CHAPTER

Cloud Infrastructure

3

In this chapter we give an overview of the cloud computing infrastructure at Amazon, Google, and Microsoft as of mid-2012. These cloud service providers support one or more of the three cloud computing delivery models discussed in Section 1.4: *Infrastructure-as-a-Service* (IaaS), *Platform-as-a-Service* (PaaS), and *Software-as-a-Service* (SaaS). Amazon is a pioneer in *IaaS*, Google's efforts are focused on *SaaS* and *PaaS* delivery models, and Microsoft is involved in *PaaS*.

Private clouds are an alternative to public clouds. Open-source cloud computing platforms such as *Eucalyptus* [269], *OpenNebula*, *Nimbus*, and *OpenStack* can be used as a control infrastructure for a private cloud. We continue our discussion of the cloud infrastructure with an overview of service level agreements (SLAs) and the responsibility sharing between users and cloud service providers, followed by a brief discussion of software licensing, energy consumption, and ecological impact of cloud computing. We conclude with a section covering user experiences with current systems.

Several other IT companies are also involved in cloud computing. IBM offers a cloud computing platform, *IBMSmartCloud*, which includes servers, storage, and virtualization components for building private and hybrid cloud computing environments. In October 2012 it was announced that IBM had teamed up with AT&T to give customers access to IBM's cloud infrastructure over AT&T's secure private lines.

In 2011 HP announced plans to enter the cloud computing club. Oracle announced its entry to enterprise computing in the early 2012. The Oracle Cloud is based on Java, SQL standards, and software systems such as Exadata, Exalogic, WebLogic, and Oracle Database. Oracle plans to offer application and platform services. Some of these services are Fusion HCM (Human Capital Management), Fusion CRM (Customer Relation Management), and Oracle Social Network; the platform services are based on Java and SQL.

3.1 Cloud computing at Amazon

Amazon introduced a computing platform that has changed the face of computing in the last decade. First, it installed a powerful computing infrastructure to sustain its core business, e-commerce, selling a variety of goods ranging from books and CDs to gourmet foods and home appliances. Then Amazon discovered that this infrastructure could be further extended to provide affordable and easy-to-use resources for enterprise computing as well as computing for the masses.

In mid-2000 Amazon introduced *Amazon Web Services* (AWS), based on the *IaaS* delivery model. In this model the cloud service provider offers an infrastructure consisting of compute and storage servers interconnected by high-speed networks that support a set of services to access these resources. An application developer is responsible for installing applications on a platform of his or her choice and managing the resources provided by Amazon.

Cloud Computing. http://dx.doi.org/10.1016/B978-0-12-404627-6.00003-8

It is reported that in 2012, businesses in 200 countries used the *AWS*, demonstrating the international appeal of this computing paradigm. A significant number of large corporations as well as start-ups take advantage of computing services supported by the *AWS* infrastructure. For example, one start-up reports that its monthly computing bills at Amazon are in the range of $100,000, whereas it would spend more than $2,000,000 to compute using its own infrastructure, without benefit of the speed and flexibility offered by *AWS*. The start-up employs 10 engineers rather than the 60 it would need to support its own computing infrastructure ("Active in cloud, Amazon reshapes computing," *New York Times*, August 28, 2012).

Amazon Web Services. Amazon was the first provider of cloud computing; it announced a limited public beta release of its Elastic Computing platform called *EC2* in August 2006. Figure 3.1 shows the palette of *AWS* services accessible via the *Management Console* in late 2011 [13–18].

Elastic Compute Cloud (EC2)[1] is a Web service with a simple interface for launching instances of an application under several operating systems, such as several *Linux* distributions, *Microsoft Windows* Server 2003 and 2008, *OpenSolaris*, *FreeBSD*, and *NetBSD*.

An instance is created either from a predefined *Amazon Machine Image* (AMI) digitally signed and stored in *S3* or from a user-defined image. The image includes the operating system, the run-time environment, the libraries, and the application desired by the user. AMI images create an exact copy of the original image but without configuration-dependent information such as the *hostname* or the MAC address. A user can: (i) Launch an instance from an existing AMI and terminate an instance; (ii) start and stop an instance; (iii) create a new image; (iv) add tags to identify an image; and (v) reboot an instance.

EC2 is based on the *Xen* virtualization strategy discussed in detail in Section 5.8. In *EC2* each virtual machine or instance functions as a virtual private server. An instance specifies the maximum amount of resources available to an application, the interface for that instance, and the cost per hour.

A user can interact with *EC2* using a set of SOAP messages (see Section 4.3) and can list available AMI images, boot an instance from an image, terminate an image, display the running instances of a user, display console output, and so on. The user has root access to each instance in the elastic and secure computing environment of *EC2*. The instances can be placed in multiple locations in different regions and availability zones.

EC2 allows the import of virtual machine images from the user environment to an instance through a facility called *VM import.* It also automatically distributes the incoming application traffic among multiple instances using the *elastic load-balancing* facility. *EC2* associates an *elastic IP address* with an account; this mechanism allows a user to mask the failure of an instance and remap a public IP address to any instance of the account without the need to interact with the software support team.

Simple Storage System (S3) is a storage service designed to store large objects. It supports a minimal set of functions: `write`, `read`, and `delete`.

S3 allows an application to handle an unlimited number of objects ranging in size from one byte to five terabytes. An object is stored in a *bucket* and retrieved via a unique developer-assigned key. A bucket can be stored in a region selected by the user. *S3* maintains the name, modification time, an access control list, and up to four kilobytes of user-defined metadata for each object. The object names

[1] Amazon *EC2* was developed by a team led by C. Pinkham, including C. Brown, Q. Hoole, R. Paterson-Jones, and W. Van Biljon, all from Cape Town, South Africa.

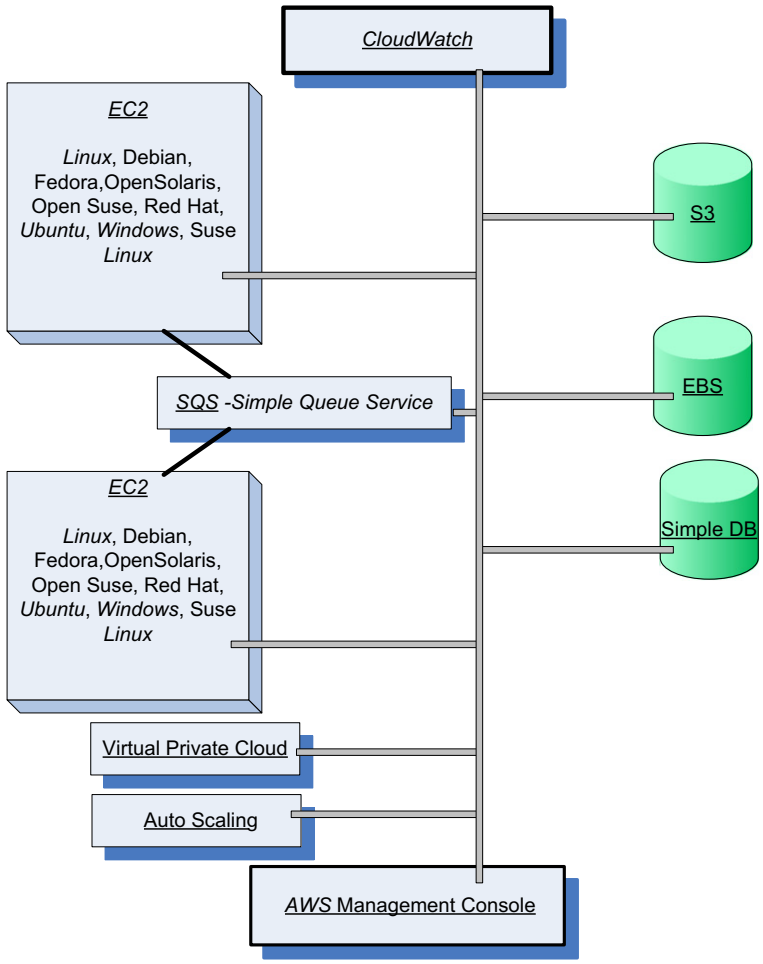

FIGURE 3.1

Services offered by *AWS* are accessible from the *AWS Management Console*. Applications running under a variety of operating systems can be launched using *EC2*. Multiple *EC2* instances can communicate using *SQS*. Several storage services are available: *S3*, *Simple DB*, and *EBS*. The *Cloud Watch* supports performance monitoring; the *Auto Scaling* supports elastic resource management. The *Virtual Private Cloud* allows direct migration of parallel applications.

are global. Authentication mechanisms ensure that data is kept secure; objects can be made public, and rights can be granted to other users.

