure 4.33(b). Successive attack-transit routers along the tree reveal the abnormal surge in traffic. This DDoS defense system is based on change-point detection by all routers. Based on the anomaly pattern detected in covered network domains, the scheme detects

a DDoS attack before the victim is overwhelmed. The detection scheme is suitable for protecting cloud core networks. The provider-level cooperation eliminates the need for intervention by edge networks.

4.6.3 Data and Software Protection Techniques

4.6.3.1 Data Integrity and Privacy Protection

Users desire a software environment that provides many useful tools to build cloud applications over large data sets. In addition to application software for MapReduce, BigTable, EC2, 3S, Hadoop, AWS, GAE, and WebSphere2, users need some security and privacy protection software for using the cloud. Such software should offer the following features:

• Special APIs for authenticating users and sending e-mail using commercial accounts

• Fine-grained access control to protect data integrity and deter intruders or hackers

• Shared data sets protected from malicious alteration, deletion, or copyright violation

Ability to secure the ISP or cloud service provider from invading users' privacy

• Personal firewalls at user ends to keep shared data sets from Java, JavaScript, and ActiveX applets

• A privacy policy consistent with the cloud service provider's policy, to protect against identity theft, spyware, and web bugs

• VPN channels between resource sites to secure transmission of critical data objects

4.6.3.2 Data Coloring and Cloud Watermarking

With shared files and data sets, privacy, security, and copyright information could be compromised in a cloud computing environment. Users desire to work in a trusted software environment that provides useful tools to build cloud applications over protected data sets. In the past, watermarking was mainly used for digital copyright management. As shown in Figure 4.35, the system generates special colors for each data object. Data coloring means labeling each data object by a unique color. Differently colored data objects are thus distinguishable.

The user identification is also colored to be matched with the data colors. This color matching process can be applied to implement different trust management events. Cloud storage provides a process for the generation, embedding, and extraction of the watermarks in colored objects. Cryptography and watermarking or coloring can be used jointly in a cloud environment.

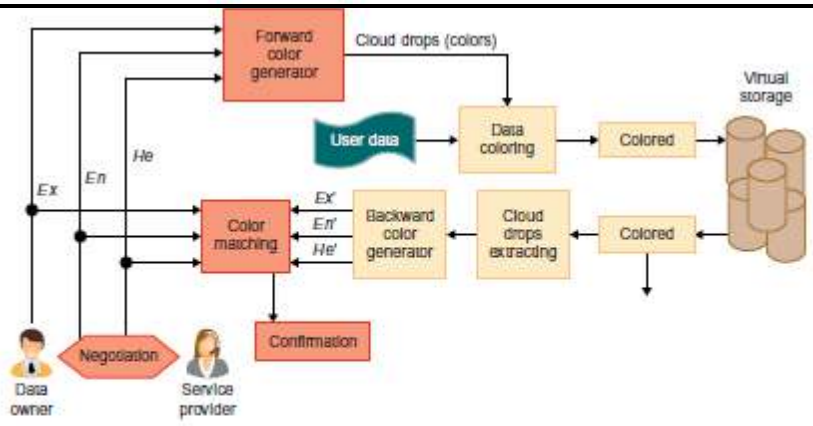

FIGURE 4.35

Data coloring with cloud watermarking for trust management at various security clearance levels in data centers.

4.6.3.3 Data Lock-in Problem and Proactive Solutions

Cloud computing moves both the computation and the data to the server clusters maintained by cloud service providers. Once the data is moved into the cloud, users cannot easily extract their data and programs from cloud servers to run on another platform. This leads to a data lock-in problem. This has hindered the use of cloud computing. Data lock-in is attributed to two causes: lack of interoperability, whereby each cloud vendor has its proprietary API that limits users to extract data once submitted; and lack of application compatibility, in that most computing clouds expect users to write new applications from scratch, when they switch cloud platforms.

One possible solution to data lock-in is the use of standardized cloud APIs. This requires building standard-ized virtual platforms that adhere to OVF, a platform-independent, efficient, extensible, and open format for VMs. This will enable efficient, secure software distribution, facilitating the mobility of VMs. Using OVF one can move data from one application to another. This will enhance QoS, and thus enable cross-cloud applications, allowing workload migration among data centers to user-specific storage. By deploying applications, users can access and intermix applications across different cloud services.

4.6.4 Reputation-Guided Protection of Data Centers

Trust is a personal opinion, which is very subjective and often biased. Trust can be transitive but not necessarily symmetric between two parties. Reputation is a public opinion, which is more objective and often relies on a large opinion aggregation process to evaluate. Reputation may change or decay over time. Recent reputation should be given more preference than past reputation.

4.6.4.1 Reputation System Design Options

Figure 4.36 provides an overview of reputation system design options. Many reputation systems have been proposed in the past mainly for P2P, multiagent, or e-commerce systems. To address reputation systems for cloud services, a systematic approach is based on the design criteria and administration of the reputation systems. Figure 4.36 shows a two-tier classification of existing reputation systems that have been proposed in recent years. Most of them were designed for P2P or social networks. These reputation systems can be converted for protecting cloud computing applications. In general, the reputation systems are classified as centralized or distributed depending on how they are implemented. In a centralized system, a single

central authority is responsible for managing the reputation system, while the distributed model involves multiple control centers working collectively. Reputation-based trust management and techniques for securing P2P and social networks could be merged to defend data centers and cloud platforms against attacks from the open network.

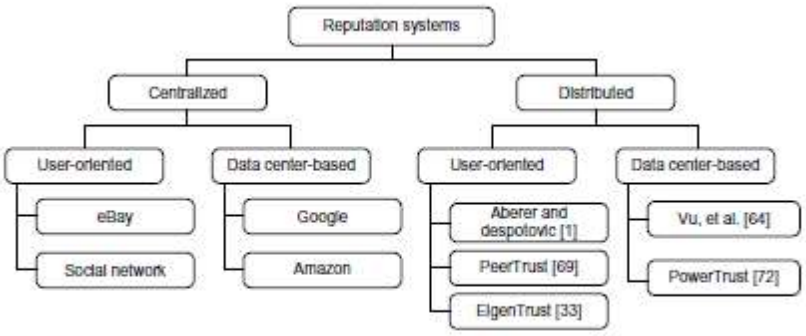

FIGURE 4.36

Design options of reputation systems for social networks and cloud platforms.