S3 supports `PUT`, `GET`, and `DELETE` primitives to manipulate objects but does not support primitives to copy, rename, or move an object from one bucket to another. Appending to an object requires a `read` followed by a `write` of the entire object.

S3 computes the *MD5*[2] of every object written and returns it in a field called *ETag*. A user is expected to compute the *MD5* of an object stored or written and compare this with the *ETag*; if the two values do not match, then the object was corrupted during transmission or storage.

The *Amazon S3 SLA* guarantees reliability. *S3* uses standards-based REST and SOAP interfaces (see Section 4.3); the default download protocol is HTTP, but BitTorrent[3] protocol interface is also provided to lower costs for high-scale distribution.

Elastic Block Store (EBS) provides persistent block-level storage volumes for use with Amazon *EC2* instances. A volume appears to an application as a raw, unformatted, and reliable physical disk; the size of the storage volumes ranges from one gigabyte to one terabyte. The volumes are grouped together in availability zones and are automatically replicated in each zone. An *EC2* instance may mount multiple volumes, but a volume cannot be shared among multiple instances. The *EBS* supports the creation of snapshots of the volumes attached to an instance and then uses them to restart an instance. The storage strategy provided by *EBS* is suitable for database applications, file systems, and applications using raw data devices.

Simple DB is a nonrelational data store that allows developers to store and query data items via Web services requests. It supports store-and-query functions traditionally provided only by relational databases. *Simple DB* creates multiple geographically distributed copies of each data item and supports high-performance Web applications; at the same time, it automatically manages infrastructure provisioning, hardware and software maintenance, replication and indexing of data items, and performance tuning.

Simple Queue Service (SQS) is a hosted message queue. *SQS* is a system for supporting automated workflows; it allows multiple Amazon *EC2* instances to coordinate their activities by sending and receiving *SQS* messages. Any computer connected to the Internet can add or `read` messages without any installed software or special firewall configurations.

Applications using *SQS* can run independently and asynchronously and do not need to be developed with the same technologies. A received message is "locked" during processing; if processing fails, the lock expires and the message is available again. The time-out for locking can be changed dynamically via the *ChangeMessageVisibility* operation. Developers can access *SQS* through standards-based *SOAP* and Query interfaces. Queues can be shared with other *AWS* accounts and *anonymously*; queue sharing can also be restricted by IP address and time-of-day. An example showing the use of message queues is presented in Section 4.7.

CloudWatch is a monitoring infrastructure used by application developers, users, and system administrators to collect and track metrics important for optimizing the performance of applications and for increasing the efficiency of resource utilization. Without installing any software, a user can monitor approximately a dozen preselected metrics and then view graphs and statistics for these metrics.

When launching an Amazon Machine Image (AMI), a user can start the *CloudWatch* and specify the type of monitoring. *Basic Monitoring* is free of charge and collects data at five-minute intervals for up

[2]MD5 (Message-Digest Algorithm) is a widely used cryptographic hash function; it produces a 128-bit hash value. It is used for checksums. SHA-i (Secure Hash Algorithm, $0 \leqslant i \leqslant 3$) is a family of cryptographic hash functions; SHA-1 is a 160-bit hash function that resembles MD5.

[3]BitTorrent is a peer-to-peer (P2P) communications protocol for file sharing.

to 10 metrics; *Detailed Monitoring* is subject to a charge and collects data at one-minute intervals. This service can also be used to monitor the latency of access to *EBS* volumes, the available storage space for *RDS* DB instances, the number of messages in *SQS*, and other parameters of interest for applications.

Virtual Private Cloud (VPC) provides a bridge between the existing IT infrastructure of an organization and the *AWS* cloud. The existing infrastructure is connected via a virtual private network (VPN) to a set of isolated *AWS* compute resources. *VPC* allows existing management capabilities such as security services, firewalls, and intrusion detection systems to operate seamlessly within the cloud.

Auto Scaling exploits cloud elasticity and provides automatic scaling of *EC2* instances. The service supports grouping of instances, monitoring of the instances in a group, and defining *triggers* and pairs of *CloudWatch alarms and policies*, which allow the size of the group to be scaled up or down. Typically, a maximum, a minimum, and a regular size for the group are specified.

An *Auto Scaling group* consists of a set of instances described in a static fashion by *launch configurations*. When the group scales up, new instances are started using the parameters for the `runInstances` *EC2* call provided by the launch configuration. When the group scales down, the instances with older launch configurations are terminated first. The monitoring function of the *Auto Scaling* service carries out *health checks* to enforce the specified policies; for example, a user may specify a *health check for elastic load balancing* and then *Auto Scaling* will terminate an instance exhibiting a low performance and start a new one. Triggers use *CloudWatch* alarms to detect events and then initiate specific actions; for example, a trigger could detect when the CPU utilization of the instances in the group goes above 90% and then scale up the group by starting new instances. Typically, triggers to scale up and down are specified for a group.

Several new *AWS* services were introduced in 2012; some of them are in a beta stage at the time of this writing. Among the new services we note: *Route 53*, a low-latency DNS service used to manage user's DNS public records; *Elastic MapReduce (EMR)*, a service supporting processing of large amounts of data using a hosted *Hadoop* running on *EC2* and based on the *MapReduce* paradigm discussed in Section 4.6; *Simple Workflow Service (SWF)*, which supports workflow management (see Section 4.4) and allows scheduling, management of dependencies, and coordination of multiple *EC2* instances; *ElastiCache*, a service enabling Web applications to retrieve data from a managed in-memory caching system rather than a much slower disk-based database; *DynamoDB*, a scalable and low-latency fully managed NoSQL database service; *CloudFront*, a Web service for content delivery; and *Elastic Load Balancer*, a cloud service to automatically distribute the incoming requests across multiple instances of the application. Two new services, the *Elastic Beanstalk* and the *CloudFormation*, are discussed next.

Elastic Beanstalk, a service that interacts with other *AWS* services, including *EC2, S3, SNS, Elastic Load Balance*, and *Auto Scaling*, automatically handles the deployment, capacity provisioning, load balancing, *Auto Scaling*, and application monitoring functions [356]. The service automatically scales the resources as required by the application, either up, or down based on default Auto Scaling settings. Some of the management functions provided by the service are: (i) deployment of a new application version (or rollback to a previous version); (ii) access to the results reported by *CloudWatch* monitoring service; (iii) email notifications when application status changes or application servers are added or removed; and (iv) access to server login files without needing to login to the application servers.

The *Elastic Beanstalk* service is available to developers using a *Java* platform, the *PHP* server-side description language, or *.NET* framework. For example, a Java developer can create the application

using any Integrated Development Environment (IDE) such as *Eclipse* and package the code into a Java Web Application Archive (a file of type ".war") file. The ".war" file should then be uploaded to the *Elastic Beanstalk* using the *Management Console* and then deployed, and in a short time the application will be accessible via a URL.

CloudFormation allows the creation of a *stack* describing the infrastructure for an application. The user creates a template, a text file formatted as in *Javascript Object Notation* (JSON), describing the resources, the configuration values, and the interconnection among these resources. The template can be parameterized to allow customization at run time, (e.g., to specify the types of instances, database port numbers, or RDS size). A template for the creation of an *EC2* instance follows:

```
{
  "Description" : "Create instance running Ubuntu Server 12.04 LTS
                   64 bit AMI
   "Parameters" : {
        "KeyPair" : {
             "Description" : "Key Pair to allow SSH access to the instance",
             "Type" : "String"
        }
  },
  "Resources" : {
        "Ec2Instance" : {
             "Type" : "AWS::EC2::Instance",
             "Properties" : {
                   "KeyName" : {"Ref" : "KeyPair"},
                   "ImageId" : "aki-004ec330"
             }
        }
  },
  "Outputs" : {
        "InstanceId" : {
             "Description" : "The InstanceId of the newly created
             instance",
             "Value" : { "Ref" : "Ec2InstDCM"}
        }
  },
  "AWSTemplateFormatVersion" : "2012-03-09"
}
```

The Amazon Web Services Licensing Agreement (AWSLA) allows the cloud service provider to terminate service to any customer at any time for any reason and contains a covenant not to sue Amazon or its affiliates for any damages that might arise out of the use of *AWS*. As noted in [133], the AWSLA prohibits the use of "other information obtained through *AWS* for the purpose of direct marketing, spamming, contacting sellers or customers." It prohibits *AWS* from being used to store any content

Table 3.1 Amazon data centers are located in several regions; in each region there are multiple availability zones. The billing rates differ from one region to another and can be roughly grouped into four categories: low, medium, high, and very high.