A centralized reputation system is easier to implement, but demands more powerful and reliable server resources; a distributed reputation system is much more complex to build. Distributed systems are more scalable and reliable in terms of handling failures. At the second tier, reputation systems are further classif-ied by the scope of reputation evaluation. User-oriented reputation systems focus on individual users or agents. Most P2P reputation systems belong to this category. In data centers, reputation is modeled for the resource site as a whole. This reputation applies to products or services offered by the cloud. Commercial reputation systems have been built by eBay, Google, and Amazon in connection with the services they provide. These are centralized reputation systems.

4.6.4.2 Reputation Systems for Clouds

The reputation system must be designed to benefit both cloud users and data centers. Data objects used in cloud computing reside in multiple data centers over a SAN. In the past, most reputation systems were designed for P2P social networking or for online shopping services. These reputation systems can be converted to protect cloud platform resources or user applications in the cloud. A centralized reputation system is easier to implement, but demands more powerful and reliable server resources. Distributed reputation systems are more scalable and reliable in terms of handling failures. The five security mechanisms presented earlier can be greatly assisted by using a reputation system specifically designed for data centers.

However, it is possible to add social tools such as reputation systems to support safe cloning of VMs. Snapshot control is based on the defined RPO. Users demand new security mechanisms to protect the cloud.

4.6.4.3 Trust Overlay Networks

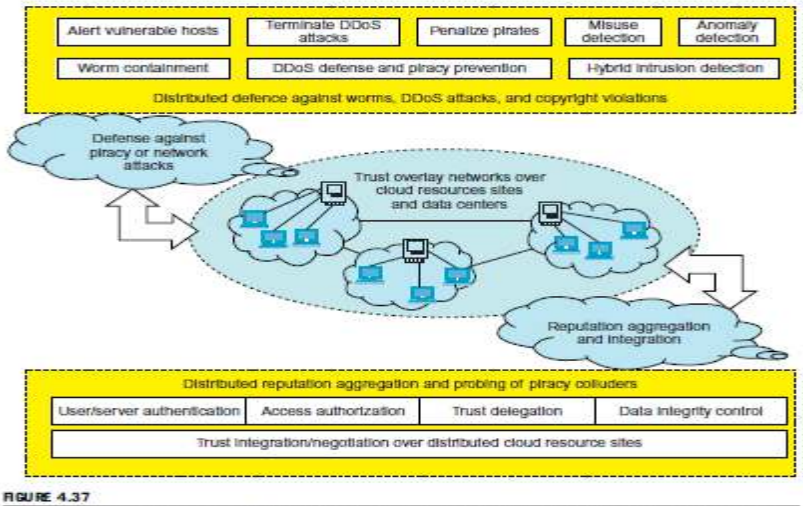

FIGURE 4.37
DHT-based trust overlay networks built over cloud resources provisioned from multiple data centers for trust management and distributed security enforcement.

Figure 4.37 shows construction of the two layers of the trust overlay network. At the bottom layer is the trust overlay for distributed trust negotiation and reputation aggregation over multiple resource sites. This layer handles user/server authentication, access authorization, trust delegation, and data integrity control. At the top layer is an overlay for fast virus/worm signature generation and dissemination and for piracy detection. This overlay facilitates worm containment and IDSes against viruses, worms, and DDoS attacks. The content poisoning technique is reputation-based. This protection scheme can stop copyright violations in a cloud environment over multiple data centers.

5.1 SERVICES AND SERVICE-ORIENTED ARCHITECTURE

In general, SOA is about how to design a software system that makes use of services of new or legacy applications through their published or discoverable interfaces. These applications are often distributed over the networks. SOA also aims to make service interoperability extensible and effective. It prompts architecture styles such as loose coupling, published interfaces, and a standard communication model in order to support this goal. The World Wide Web Consortium (W3C) defines SOA as a form of distributed systems architecture characterized by the following properties :

Logical view: The SOA is an abstracted, logical view of actual programs, databases, business processes, and so on, defined in terms of what it does, typically carrying out a business-level operation. The service is formally defined in terms of the messages exchanged between provider agents and requester agents.

Message orientation: The internal structure of providers and requesters include the implementation language, process structure, and even database structure. These features are deliberately abstracted away in the SOA: Using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed. A key benefit of this concerns legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application to adhere to the formal service definition.

Description orientation: A service is described by machine-executable metadata. The description supports the public nature of the SOA: Only those details that are exposed to the public and are important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description.

• Granularity Services tend to use a small number of operations with relatively large and complex messages.

• Network orientation Services tend to be oriented toward use over a network, though this is not an absolute requirement.

• Platform-neutral Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

SOA is related to early efforts on the architecture style of large-scale distributed systems, particularly Representational State Transfer (REST). Nowadays, REST still provides an alternative to the complex standard-driven web services technology and is used in many Web 2.0 services.

5.1.1 REST and Systems of Systems

REST is a software architecture style for distributed systems, particularly distributed hypermedia systems, such as the World Wide Web. It has recently gained popularity among enterprises such as Google, Amazon, Yahoo!, and especially social networks such as Facebook and Twitter because of its simplicity, and its ease of being published and consumed by clients. REST is shown in Figure 5.1. The REST architectural style is based on four principles: Resource Identification through URIs: The RESTful web service exposes a set of resources which identify targets of interaction with its clients.

FIGURE 5.1
A simple REST interaction between user and server in HTTP specification.