Region	Location	Availability Zones	Cost
US West	Oregon	us-west-2a/2b/2c	Low
US West	North California	us-west-1a/1b/1c	High
US East	North Virginia	us-east-1a/2a/3a/4a	Low
Europe	Ireland	eu-west-1a/1b/1c	Medium
South America	Sao Paulo, Brazil	sa-east-1a/1b	Very high
Asia/Pacific	Tokyo, Japan	ap-northeast-1a/1b	High
Asia/Pacific	Singapore	ap-southeast-1a/1b	Medium

that is "obscene, libelous, defamatory or otherwise malicious or harmful to any person or entity." It also prohibits *S3* from being used "in any way that is otherwise illegal or promotes illegal activities, including without limitation in any manner that might be discriminatory based on race, sex, religion, nationality, disability, sexual orientation, or age."

Users have several choices for interacting with and managing *AWS* resources from either a Web browser or from a system running *Linux* or *Microsoft Windows*:

1. The *AWS* Web Management Console, available at `http://aws.amazon.com/console/`; this is the easiest way to access all services, but not all options may be available in this mode.
2. Command-line tools; see `http://aws.amazon.com/developertools`.
3. AWS SDK libraries and toolkits provided for several programming languages, including Java, PHP,[4] C#, and Obj C.
4. Raw REST requests (see Section 4.3 for a discussion of architectural styles for cloud applications).

Regions and Availability Zones. Today Amazon offers cloud services through a network of data centers on several continents, (see Table 3.1[5]). In each *region* there are several *availability zones* interconnected by high-speed networks; regions communicate through the Internet and do not share resources.

An availability zone is a data center consisting of a large number of servers. A server may run multiple virtual machines or instances, started by one or more users; an instance may use storage services, *S3*, *EBS)*, and *Simple DB*, as well as other services provided by *AWS* (see Figure 3.2). A cloud interconnect allows all systems in an availability zone to communicate with one another and with systems in other availability zones of the same region.

Storage is automatically replicated within a region; *S3* buckets are replicated within an availability zone and between the availability zones of a region, whereas *EBS* volumes are replicated only within the

[4]PHP evolved from a set of Perl scripts designed to produce dynamic Web pages called Personal Home Page Tools into a general-purpose server-side scripting language. The code embedded into an HTML source document is interpreted by a Web server with a PHP processor module, which generates the resulting Web page.

[5]In November 2012 Amazon announced a new region, Asia Pacific-Sydney.

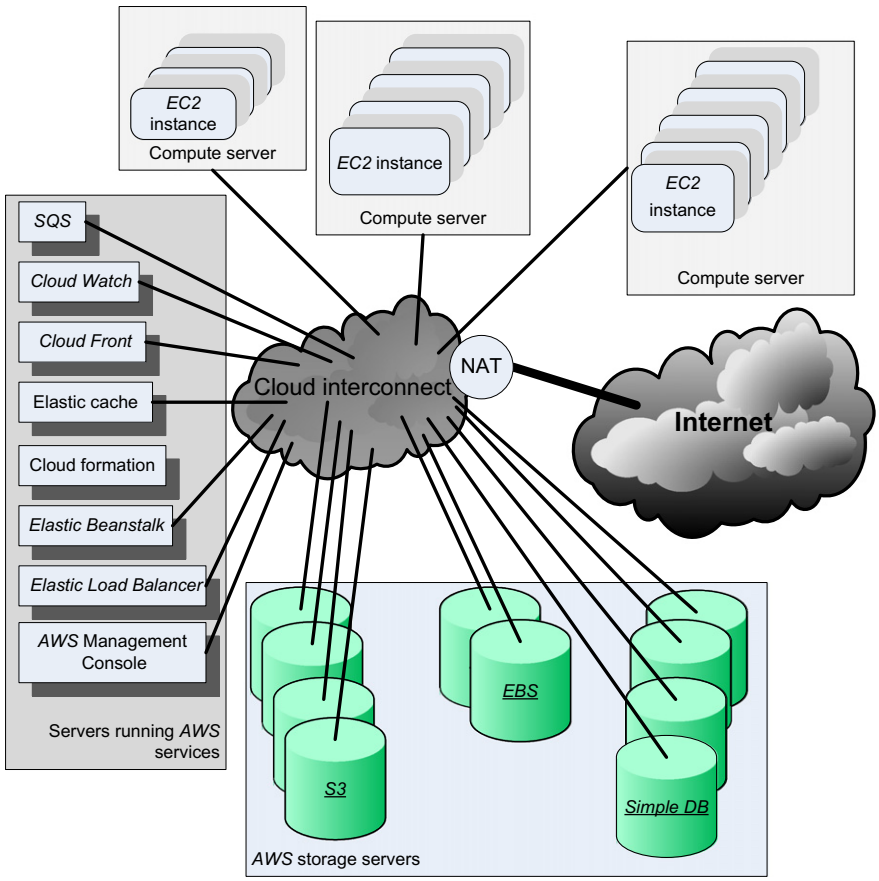

FIGURE 3.2

The configuration of an availability zone supporting *AWS* services. A cloud interconnect supports high-speed communication among compute and storage servers in the zone. It also supports communication with servers in other availablity zones and with cloud users via a Network Address Translation (NAT). NAT maps external IP addresses to internal ones. Multitenancy increases server utilization and lowers costs.

same availability zone. Critical applications are advised to replicate important information in multiple regions to be able to function when the servers in one region are unavailable due to catastrophic events.

A user can request virtual servers and storage located in one of the regions. The user can also request virtual servers in one of the availability zones of that region. The *Elastic Compute Cloud (EC2)* service allows a user to interact and to manage the virtual servers.

The billing rates in each region are determined by the components of the operating costs, including energy, communication, and maintenance costs. Thus, the choice of the region is motivated by the desire to minimize costs, reduce communication latency, and increase reliability and security.

An *instance* is a virtual server. The user chooses the region and the availability zone where this virtual server should be placed and selects from a limited menu of instance types: the one that provides the resources, CPU cycles, main memory, secondary storage, communication, and I/O bandwidth needed by the application.

When launched, an instance is provided with a *DNS name*. This name maps to a *private IP address* for internal communication within the internal *EC2* communication network and a *public IP address* for communication outside the internal Amazon network, (e.g., for communication with the user that launched the instance). Network Address Translation (NAT) maps external IP addresses to internal ones.

The public IP address is assigned for the lifetime of an instance and it is returned to the pool of available public IP addresses when the instance is either stopped or terminated. An instance can request an *elastic IP address*, rather than a public IP address. The elastic IP address is a static public IP address allocated to an instance from the available pool of the availability zone. An elastic IP address is not released when the instance is stopped or terminated and must be released when no longer needed.

The Charges for Amazon Web Services. Amazon charges a fee for *EC2* instances, *EBS* storage, data transfer, and several other services. The charges differ from one region to another and depend on the pricing model; see `http://aws.amazon.com/ec2/pricing` for the current pricing structure.

There are three pricing models for *EC2* instances: on-demand, reserved, and spot. *On-demand instances* use a flat hourly rate, and the user is charged for the time an instance is running; no reservation is required for this most popular model. For *reserved instances* a user pays a one-time fee to lock in a typically lower hourly rate. This model is advantageous when a user anticipates that the application will require a substantial number of CPU cycles and this amount is known in advance. Additional capacity is available at the larger standard rate. In case of *spot instances*, users bid on unused capacity and their instances are launched when the market price reaches a threshold specified by the user.

The *EC2* system offers several instance types:

- *Standard instances.* Micro (StdM), small (StdS), large (StdL), extra large (StdXL); small is the default.
- *High memory instances.* High-memory extra-large (HmXL), high-memory double extra-large (Hm2XL), and high-memory quadruple extra-large (Hm4XL).
- *High CPU instances.* High-CPU extra-large (HcpuXL).
- *Cluster computing.* Cluster computing quadruple extra-large (Cl4XL).

Table 3.2 summarizes the features and the amount of resources supported by each instance. The resources supported by each configuration are main memory, virtual computers (VCs) with a 32- or 64-bit architecture, instance memory (I-memory) on persistent storage, and I/O performance at two levels: moderate (M) or high (H). The computing power of a virtual core is measured in *EC2 compute units (CUs)*.

A main attraction of Amazon cloud computing is the low cost. The dollar amounts charged for one hour of running Amazon's services under *Linux* or *Unix* and *Microsoft Windows* in mid-2012 are summarized in Table 3.3. There are no charges for data transfers from the user's site to the Amazon network or within the Amazon network; the charges for data transfer from the *AWS* network to the outside world depend on the region. For example, the charges for the US West (Oregon) region are shown in Table 3.4.

An Evaluation of Amazon Web Services. In 2007 Garfinkel reported the results of an early evaluation of Amazon Web Services [133]. The paper reports that *EC2* instances are fast, responsive, and

Table 3.2 The nine instances supported by *EC2*. The cluster computing `Cl4XL` (quadruple extra-large) instance uses two Intel Xeon X5570, Quad-Core Nehalem Architecture processors. The instance memory (I-memory) refers to persistent storage; the I/O performance can be moderate (M) or high (H).

Instance Name	API Name	Platform (32/64-bit)	Memory (GB)	Max *EC2* Compute Units	I-Memory (GB)	I/O (M/H)
StdM		32 and 64	0.633	1 VC; 2 CUs		
StdS	`m1.small`	32	1.7	1 VC; 1 CU	160	M
StdL	`m1.large`	64	7.5	2 VCs; 2 × 2 CUs	85	H
StdXL	`m1.xlarge`	64	15	4 VCs; 4 × 2 CUs	1,690	H
HmXL	`m2.xlarge`	64	17.1	2 VCs; 2 × 3.25 CUs	420	M
Hm2XL	`m2.2xlarge`	64	34.2	4 VCs; 4 × 3.25 CUs	850	H
Hm4XL	`m2.4xlarge`	64	68.4	8 VCs; 8 × 3.25 CUs	1,690	H
HcpuXL	`c1.xlarge`	64	7	8 VCs; 8 × 2.5 CUs	1,690	H
Cl4XL	`cc1.4xlarge`	64	18	33.5 CUs	1,690	H

Table 3.3 The charges in dollars for one hour of Amazon's cloud services running under *Linux* or *Unix* and under *Microsoft Windows* for several *EC2* instances.

Instance	*Linux/Unix*	*Windows*
StdM	0.007	0.013
StdS	0.03	0.048
StdL	0.124	0.208
StdXL	0.249	0.381
HmXL	0.175	0.231
Hm2XL	0.4	0.575
Hm4XL	0.799	1.1
HcpuXL	0.246	0.516
Cl4XL	0.544	N/A

Table 3.4 Monthly charges in dollars for data transfer out of the US West (Oregon) region.

Amount of Data	Charge $
First 1 GB	0.00
Up to 10 TB	0.12
Next 40 TB	0.09
Next 100 TB	0.07
Next 350 TB	0.05

very reliable; a new instance could be started in less than two minutes. During the year of testing, one unscheduled reboot and one instance freeze were experienced. No data was lost during the reboot, but no data could be recovered from the virtual disks of the frozen instance.

To test the *S3* service, a bucket was created and loaded with objects in sizes of 1 byte, 1 KB, 1 MB, 16 MB, and 100 MB. The measured throughput for the 1-byte objects reflected the transaction speed of *S3* because the testing program required that each transaction be successfully resolved before the next was initiated. The measurements showed that a user could execute at most 50 non-overlapping *S3* transactions. The 100 MB probes measured the maximum data throughput that the *S3* system could deliver to a single client thread. From the measurements, the author concluded that the data throughput for large objects was considerably larger than for small objects due to a high transaction overhead. The write bandwidth for 1 MB data was roughly 5 MB/s, whereas the `read` bandwidth was five times lower at 1 MB/s.

Another test was designed to see if concurrent requests could improve the throughput of *S3*. The experiment involved two virtual machines running on two different clusters and accessing the same bucket with repeated 100 MB `GET` and `PUT` operations. The virtual machines were coordinated, with each one executing one to six threads for 10 min and then repeating the pattern for 11 h. As the number of threads increased from one to six, the bandwidth received by each thread was roughly cut in half and the aggregate bandwidth of the six threads was 30 MB/s, about three times the aggregate bandwidth of one thread. In 107,556 tests of *EC2*, each one consisting of multiple `read` and `write` probes, only six `write` retries, three `write` errors, and four `read` retries were encountered.

3.2 Cloud computing: the Google perspective

Google's effort is concentrated in the area of *Software-as-a-Service* (SaaS). It is estimated that the number of servers used by Google was close to 1.8 million in January 2012 and was expected to reach close to 2.4 million in early 2013 [289].

Services such as *Gmail, Google Drive, Google Calendar, Picasa*, and *Google Groups* are free of charge for individual users and available for a fee for organizations. These services are running on a cloud and can be invoked from a broad spectrum of devices, including mobile ones such as iPhones, iPads, Black-Berrys, and laptops and tablets. The data for these services is stored in data centers on the cloud.

The *Gmail* service hosts emails on Google servers and, provides a Web interface to access them and tools for migrating from Lotus Notes and Microsoft Exchange. *Google Docs* is Web-based software for building text documents, spreadsheets, and presentations. It supports features such as tables, bullet points, basic fonts, and text size; it allows multiple users to edit and update the same document and view the history of document changes; and it provides a spell checker. The service allows users to import and export files in several formats, including Microsoft Office, PDF, text, and OpenOffice extensions.

Google Calendar is a browser-based scheduler; it supports multiple calendars for a user, the ability to share a calendar with other users, the display of daily/weekly/monthly views, and the ability to search events and synchronize with the Outlook Calendar. Google Calendar is accessible from mobile devices. Event reminders can be received via SMS, desktop popups, or emails. It is also possible to share your calendar with other Google Calendar users. *Picasa* is a tool to upload, share, and edit images; it provides 1 GB of disk space per user free of charge. Users can add tags to images and attach locations to photos

using *Google Maps*. *Google Groups* allows users to host discussion forums to create messages online or via email.

Google is also a leader in the *Platform-as-a-Service* (PaaS) space. AppEngine is a developer platform hosted on the cloud. Initially it supported only Python, but support for Java was added later and detailed documentation for Java is available. The database for code development can be accessed with Google Query Language (GQL) with a SQL-like syntax.

The concept of *structured data* is important to Google's service strategy. The change of search philosophy reflects the transition from unstructured Web content to structured data, data that contains additional information, such as the place where a photograph was taken, information about the singer of a digital recording of a song, the local services at a geographic location, and so on [227].

Search engine crawlers rely on hyperlinks to discover new content. The *deep Web* is content stored in databases and served as pages created dynamically by querying HTML forms. Such content is unavailable to crawlers that are unable to fill out such forms. Examples of *deep Web* sources are sites with geographic-specific information, such as local stores, services, and businesses; sites that report statistics and analysis produced by governmental and nongovernmental organizations; art collections; photo galleries; bus, train, and airline schedules; and so on. Structured content is created by labeling; *Flickr* and *Google Co-op* are examples of structures where labels and annotations are added to objects, images, and pages stored on the Web.

Google Co-op allows users to create customized search engines based on a set of *facets* or categories. For example, the facets for a search engine for the database research community available at `http://data.cs.washington.edu/coop/dbresearch/index.html` are `professor`, `project`, `publication`, `jobs`.

Google Base is a service allowing users to load structured data from different sources to a central repository that is a very large, self-describing, semi-structured, heterogeneous database. It is self-describing because each item follows a simple schema: (item type, attribute names). Few users are aware of this service. *Google Base* is accessed in response to keyword queries posed on *Google.com*, provided that there is relevant data in the database. To fully integrate Google Base, the results should be ranked across properties. In addition, the service needs to propose appropriate refinements with candidate values in select menus; this is done by computing histograms on attributes and their values during query time.

Google Drive is an online service for data storage that has been available since April 2012. It gives users 5 GB of free storage and charges $4 per month for 20 GB. It is available for PCs, MacBooks, iPhones, iPads, and Android devices and allows organizations to purchase up to 16 TB of storage.