The key abstraction of information in REST is a resource. Any information that can be named can be a resource, such as a document or image or a temporal service. A resource is a conceptual mapping to a set of entities. Each particular resource is identified by a unique name, or more precisely, a Uniform Resource Identifier (URI) which is of type URL, providing a global addressing space for resources involved in an interaction between components as well as facilitating service discovery. The URIs can be bookmarked or exchanged via hyperlinks, providing more readability and the potential for advertisement.

Uniform, Constrained Interface: Interaction with RESTful web services is done via the HTTP standard, client/server cacheable protocol. Resources are manipulated using a fixed set of four CRUD (create, read,

update, delete) verbs or operations: PUT, GET, POST, and DELETE. PUT creates a new resource, which can then be destroyed by using DELETE. GET retrieves the current state of a resource. POST transfers a new state onto a resource.

Self-Descriptive Message: A REST message includes enough information to describe how to process the message. This enables intermediaries to do more with the message without parsing the message contents. In REST, resources are decoupled from their representation so that their content can be accessed in a variety of standard formats (e.g., HTML, XML, MIME, plain text, PDF, JPEG, JSON, etc.). REST provides multiple/alternate representations of each resource. Metadata about the resource is available and can be used for various purposes, such as cache control, transmission error detection, authentication or authorization, and access control.

Stateless Interactions: The REST interactions are "stateless" in the sense that the meaning of a message does not depend on the state of the conversation. Stateless communications improve visibility, since a monitoring system does not have to look beyond a single request data field in order to determine the full nature of the request reliability as it facilitates the task of recovering from partial failures, and increases scalability as discarding state between requests allows the server component to quickly free resources. However, stateless interactions may decrease network performance by increasing the repetitive data (per-interaction overhead). Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction. Such lightweight infrastructure, where services can be built with minimal development tools, is inexpensive and easy to adopt. The effort required to build a client to interact with a RESTful service is rather small as developers can begin testing such services from an ordinary web browser, without having to develop custom client-side software. From an operational point of view, a stateless RESTful web service is scalable to serve a very large number of clients, as a result of REST support for caching, clustering, and load balancing.

RESTful web services can be considered an alternative to SOAP stack or "big web services," described in the next section, because of their simplicity, lightweight nature, and integration with HTTP. With the help of URIs and hyperlinks, REST has shown that it is possible to discover web resources without an approach based on registration to a centralized repository. Recently, the web Application Description Language (WADL) [3] has been proposed as an XML vocabulary to describe RESTful web services, enabling them to be discovered and accessed immediately by potential clients. However, there are not a variety of toolkits for developing RESTful applications. Also, restrictions on GET length, which does not allow encoding of more than 4 KB of data in the resource URI, can create problems because the server would reject such malformed URIs, or may even be subject to crashes. REST is not a standard. It is a design and architectural style for largescale distributed systems.

Table 5.1 REST Architectural Elements (Adapted from [2])

REST Elements	Elements	Example
Data elements	Resource	The intended conceptual target of a hyperted reference
	Resource identifier	URL
	Representation	HTML document, JPEG image, XML, etc.
	Representation metadata	Media type, last-modified time
	Resource metadata	Source link, alternates, vary
	Control data	If-modified-since, cache-control
Connectors	Client	libwww, libwww-perl
	Server	libwww, Apache API, NSAPI
	Cache	Browser cache, Akamai cache network
	Resolver	Bind (DNS lookup library)
	Tunnel	SSL after HTTP CONNECT
Components	Origin server	Apache httpd, Microsoft IIS
	Gateway	Squid, CGI, Reverse Proxy
	Proxy	CERN Proxy, Netscape Proxy, Gauntlet
	User agent	Netscape Navigator, Lynx, MOMspider

Table 5.1 lists the REST architectural elements. Several Java frameworks have emerged to help with building RESTful web services. Restlet [4], a lightweight framework, implements REST architectural elements such as resources, representation, connector, and media type for any kind of RESTful system, including web services. In the Restlet framework, both the client and the server are components. Components communicate with each other via connectors.

5.1.2 Services and Web Services

In an SOA paradigm, software capabilities are delivered and consumed via loosely coupled, reusable, coarse-grained, discoverable, and self-contained services interacting via a message-based communication model. The web has becomes a medium for connecting remote clients with applications for years, and more recently, integrating applications across the Internet has gained in popularity. The term "web service" is often referred to a self-contained, self-describing, modular application designed to be used and accessible by other software applications across the web. Once a web service is deployed, other applications and other web services can discover and invoke the deployed service (Figure 5.2).

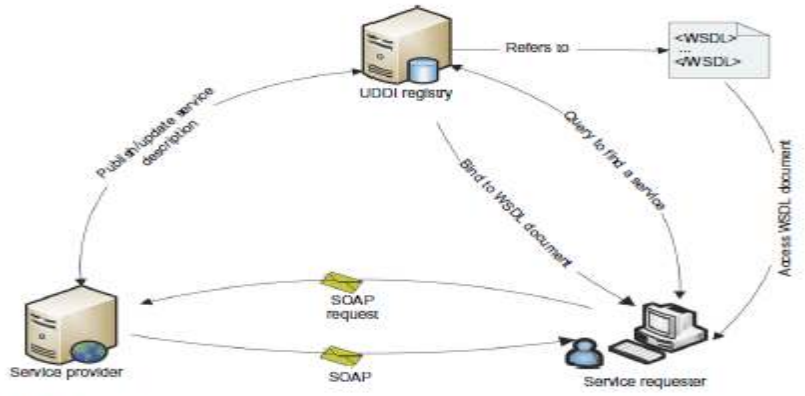

FIGURE 5.2
A simple web service interaction among provider, user, and the UDDI registry.

In fact, a web service is one of the most common instances of an SOA implementation. The W3C working group [1] defines a web service as a software system designed to support interoperable machine-to-machine interaction over a network. According to this definition, a web service has an interface described in a machine-executable format (specifically Web Services Description Language or WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP

with an XML serialization in conjunction with other web-related standards. The technologies that make up the core of today's web services are as follows:

Simple Object Access Protocol (SOAP) SOAP provides a standard packaging structure for transmission of XML documents over various Internet protocols, such as SMTP, HTTP, and FTP. By having such a standard message format, heterogeneous middleware systems can achieve interoperability. A SOAP message consists of a root element called envelope, which contains a header: a container that can be extended by intermediaries with additional application-level elements such as routing information, authentication, transaction management, message parsing instructions, and Quality of Service (QoS) configurations, as well as a body element that carries the payload of the message. The content of the payload will be marshaled by the sender's SOAP engine and unmarshaled at the receiver side, based on the XML schema that describes the structure of the SOAP message.

Web Services Description Language (WSDL) WSDL describes the interface, a set of operations supported by a web service in a standard format. It standardizes the representation of input and output parameters of its operations as well as the service's protocol binding, the way in which the messages will be transferred on the wire. Using WSDL enables disparate clients to automatically understand how to interact with a web service.

Universal Description, Discovery, and Integration (UDDI) UDDI provides a global registry for advertising and discovery of web services, by searching for names, identifiers, categories, or the specification implemented by the web service. UDDI is explained in detail in Section 5.4.

SOAP is an extension, and an evolved version of XML-RPC, a simple and effective remote procedure call protocol which uses XML for encoding its calls and HTTP as a transport mechanism, introduced in 1999 [7]. According to its conventions, a procedure executed on the server and the value it returns was a formatted in XML. However, XML-RPC was not fully aligned with the latest XML standardization. Moreover, it did not allow developers to extend the request or response format of an XML-RPC call. As the XML schema became a W3C recommendation in 2001, SOAP mainly describes the protocols between interacting parties and leaves the data format of exchanging messages to XML schema to handle. The major difference between web service technology and other technologies such as J2EE, CORBA, and CGI scripting is its standardization, since it is based on standardized XML, providing a language-neutral representation of data. Most web services transmit messages over HTTP, making them available as Internet-scale applications. In addition, unlike CORBA and J2EE, using HTTP as the tunneling protocol by web services enables remote communication through firewalls and proxies. SOAP-based web services are also referred to as "big web services" [7]. As we saw earlier in this chapter, RESTful [8] services can also be considered a web service, in an HTTP context. SOAP-based web services interaction can be either synchronous or asynchronous, making them suitable for both request-response and one-way exchange patterns, thus increasing web service availability in case of failure.

5.1.2.1 WS-I Protocol Stack

FIGURE 5.3
WS-I protocol stack and its related specifications.

Unlike RESTful web services that do not cover QoS and contractual properties, several optional specifications have been proposed for SOAP-based web services to define nonfunctional requirements and to guarantee a certain level of quality in message communication as well as reliable, transactional policies, such as WS-Security [9], WS-Agreement [10], WS-ReliableMessaging [11], WS-Transaction [12], and WS-Coordination [13] as shown in Figure 5.3.

As mentioned, SOAP messages are encoded using XML, which requires that all self-described data be sent as ASCII strings. The description takes the form of start and end tags which often constitute half or more of the message's bytes. Transmitting data using XML leads to a considerable transmission overhead, increasing the amount of transferred data by a factor 4 to 10 [14]. Moreover, XML processing is compute and memory intensive and grows with both the total size of the data and the number of data fields, making web services inappropriate for use by limited-profile devices, such as handheld PDAs and mobile phones.

Web services provide on-the-fly software composition, described further in Section 5.5, through the use of loosely coupled, reusable software components. By using Business Process Execution Language for Web Services (BPEL4WS), a standard executable language for specifying interactions between web services recommended by OASIS, web services can be composed together to make more complex web services and workflows. BPEL4WS is an XML-based language, built on top of web service specifications, which is used to define and manage long-lived service orchestrations or processes.

In BPEL, a business process is a large-grained stateful service, which executes steps to complete a business goal. That goal can be the completion of a business transaction, or fulfillment of the job of a service. The steps in the BPEL process execute activities (represented by BPEL language elements) to accomplish work. Those activities are centered on invoking partner services to perform tasks (their job) and return results back to the process. BPEL enables organizations to automate their business processes by orchestrating services. Workflow in a grid context is defined [15] as "The automation of the processes, which involves the orchestration of a set of Grid services, agents and actors that must be combined together to solve a problem or to define a new service."

The JBPM [16] Project, built for the JBoss [17] open source middleware platform, is an example of a workflow management and business process execution system. Another workflow system, Taverna [18], has been extensively used in life science applications. There are a variety of tools for

developing and deploying web services in different languages, among them SOAP engines such as Apache Axis for Java, gSOAP [19] for C++, the Zolera Soap Infrastructure (ZSI) [20] for Python, and Axis2/Java and Axis2/C. These toolkits, consisting of a SOAP engine and WSDL tools for generating client stubs, considerably hide the complexity of web service application development and integration. As there is no standard SOAP mapping for any of the aforementioned languages, two different implementations of SOAP may produce different encodings for the same objects.

Since SOAP can combine the strengths of XML and HTTP, as a standard transmission protocol for data, it is an attractive technology for heterogeneous distributed computing environments, such as grids and clouds, to ensure interoperability. As we discussed in Section 7.4, Open Grid Services Architecture (OGSA) grid services are extensions of web services and in new grid middleware, such as Globus Toolkit 4 and its latest released version GT5, pure standard web services. Amazon S3 as a cloud-based persistent storage service is accessible through both a SOAP and a REST interface. However, REST is the preferred mechanism for communicating with S3 due to the difficulties of processing large binary objects in the SOAP API, and in particular, the limitation that SOAP puts on the object size to be managed and processed. Table 5.3 depicts a sample SOAP request-response to get a user object from S3.

A SOAP message consists of an envelope used by the applications to enclose information that need to be sent. An envelope contains a header and a body block. The EncodingStyle element refers

to the URI address of an XML schema for encoding elements of the message. Each element of a SOAP message may have a different encoding, but unless specified, the encoding of the whole message is as defined in the XML schema of the root element. The header is an optional part of a SOAP message that may contain auxiliary information as mentioned earlier, which does not exist in this example.