Specialized structure-aware search engines for several interest areas, including travel, weather, and local services, have already been implemented. However, the data available on the Web covers a wealth of human knowledge; it is not feasible to define all the possible domains and it is nearly impossible to decide where one domain ends and another begins.

Google has also redefined the laptop with the introduction of the *Chromebook*, a purely Web-centric device running *Chrome OS*. Cloud-based applications, extreme portability, built-in *3G* connectivity, almost instant-on, and all-day battery life are the main attractions of this device with a keyboard.

Google adheres to a bottom-up, engineer-driven, liberal licensing and user application development philosophy, whereas Apple, a recent entry in cloud computing, tightly controls the technology stack,

builds its own hardware, and requires application developers to follow strict rules. Apple products, including the iPhone, the iOS, the iTunes Store, *Mac OS X*, and iCloud, offer unparalleled polish and effortless interoperability, but the flexibility of Google results in more cumbersome user interfaces for the broad spectrum of devices running the Android OS.

Google as well as the other cloud service providers must manage vast amounts of data. In a world where users would most likely desire to use multiple cloud services from independent providers, the question of whether the traditional data base management services (DBMSs) are sufficient to ensure interoperability comes to mind. A DBMS efficiently supports data manipulations and query processing but operates in a single administrative domain and uses well-defined schema. The interoperability of data management services requires *semantic integration* of services based on different schemas. An answer to the limitations of traditional DBMS is the so-called *dataspaces* introduced in [127]; dataspaces do not aim at data integration but rather at data coexistence.

3.3 *Microsoft Windows Azure* and online services

Azure and *Online Services* are, respectively, *PaaS* and *SaaS* cloud platforms from Microsoft. *Windows Azure* is an operating system, *SQL Azure* is a cloud-based version of the SQL Server, and *Azure AppFabric* (formerly .NET Services) is a collection of services for cloud applications.

Windows Azure has three core components (see Figure 3.3): *Compute*, which provides a computation environment; *Storage* for scalable storage; and *Fabric Controller*, which deploys, manages, and monitors applications; it interconnects nodes consisting of servers, high-speed connections, and switches.

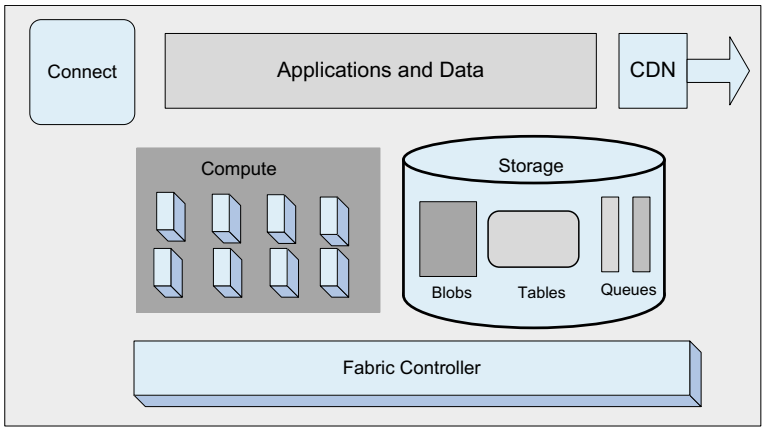

FIGURE 3.3

The components of *Windows Azure*: *Compute*, which runs cloud applications; *Storage*, which uses blobs, tables, and queues to store data; *Fabric Controller*, which deploys, manages, and monitors applications; CDN, which maintains cache copies of data; and *Connect*, which allows IP connections between the user systems and applications running on *Windows Azure*.

The *Content Delivery Network* (CDN) maintains cache copies of data to speed up computations. The *Connect* subsystem supports IP connections between the users and their applications running on *Windows Azure*. The API interface to *Windows Azure* is built on REST, HTTP, and XML. The platform includes five services: *Live Services, SQL Azure, AppFabric, SharePoint*, and *Dynamics CRM*. A client library and tools are also provided for developing cloud applications in Visual Studio.

The computations carried out by an application are implemented as one or more *roles*; an application typically runs multiple *instances of a role*. We can distinguish (i) Web role instances used to create Web applications; (ii) Worker role instances used to run Windows-based code; and (iii) VM role instances that run a user-provided Windows Server 2008 R2 image.

Scaling, load balancing, memory management, and reliability are ensured by a *fabric controller*, a distributed application replicated across a group of machines that owns all of the resources in its environment – computers, switches, load balancers – and it is aware of every *Windows Azure* application. The fabric controller decides where new applications should run; it chooses the physical servers to optimize utilization using configuration information uploaded with each *Windows Azure* application. The configuration information is an XML-based description of how many Web role instances, how many Worker role instances, and what other resources the application needs. The fabric controller uses this configuration file to determine how many VMs to create.

Blobs, tables, queues, and drives are used as scalable storage. A blob contains binary data; a container consists of one or more blobs. Blobs can be up to a terabyte and they may have associated metadata (e.g., the information about where a JPEG photograph was taken). Blobs allow a *Windows Azure* role instance to interact with persistent storage as though it were a local NTFS[6] file system. Queues enable Web role instances to communicate asynchronously with Worker role instances.

The Microsoft Azure platform currently does not provide or support any distributed parallel computing frameworks, such as *MapReduce, Dryad*, or *MPI*, other than the support for implementing basic queue-based job scheduling [148].

3.4 Open-source software platforms for private clouds

Private clouds provide a cost-effective alternative for very large organizations. A private cloud has essentially the same structural components as a commercial one: the servers, the network, virtual machines monitors (VMMs) running on individual systems, an archive containing disk images of virtual machines (VMs), a front end for communication with the user, and a cloud control infrastructure. Open-source cloud computing platforms such as *Eucalyptus* [269], *OpenNebula*, and *Nimbus* can be used as a control infrastructure for a private cloud.

Schematically, a cloud infrastructure carries out the following steps to run an application:

- Retrieves the user input from the front end.
- Retrieves the disk image of a VM from a repository.
- Locates a system and requests the VMM running on that system to set up a VM.

[6]New Technology File System (NTFS) is the standard file system of the *Microsoft Windows* operating system starting with *Windows NT 3.1*, *Windows 2000*, and *Windows XP*.

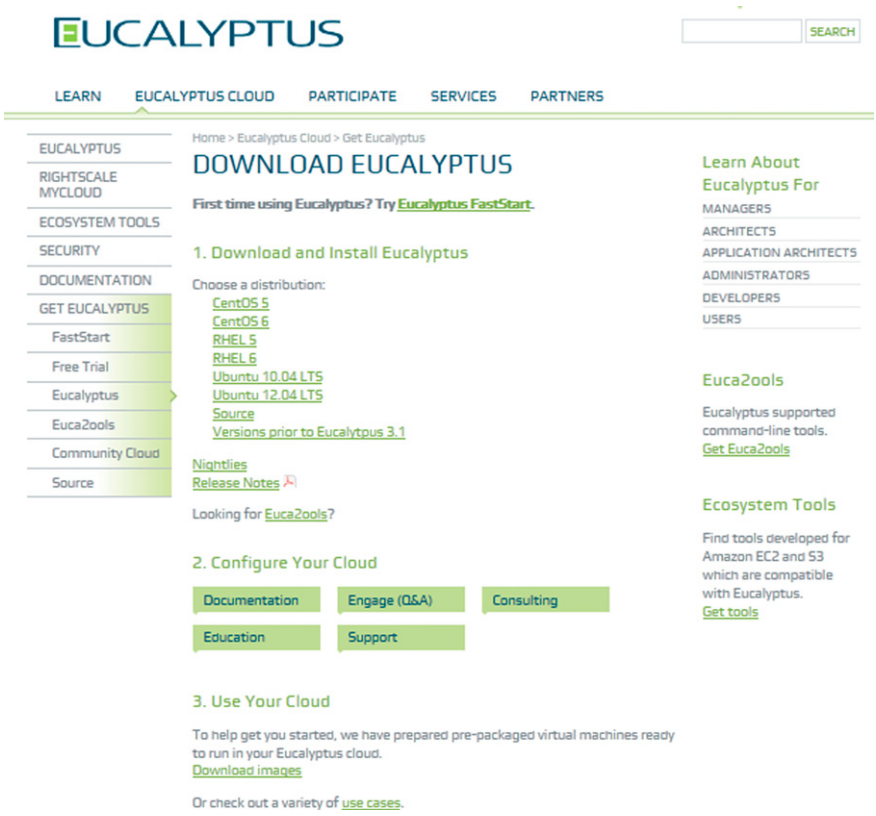

FIGURE 3.4

Eucalyptus supports several distributions and is well-documented software for private clouds.