The body of a SOAP request-response message contains the main information of the conversation, formatted in one or more XML blocks. In this example, the client is calling CreateBucket of the Amazon S3 web service interface. In case of an error in service invocation, a SOAP message including a Fault element in the body will be forwarded to the service client as a response, as an indicator of a protocol-level error.

5.1.2.2 WS-* Core SOAP Header Standards

Table 5.4 summarizes some of the many (around 100) core SOAP header specifications. There are many categories and several overlapping standards in each category. Many are expanded in this chapter with XML, WSDL, SOAP, BPEL, WS-Security, UDDI, WSRF, and WSRP. The number and complexity of the WS-* standards have contributed to the growing trend of using REST and not web services. It was a brilliant idea to achieve interoperability through self-describing messages, but experience showed that it was too hard to build the required tooling with the required performance and short implementation time.

Table 5.4 The 10 Areas Covered by the Core WS-* Specifications

WS-* Specification Area	Examples
1. Core Service Model	XML, WSDL, SOAP
2. Service Internet	WS-Addressing, WS-MessageDelivery, Reliable WSRM, Efficient MOTM
3. Notification	WS-Notification, WS-Eventing (Publish-Subscribe)
4. Workflow and Transactions	BPEL, WS-Choreography, WS-Coordination
5. Security	WS-Security, WS-Trust, WS-Federation, SAML, WS-SecureConversation
6. Service Discovery	UDDI, WS-Discovery
7. System Metadata and State	WSRF, WS-MetadataExchange, WS-Context
8. Management	WSDM, WS-Management, WS-Transfer
9. Policy and Agreements	WS-Policy, WS-Agreement
10. Portals and User Interfaces	WSRP (Remote Portlets)

5.1.3 Enterprise Multitier Architecture

Enterprise applications often use multitier architecture to encapsulate and integrate various functionalities. Multitier architecture is a kind of client/server architecture in which the presentation, the application processing, and the data management are logically separate processes. The simplest known multilayer architecture is a two-tier or client/server system. This traditional two-tier, client/server model requires clustering and disaster recovery to ensure resiliency. While the use of fewer nodes in an enterprise simplifies manageability, change management is difficult as it requires servers to be taken offline for repair, upgrading, and new application deployments. Moreover, the deployment of new applications and enhancements is complex and time-consuming in fat-client environments, resulting in reduced availability. A three-tier information system consists of the following layers (Figure 5.4):

- Presentation layer Presents information to external entities and allows them to interact with the system by submitting operations and getting responses.

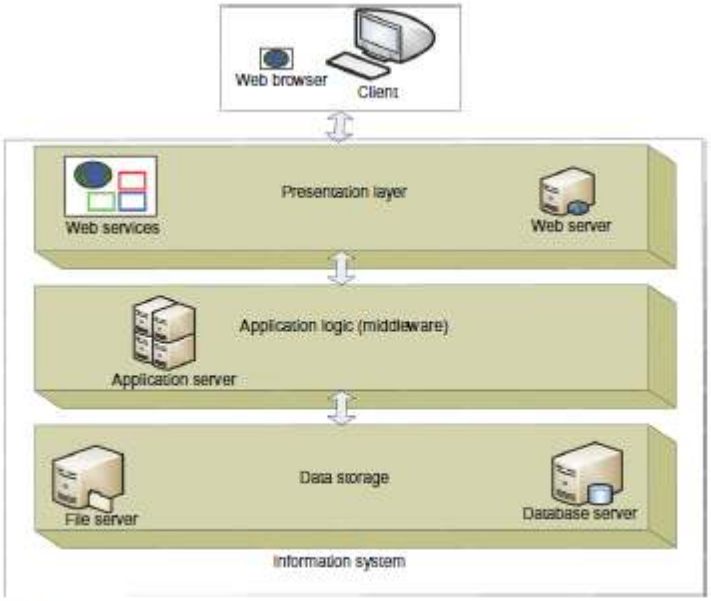

FIGURE 5.4
Three-tier system architecture.

- Business/application logic layer or middleware Programs that implement the actual operations requested by the client through the presentation layer. The middle tier can also control user

authentication and access to resources, as well as performing some of the query processing for the client, thus removing some of the load from the database servers.

- Resource management layer Also known as the data layer, deals with and implements the different data sources of an information system.

In fact, a three-tier system is an extension of two-tier architecture where the application logic is separated from the resource management layer [21]. By the late 1990s, as the Internet became an important part of many applications, the industry extended the three-tier model to an N-tier approach. SOAP-based and RESTful web services have become more integrated into applications. As a consequence, the data tier split into a data storage tier and a data access tier. In very sophisticated systems, an additional wrapper tier can be added to unify data access to both databases and web services. Web services can benefit from the separation of concerns inherent in multitier architecture in almost the same way as most dynamic web applications [22].

The business logic and data can be shared by both automated and GUI clients. The only differences are the nature of the client and the presentation layer of the middle tier. Moreover, separating business logic from data access enables database independence. N-tier architecture is characterized by the functional decomposition of applications, service components, and their distributed deployment. Such an architecture for both web services and dynamic web applications leads to reusability, simplicity, extensibility, and clear separation of component functionalities.

Web services can be seen as another tier on top of the middleware and application integration infrastructure [23], allowing systems to interact with a standard protocol across the Internet. Because each tier can be managed or scaled independently, flexibility is increased in the IT infrastructure that employs N-tier architecture. In the next section, we will describe OGSA, as multitier, service-oriented architecture for middleware which describes the capabilities of a grid computing environment and embodies web services to make computing resources accessible in large-scale heterogeneous environments.

5.1.4 Grid Services and OGSA

The OGSA [24], developed within the OGSA Working Group of the Global Grid Forum (recently renamed to Open Grid Forum or OGF and being merged with the Enterprise Grid Alliance or EGA in June 2006), is a service-oriented architecture that aims to define a common, standard, and open architecture for grid-based applications. “Open” refers to both the process to develop standards and the standards themselves. In OGSA, everything from registries, to computational tasks, to data resources is considered a service. These extensible set of services are the building blocks of an OGSA-based grid. OGSA is intended to:

- Facilitate use and management of resources across distributed, heterogeneous environments
- Deliver seamless QoS
- Define open, published interfaces in order to provide interoperability of diverse resources
- Exploit industry-standard integration technologies

• Develop standards that achieve interoperability • Integrate, virtualize, and manage services and resources in a distributed, heterogeneous environment

- Deliver functionality as loosely coupled, interacting services aligned with industry-accepted web service standards

Based on OGSA, a grid is built from a small number of standards-based components, called grid services. [25] defines a grid service as "a web service that provides a set of well-defined interfaces, following specific conventions (expressed using WSDL)." OGSA gives a high-level architectural view of grid services and doesn't go into much detail when describing grid services. It basically outlines what a grid service should have. A grid service implements one or more interfaces, where each interface defines a set of operations that are invoked by exchanging a defined sequence of messages, based on the Open Grid Services Infrastructure (OGSI) [26]. OGSI, also developed by the Global Grid Forum, gives a formal and technical specification of a grid service.

Grid service interfaces correspond to portTypes in WSDL. The set of portTypes supported by a grid service, along with some additional information relating to versioning, are specified in the grid service's serviceType, a WSDL extensibility element defined by OGSA. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; whereas the conventions address naming and upgradeability. Grid service implementations can target native platform facilities for integration with, and of, existing IT infrastructures.

According to [25], OGSA services fall into seven broad areas, defined in terms of capabilities frequently required in a grid scenario. Figure 5.5 shows the OGSA architecture. These services are summarized as follows:

- Infrastructure Services Refer to a set of common functionalities, such as naming, typically required by higher level services.

- Execution Management Services Concerned with issues such as starting and managing tasks, including placement, provisioning, and life-cycle management. Tasks may range from simple jobs to complex workflows or composite services.

- Data Management Services Provide functionality to move data to where it is needed, maintain replicated copies, run queries and updates, and transform data into new formats. These services must handle issues such as data consistency, persistency, and integrity. An OGSA data service is a web service that implements one or more of the base data interfaces to enable access to, and management of, data resources in a distributed environment. The three base interfaces, Data Access, Data Factory, and Data Management, define basic operations for representing, accessing, creating, and managing data.

- Resource Management Services Provide management capabilities for grid resources: management of the resources themselves, management of the resources as grid components, and management of the OGSA infrastructure. For example, resources can be monitored, reserved,

deployed, and configured as needed to meet application QoS requirements. It also requires an information model (semantics) and data model (representation) of the grid resources and services.

- Security Services Facilitate the enforcement of security-related policies within a (virtual) organization, and supports safe resource sharing. Authentication, authorization, and integrity assurance are essential functionalities provided by these services.

- Information Services Provide efficient production of, and access to, information about the grid and its constituent resources. The term "information" refers to dynamic data or events used for

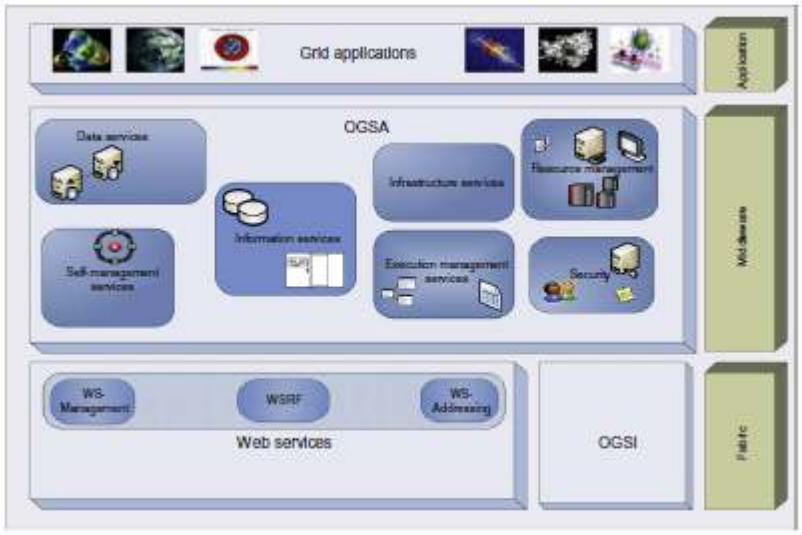

FIGURE 5.5
The OGSA architecture.

status monitoring; relatively static data used for discovery; and any data that is logged. Troubleshooting is just one of the possible uses for information provided by these services.

- Self-Management Services Support service-level attainment for a set of services (or resources), with as much automation as possible, to reduce the costs and complexity of managing the system. These services are essential in addressing the increasing complexity of owning and operating an IT infrastructure.

OGSA has been adopted as reference grid architecture by a number of grid projects. The first prototype grid service implementation was demonstrated January 29, 2002, at a Globus Toolkit tutorial held at Argonne National Laboratory. Since then, the Globus Toolkit 3.0 and 3.2 have offered an OGSA implementation based on OGSI. Two key properties of a grid service are transience and statefulness. Creation and destruction of a transient grid service can be done dynamically. The creation and lifetime of OGSA grid services are handled following the "factory pattern," to be explained in Section 7.3.1. Web service technologies are designed to support loosely coupled, coarse-grained dynamic systems, and hence do not meet all grid requirements, such as keeping state information, and thus they are unable to fully address the wide range of distributed systems OGSA is designed to support.

OGSA applies a set of WSDL extensions to represent the identifiers necessary to implement a grid service instance across any system. These extensions were defined by OGSI. A key extension is the grid service reference: a network-wide pointer to a specific grid service instance, which makes that instance accessible to remote client applications. These extensions, including the Grid Service Handle (GSH) and Grid Service Reference (GSR), will be described in Chapter 7. These extensions include stateful grid services and the shortcomings of OGSI with its dense and long specifications. Further problems concern incompatibility with some current web service tools and the fact that it takes a lot of concepts from object orientation.