- Invokes the DHCP[7] and the IP bridging software to set up a MAC and IP address for the VM.

We discuss briefly the three open-source software systems, *Eucalyptus, OpenNebula*, and *Nimbus*.

Eucalyptus (`www.eucalyptus.com`) can be regarded as an open-source counterpart of Amazon's *EC2*, (see Figure 3.4). The systems supports several operating systems including CentOS 5 and 6, RHEL 5 and 6, *Ubuntu* 10.04 LTS, and 12.04 LTS.

The components of the system are:

[7]The Dynamic Host Configuration Protocol (DHCP) is an automatic configuration protocol; it assigns an IP address to a client system. A DHCP server has three methods of allocating IP addresses. (1) Dynamic allocation: A network administrator assigns a range of IP addresses to DHCP, and each client computer on the LAN is configured to request an IP address from the DHCP server during network initialization. The request-and-grant process uses a lease concept with a controllable time period, allowing the DHCP server to reclaim (and then reallocate) IP addresses that are not renewed. (2) Automatic allocation: The DHCP server permanently assigns a free IP address to a client from the range defined by the administrator. (3) Static allocation: The DHCP server allocates an IP address based on a table with MAC address/IP address pairs, which are manually filled in; only clients with a MAC address listed in this table will be allocated an IP address.

- *Virtual machine.* Runs under several VMMs, including *Xen, KVM*, and *Vmware*.
- *Node controller.* Runs on every server or node designated to host a VM and controls the activities of the node. Reports to a cluster controller.
- *Cluster controller.* Controls a number of servers. Interacts with the node controller on each server to schedule requests on that node. Cluster controllers are managed by the cloud controller.
- *Cloud controller.* Provides the cloud access to end users, developers, and administrators. It is accessible through command-line tools compatible with *EC2* and through a Web-based Dashboard. Manages cloud resources, makes high-level scheduling decisions, and interacts with cluster controllers.
- *Storage controller.* Provides persistent virtual hard drives to applications. It is the correspondent of *EBS*. Users can create snapshots from EBS volumes. Snapshots are stored in *Walrus* and made available across availability zones.
- *Storage service (Walrus).* Provides persistent storage and, similarly to *S3*, allows users to store objects in buckets.

The system supports a strong separation between the user space and the administrator space; users access the system via a Web interface, whereas administrators need root access. The system supports a decentralized resource management of multiple clusters with multiple cluster controllers, but a single head node for handling user interfaces. It implements a distributed storage system, the analog of Amazon's *S3* system, called *Walrus*. The procedure to construct a virtual machine is based on the generic one described in [323]:

- The *euca2ools* front end is used to request a VM.
- The VM disk image is transferred to a compute node.
- This disk image is modified for use by the VMM on the compute node.
- The compute node sets up network bridging to provide a virtual network interface controller (NIC)[8] with a virtual Media Access Control (MAC) address.[9]
- In the head node the DHCP is set up with the MAC/IP pair.
- VMM activates the VM.
- The user can now `ssh`[10] directly into the VM.

The system can support a large number of users in a corporate enterprise environment. Users are shielded from the complexity of disk configurations and can choose their VM from a set of five configurations for available processors, memory, and hard drive space set up by the system administrators.

Open-Nebula (`www.opennebula.org`) is a private cloud with users actually logging into the head node to access cloud functions. The system is centralized and its default configuration uses NFS (Network File System). The procedure to construct a virtual machine consists of several steps: (i) the user signs into the head node using `ssh`; (ii) the system uses the `onevm` command to request a VM; (iii) the VM

[8] An NIC is the hardware component connecting a computer to a LAN. It is also known as a network interface card, network adapter, or LAN adapter.

[9] A MAC address is a unique identifier permanently assigned to a network interface by the manufacturer.

[10] Secure Shell (`ssh`) is a network protocol that allows data to be exchanged using a secure channel between two networked devices. `ssh` uses public-key cryptography to authenticate the remote computer and allow the remote computer to authenticate the user. It also allows remote control of a device.

Table 3.5 A side-by-side comparison of *Eucalyptus*, *OpenNebula*, and *Nimbus*.

	Eucalyptus	*OpenNebula*	*Nimbus*
Design	Emulate *EC2*	Customizable	Based on Globus
Cloud type	Private	Private	Public/Private
User population	Large	Small	Large
Applications	All	All	Scientific
Customizability	Administrators and limited users	Administrators and users	All but image storage and credentials
Internal security	Strict	Loose	Strict
User access	User credentials	User credentials	x509 credentials
Network access	To cluster controller	—	To each compute node

template disk image is transformed to fit the correct size and configuration within the NFS directory on the head node; (iv) the `oned` daemon on the head node uses `ssh` to log into a compute node; (v) the compute node sets up network bridging to provide a virtual NIC with a virtual MAC; (vi) the files needed by the VMM are transferred to the compute node via the NFS; (vii) the VMM on the compute node starts the VM; and (viii) the user is able to `ssh` directly to the VM on the compute node.

According to the analysis in [323], the system is best suited for an operation involving a small-to medium-sized group of trusted and knowledgeable users who are able to configure this versatile system based on their needs.

Nimbus (`www.nimbusproject.org`) is a cloud solution for scientific applications based on the Globus software. The system inherits from Globus the image storage, the credentials for user authentication, and the requirement that a running Nimbus process can `ssh` into all compute nodes. Customization in this system can only be done by the system administrators.

Table 3.5 summarizes the features of the three systems [323]. The conclusions of the comparative analysis are as follows: *Eucalyptus* is best suited for a large corporation with its own private cloud because it ensures a degree of protection from user malice and mistakes. *OpenNebula* is best suited for a testing environment with a few servers. *Nimbus* is more adequate for a scientific community less interested in the technical internals of the system than with broad customization requirements.

OpenStack is an open-source project started in 2009 at the National Aeronautics and Space Administration (NASA) in collaboration with Rackspace (`www.rackspace.com`) to develop a scalable cloud operating system for farms of servers using standard hardware. Though recently NASA has moved its cloud infrastructure to *AWS* in addition to Rackspace, several other companies, including HP, Cisco, IBM, and Red Hat, have an interest in *OpenStack*. The current version of the system supports a wide range of features such as application programming interfaces (APIs) with rate limiting and authentication; live VM management to run, reboot, suspend, and terminate instances; role-based access control; and the ability to allocate, track, and limit resource utilization. The administrators and the users control their resources using an extensible Web application called the *Dashboard*.

3.5 Cloud storage diversity and vendor lock-in

There are several risks involved when a large organization relies solely on a single cloud provider. As the short history of cloud computing shows, cloud services may be unavailable for a short or even an extended period of time. Such an interruption of service is likely to negatively impact the organization and possibly diminish or cancel completely the benefits of utility computing for that organization. The potential for permanent data loss in case of a catastrophic system failure poses an equally great danger.

Last but not least, a Cloud Service Provider (CSP) may decide to increase the prices for service and charge more for computing cycles, memory, storage space, and network bandwidth than other CSPs. The alternative in this case is switching to another provider. Unfortunately, this solution could be very costly due to the large volume of data to be transferred from the old to the new provider. Transferring terabytes or possibly petabytes of data over the network takes a fairly long time and incurs substantial charges for the network bandwidth.

This chapter discusses the storage models supported by the cloud infrastructure provided by Amazon, Google, and Microsoft; Chapter 8 covers the architecture of storage systems in general. Reliability is a major concern, and here we discuss a solution that addresses both avoidance of vendor lock-in and storage reliability.

A solution to guarding against the problems posed by the vendor lock-in is to replicate the data to multiple cloud service providers. Straightforward replication is very costly and, at the same time, poses technical challenges. The overhead to maintain data consistency could drastically affect the performance of the virtual storage system consisting of multiple full replicas of the organization's data spread over multiple vendors. Another solution could be based on an extension of the design principle of a RAID-5 system used for reliable data storage.