Unlike the nature of web services, this has led to close cooperation between the grid and web service communities. As a result of these joint efforts, the Web Services Resource Framework (WSRF) [27], WS-Addressing [28], and WS-Notification (WSN) specifications have been proposed to OASIS. Consequently, OGSI extensions to web services have been deprecated in favor of new web service standards, and in particular, WSRF. WSRF is a collection of five different specifications. Of course, they all relate to the management of WS-Resources. Table 5.5 depicts WSRF-related interface operations.

Plain web services are usually stateless. This means the web service can't "remember" information, or keep state, from one invocation to another. However, since a web service is stateless, the following invocations have no idea of what was done in the previous invocations. Grid applications generally require web services to keep state information as they interact with the clients or other web services. The purpose of WSRF is to define a generic framework for modeling and accessing

Table 5.5 WSRF and Its Related Specifications

Specification			Description
WSRF Specifications	WS-ResourceProperties		Standardizes the definition of the resource properties, its association with the WS interface, and the messages defining the query and update capability against resource properties
	WS-ResourceLifetime		Provides standard mechanisms to manage the life cycle of WS-resources (e.g., setting termination time)
	WS-ServiceGroup		Standard expression of aggregating web services and WS-Resources
	WS-Basefault		Provides a standard way of reporting faults
WSRF-Related Specifications	WS-Notification	WS-Base Notification	Proposes a standard way of expressing the basic roles involved in web service publish and subscribe for notification message exchange
		WS-BrokeredNotification	Standardizes message exchanges involved in web service publish and subscribe of a message broker
		WS-Topics	Defines a mechanism to organize and categorize items of interest for a subscription known as "topics"
	WS-Addressing		Transport-neutral mechanisms to address web service and messages

persistent resources using web services in order to facilitate the definition and implementation of a service and the integration and management of multiple services. Note that "stateless" services can, in fact, remember state if that is carried in messages they receive. These could contain a token remembered in a cookie on the client side and a database or cache accessed by the service. Again, the user accessing a stateless service can establish state for the session through the user login that references permanent information stored in a database.

The state information of a web service is kept in a separate entity called a resource. A service may have more than one (singleton) resource, distinguished by assigning a unique key to each

resource. Resources can be either in memory or persistent, stored in secondary storage such as a file or database. The pairing of a web service with a resource is called a WS-Resource. The preferred way of addressing a specific WS-Resource is to use the qualified endpoint reference (EPR) construct, proposed by the WS-Addressing specification. Resources store actual data items, referred to as resource properties. Resource properties are usually used to keep service data values, providing information on the current state of the service, or metadata about such values, or they may contain information required to manage the state, such as the time when the resource must be destroyed. Currently, the Globus Toolkit 4.0 provides a set of OGSA capabilities based on WSRF.

5.1.5 Other Service-Oriented Architectures and Systems

A survey of services and how they are used can be found in [29]. Here we give two examples: one of a system and one of a small grid.

5.2 MESSAGE-ORIENTED MIDDLEWARE

5.2.1 Enterprise Bus

The services, by definition, interact with messages with a variety of different formats (APIs), wire protocols, and transport mechanisms. It is attractive to abstract the communication mechanism so that services can be defined that communicate independent of details of the implementation. For example, the author of a service should not need to worry that a special port is to be used to avoid firewall difficulties or that we need to use UDP and special fault tolerance approaches to achieve satisfactory latency on a long-distance communication. Further, one may wish to introduce a wrapper so that services expecting messages in different styles (say, SOAP, REST, or Java RMI) can communicate with each other. The term “enterprise service bus” or ESB [30,31] refers to the case where the bus supports the convenient integration of many components, often in different styles. These remarks motivate the messaging black box abstraction shown in Figure 5.6

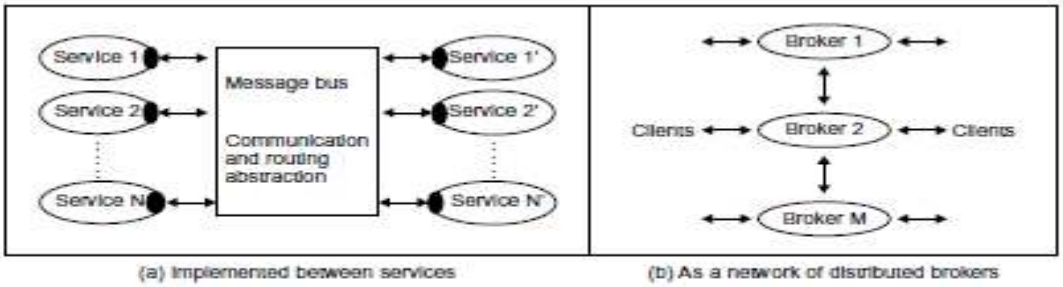

FIGURE 5.6
Two message bus implementations between services or using a broker network.

One does not open a channel between source and destination, but rather injects a message into the bus with enough information to allow it to be delivered correctly. This injection is performed by code loaded into each service and represented by the filled ovals as client interfaces in Figure 5.6(a). The message bus is shown linking services in this figure, but it can work with any software or hardware

entity sending and receiving messages. A simple example could be desktops or smart phones as the clients. Further, such buses can be implemented internally to an application, or in a distributed fashion. In the latter case, the message bus is typically implemented as a set of "brokers" shown in Figure 5.6(b).

The use of multiple brokers allows the bus to scale to many clients (services) and large message traffic. Note that the brokers of Figure 5.6(b) are "just" special servers/services that receive messages and perform needed transformations and routing on them and send out new messages. There is a special (simple) case of message buses where the brokers shown in Figure 5.6(b) are not separate servers but are included in the client software. Note that such buses support not just point-to-point messaging but broadcast or selective multicast to many recipient clients (services).

Often, one implements brokers as managers of queues, and software in this area often has MQ or "Message Queue" in its description. An early important example is MQSeries [32] from IBM which is now marketed as the more recent WebSphereMQ [32,33]. Later, when we study cloud platforms in Chapter 8, we will find that both Azure and Amazon offer basic queuing software. A typical use of a message queue is to relate the master and workers in the "farm" model of parallel computing where the "master" defines separate work items that are placed in a queue which is accessed by multiple workers that select the next available item. This provides a simple dynamically load-balanced parallel execution model. If necessary, the multiple brokers of Figure 5.6(b) can be used to achieve scalability.

5.2.2 Publish-Subscribe Model and Notification

An important concept here is "publish-subscribe" [34] which describes a particular model for linking source and destination for a message bus. Here the producer of the message (publisher) labels the message in some fashion; often this is done by associating one or more topic names from a (controlled) vocabulary. Then the receivers of the message (subscriber) will specify the topics for which they wish to receive associated messages. Alternatively, one can use content-based delivery systems where the content is queried in some format such as SQL.

The use of topic or content-based message selection is termed message filtering. Note that in each of these cases, we find a many-to-many relationship between publishers and subscribers. Publish-subscribe messaging middleware allows straightforward implementation of notification or event-based programming models. The messages could, for example, be labeled by the desired notifying topic (e.g., an error or completion code) and contain content elaborating the notification [34].

5.2.3 Queuing and Messaging Systems

There are several useful standards in this field. The best known is the Java Message Service (JMS) [35] which specifies a set of interfaces outlining the communication semantics in pub/sub and queuing systems. Advanced Message Queuing Protocol (AMQP) [36] specifies the set of wire formats for communications; unlike APIs, wire formats are cross-platform. In the web service arena, there are competing standards, WS-Eventing and WS-Notification, but neither has developed a strong following. Table 5.8 compares a few common messaging and queuing systems. We selected two cloud systems: Amazon Simple Queue and Azure Queue.

We also list MuleMQ [37], which is the messaging framework underlying the ESB [30,31] system Mule, developed in Java, of which there are 2,500 product deployments as of 2010. The focus of Mule is to simplify the integration of existing systems developed using JMS, Web Services, SOAP, JDBC, and traditional HTTP. Protocols supported within Mule include POP, IMAP, FTP, RMI, SOAP, SSL, and SMTP. ActiveMQ [38] is a popular Apache open source message broker while WebSphereMQ [33] is IBM's enterprise message bus offering. Finally, we list the open source NaradaBrokering [39] that is notable for its broad range of supported transports and was successfully used to support a software Multipoint Control Unit (MCU) for multipoint video conferencing and other collaboration capabilities.

Note that the four noncloud systems support JMS. Also, some key features of messaging systems are listed in the table but are not discussed in this brief section. These are security approach and guarantees and mechanisms for message delivery. Time-decoupled delivery refers to situations where the producer and consumer do not have to be present at the same time to exchange messages. Fault tolerance is also an important property: Some messaging systems can back up messages and provide definitive guarantees. This table is only illustrative and there are many other important messaging systems. For example, RabbitMQ [40] is a new impressive system based on the AMQP standard.

5.2.4 Cloud or Grid Middleware Applications

Three examples are given here to illustrate the use of the NaradaBrokering middleware service with distributed computing. The first example is related to environmental protection. The second is for Internet conferencing and the third is for earthquake science applications.

Table 5.8 Comparison of Selected Messaging and Queuing Systems

System Features	Amazon Simple Queue [41]	Azure Queue [42]	ActiveMQ	MuleMQ	WebSphere MQ	Narada Brokering
AMQP compliant	No	No	No, uses OpenWire and Stomp	No	No	No
JMS compliant	No	No	Yes	Yes	Yes	Yes
Distributed broker	No	No	Yes	Yes	Yes	Yes
Delivery guarantees	Message retained in queue for four days	Message accessible for seven days	Based on journaling and JDBC drivers to databases	Disk store uses one file/channel, TTL purges messages	Exactly-once delivery supported	Guaranteed and exactly-once
Ordering guarantees	Best effort, once delivery, duplicate messages exist	No ordering, message returns more than once	Publisher order guarantee	Not clear	Publisher order guarantee	Publisher- or time-order by Network Time Protocol
Access model	SOAP, HTTP-based GET/POST	HTTP REST interfaces	Using JMS classes	JMS, Adm. API, and JNDI	Message Queue Interface, JMS	JMS, WS-Eventing
Max. message	8 KB	8 KB	N/A	N/A	N/A	N/A
Buffering	N/A	Yes	Yes	Yes	Yes	Yes
Time decoupled delivery	Up to four days; supports timeouts	Up to seven days	Yes	Yes	Yes	Yes
Security scheme	Based on HMAC-SHA1 signature, Support for WS-Security 1.0	Access to queues by HMAC SHA256 signature	Authorization based on JAAS for authentication	Access control, authentication, SSL for communication	SSL, end-to-end application-level data security	SSL, end-to-end application-level data security, and ACLs
Support for web services	SOAP-based interactions	REST interfaces	REST	REST	REST, SOAP interactions	WS-Eventing
Transports	HTTP/HTTPS, SSL	HTTP/HTTPS	TCP, UDP, SSL, HTTP/S, Multicast, in-VM, JXTA	Mule ESB supports TCP, UDP, RMI, SSL, SMTP, and FTP	TCP, UDP, Multicast, SSL, HTTP/S	TCP, Parallel TCP, UDP, Multicast, SSL, HTTP/S, IPsec
Subscription formats	Access is to individual queues	Access is to individual queues	JMS spec allows for SQL selectors; also access to individual queues	JMS spec allows for SQL selectors; also access to individual queues	JMS spec allows SQL selectors; access to individual queues	SQL selectors, regular expressions, <tag, value> pairs, XQuery and XPath