A RAID-5 system uses block-level stripping with distributed parity over a disk array, as shown in Figure 3.5(a); the disk controller distributes the sequential blocks of data to the physical disks and computes a parity block by bit-wise XOR-ing of the data blocks. The parity block is written on a different disk for each file to avoid the bottleneck possible when all parity blocks are written to a dedicated disk, as is done in the case of RAID-4 systems. This technique allows us to recover the data after a single disk loss. For example, if Disk 2 in Figure 3.5 is lost, we still have all the blocks of the third file, $c1$, $c2$, and $c3$, and we can recover the missing blocks for the others as follows:

$$\begin{aligned} a2 &= (a1) \text{ XOR } (aP) \text{ XOR } (a3) \\ b2 &= (b1) \text{ XOR } (bP) \text{ XOR } (b3) \ . \\ d1 &= (dP) \text{ XOR } (d2) \text{ XOR } (d3) \end{aligned} \tag{39}$$

Obviously, we can also detect and correct errors in a single block using the same procedure. The RAID controller also allows parallel access to data (for example, the blocks $a1$, $a2$, and $a3$ can be `read` and written concurrently) and it can aggregate multiple write operations to improve performance.

The system in Figure 3.5(b) strips the data across four clusters. The access to data is controlled by a proxy that carries out some of the functions of a RAID controller, as well as authentication and other security-related functions. The proxy ensures *before-and-after* atomicity as well as *all-or-nothing* atomicity for data access; the proxy buffers the data, possibly converts the data manipulation commands,

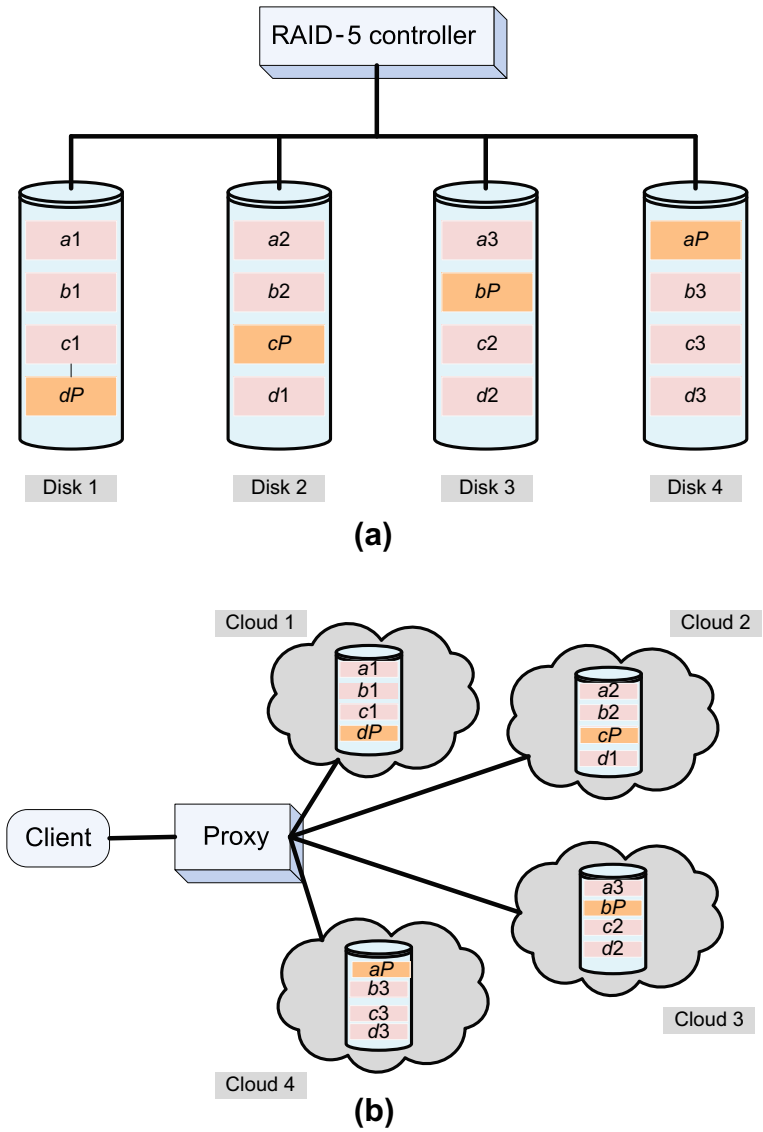

FIGURE 3.5

(a) A (3, 4) RAID-5 configuration in which individual blocks are stripped over three disks and a parity block is added; the parity block is constructed by XOR-ing the data blocks (e.g., $aP = a1 \text{XOR} a2 \text{XOR} a3$). The parity blocks are distributed among the four disks: aP is on disk 4, bP on disk 3, cP on disk 2, and dP on disk 1. (b) A system that strips data across four clouds; the proxy provides transparent access to data.

optimizes the data access (e.g., aggregates multiple write operations), converts data to formats specific to each cloud, and so on.

This elegant idea immediately raises several questions: How does the response time of such a scheme compare with that of a single storage system? How much overhead is introduced by the proxy? How could this scheme avoid a single point of failure, the proxy? Are there standards for data access implemented by all vendors?

An experiment to answer some of these question is reported in [5]; the Redundant Array of Cloud Storage (RACS) system uses the same data model and mimics the interface of the *S3* provided by *AWS*. The *S3* system, discussed in Section 3.1, stores the data in *buckets*, each bucket being a flat namespace with *keys* associated with *objects* of arbitrary size but less than 5 GB. The prototype implementation discussed in [5] led the authors to conclude that the cost increases and the performance penalties of the RACS systems are relatively minor. The paper also suggests an implementation to avoid the single point of failure by using several proxies. Then the system is able to recover from the failure of a single proxy; clients are connected to several proxies and can access the data stored on multiple clouds.

It remains to be seen whether such a solution is feasible in practice for organizations with a very large volume of data, given the limited number of cloud storage providers and the lack of standards for data storage. A basic question is whether it makes sense to trade basic tenets of cloud computing, such as simplicity and homogeneous resources controlled by a single administrative authority, for increased reliability and freedom from vendor lock-in [67].

This brief discussion hints at the need for standardization and for scalable solutions, two of the many challenges faced by cloud computing in the near future. The pervasive nature of scalability dominates all aspects of cloud management and cloud applications; solutions that perform well on small systems are no longer feasible when the number of systems or the volume of the input data of an application increases by one or more orders of magnitude. Experiments with small test-bed systems produce inconclusive results. The only alternative is to conduct intensive simulations to prove (or disprove) the advantages of a particular algorithm for resource management or the feasibility of a particular data-intensive application.

We can also conclude that cloud computing poses challenging problems to service providers and to users. The service providers have to develop strategies for resource management subject to quality of service and cost constraints, as discussed in Chapter 6. At the same time, the cloud application developers have to be aware of the limitations of the cloud computing model.

3.6 Cloud computing interoperability: the Intercloud

Cloud interoperability could alleviate the concern that users could become hopelessly dependent on a single cloud service provider, the so-called vendor lock-in discussed in Section 3.5. It seems natural to ask the question whether an Intercloud – a "cloud of clouds," a federation of clouds that cooperate to provide a better user experience – is technically and economically feasible. The Internet is a network of networks; hence, it appears that an Intercloud seems plausible [47–49].

Closer scrutiny shows that the extension of the concept of interoperability from networks to clouds is far from trivial. A network offers one high-level service, the transport of digital information from a source, a host outside a network, to a destination, another host, or another network that can deliver the information to its final destination. This transport of information through a network of networks is

feasible because before the Internet was born, agreements on basic questions were reached: (a) how to uniquely identify the source and the destination of the information; (b) how to navigate through a maze of networks; and (c) how to actually transport the data between a source and a destination. The three elements on which agreements were reached are, respectively, the IP address, the IP protocol, and transport protocols such as TCP and UDP.

The situation is quite different in cloud computing. First, there are no standards for storage of processing; second, the clouds we have seen so far are based on different delivery models: *SaaS, PaaS,* and *IaaS*. Moreover, the set of services supported by each of these delivery models is not only large, it is open; new services are offered every few months. For example, in October 2012 Amazon announced a new service, the *AWS GovCloud (US)*.

The question of whether cloud service providers (CSPs) are willing to cooperate to build an Intercloud is open. Some CSPs may think that they have a competitive advantage due to the uniqueness of the added value of their services. Thus, exposing how they store and process information may adversely affect their business. Moreover, no CSP will be willing to change its internal operation, so a first question is whether an Intercloud could be built under these conditions.

Following the concepts borrowed from the Internet, a federation of clouds that does not dictate the internal organization or the structure of a cloud but only the means to achieve cloud interoperability is feasible. Nevertheless, building such an infrastructure seems a formidable task. First, we need a set of standards for interoperability covering items such as naming, addressing, identity, trust, presence, messaging, multicast, and time. Indeed, we need common standards for identifying all the objects involved as well as the means to transfer, store, and process information, and we also need a common clock to measure the time between two events.

An Intercloud would then require the development of an *ontology*[11] for cloud computing. Then each cloud service provider would have to create a description of all resources and services using this ontology. Due to the very large number of systems and services, the volume of information provided by individual cloud service providers would be so large that a distributed database not unlike the Domain Name Service (DNS) would have to be created and maintained. According to [47] this vast amount of information would be stored in Intercloud *root* nodes, analogous to the root nodes of the DNS.

Each cloud would then require an interface, a so-called Intercloud *exchange*, to translate the common language describing all objects and actions included in a request originating from another cloud in terms of its internal objects and actions. To be more precise, a request originated in one cloud would have to be translated from the internal representation in that cloud to a common representation based on the shared ontology and then, at the destination, it would be translated into an internal representation that can be acted on by the destination cloud. This raises immediately the question of efficiency and performance. This question cannot be fully answered now, since an Intercloud exists only on paper, but there is little doubt that performance will be greatly affected.

Security is a major concern for cloud users, and an Intercloud could only create new threats. The primary concern is that tasks will cross from one administrative domain to another and that sensitive information about the tasks and users could be disclosed during this migration. A seamless migration of tasks in an Intercloud requires a well-thought-out trust model.

[11] An ontology provides the means for knowledge representation within a domain. It consists of a set of domain concepts and the relationships among the concepts.

The Public Key Infrastructure (PKI),[12] an all-or-nothing trust model, is not adequate for an Intercloud, where the trust must be nuanced. A nuanced model for handling digital certificates means that one cloud acting on behalf of a user may grant access to another cloud to `read` data in storage, but not to start new instances.

The solution advocated in [48] for trust management is based on dynamic *trust indexes* that can change in time. The Intercloud roots play the role of Certificate Authority, whereas the Intercloud exchanges determine the trust indexes between clouds.

Encryption must be used to protect the data in storage and in transit in the Intercloud. The OASIS[13] Key Management Interoperability Protocol (KMIP)[14] is proposed for key management.

In summary, the idea of an Intercloud opens up a wide range of interesting research topics. The practicality of the concepts can only be discussed after the standardization efforts under way at NIST bear fruit.

3.7 Energy use and ecological impact of large-scale data centers

We start our discussion of energy use by data centers and its economic and ecological impact with a brief analysis of the concept of *energy-proportional systems*. This is a very important concept because a strategy for resource management in a computing cloud is to concentrate the load on a subset of servers and switching the rest of the servers to a standby mode whenever possible [7]. This strategy aims to reduce power consumption and, implicitly, the cost of providing computing and storage services; we analyze this subject in depth in Chapter 6.

The operating efficiency of a system is captured by an expression of "performance per Watt of power." It is widely reported that, during the last two decades, the performance of computing systems has increased much faster than their operating efficiency; for example, during the period 1998–2007, the performance of supercomputers increased by 7,000% whereas their operating efficiency increased by only 2,000%.

In an ideal world, the energy consumed by an idle system should be near zero and should grow linearly with the system load. In real life, even machines whose power requirements scale linearly, use more than half the power when idle than they use at full load (see Figure 3.6) [42].

Energy-proportional systems could lead to large savings in energy costs for computing clouds. An *energy-proportional* system consumes no power when idle, very little power under a light load, and gradually more power as the load increases. By definition, an ideal energy-proportional system is always operating at 100% efficiency. Humans are a good approximation of an ideal energy proportional system; human energy consumption is about 70 W at rest and 120 W on average on a daily basis and can go as high as 1,000–2,000 W during a strenuous, short effort [42].

Different subsystems of a computing system behave differently in terms of energy efficiency. Many processors have reasonably good energy-proportional profiles, but significant improvements in memory

[12] PKI is a model to create, distribute, revoke, use, and store digital certificates. It involves several components: (1) The Certificate Authority (CA) binds public keys to user identities in a given domain. (2) The third-party Validation Authority (VA) guarantees the uniqueness of the user identity. (3) The Registration Authority (RA) guarantees that the binding of the public key to an individual cannot be challenged, the so-called *nonrepudiation*.

[13] OASIS stands for Organization for the Advancement of Structured Information Standards.

[14] The KMIP Specification version 1.0 is available at `http://docs.oasis-open.org/kmip/spec/v1.0/kmip-spec-1.0.html`.

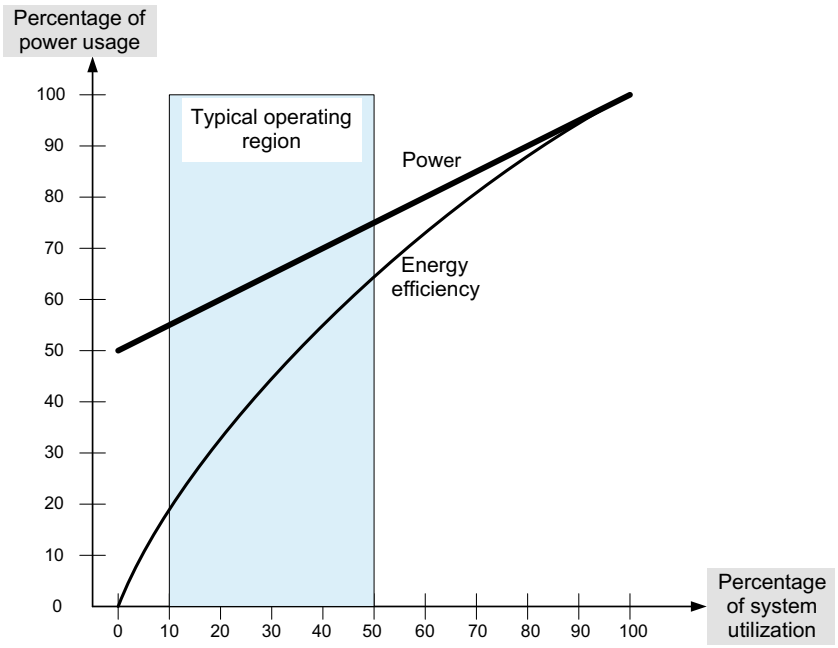

FIGURE 3.6

Even when power requirements scale linearly with the load, the energy efficiency of a computing system is not a linear function of the load; even when idle, a system may use 50% of the power corresponding to the full load. Data collected over a long period of time shows that the typical operating region for the servers at a data center is from about 10% to 50% of the load.

and disk subsystems are necessary. The processors used in servers consume less than one-third of their peak power at very low load and have a dynamic range[15] of more than 70% of peak power; the processors used in mobile and/or embedded applications are better in this respect. According to [42] the dynamic power range of other components of a system is much narrower: less than 50% for dynamic random access memory (DRAM), 25% for disk drives, and 15% for networking switches.

A number of proposals have emerged for *energy-proportional* networks; the energy consumed by such networks is proportional to the communication load. For example, in [6] the authors argue that a data center network based on a flattened butterfly topology is more energy and cost efficient than one using a different type of interconnect.

High-speed channels typically consist of multiple serial lanes with the same data rate; a physical unit is stripped across all the active lanes. Channels commonly operate plesiochronously[16] and are always on, regardless of the load, because they must still send idle packets to maintain byte and lane alignment

[15]The dynamic range in this context is the lower and the upper range of the power consumption of the device. A large dynamic range means that the device is better; it is able to operate at a lower fraction of its peak power when its load is low.

[16]Different parts of the system are almost but not quite perfectly synchronized; in this case, the core logic in the router operates at a frequency different from that of the I/O channels.

across the multiple lanes. An example of an *energy-proportional* network is *InfiniBand*, discussed in Section 3.1.

Energy saving in large-scale storage systems is also of concern. A strategy to reduce energy consumption is to concentrate the workload on a small number of disks and allow the others to operate in a low-power mode. One of the techniques to accomplish this task is based on replication. A replication strategy based on a sliding window is reported in [364]; measurement results indicate that it performs better than LRU, MRU, and LFU[17] policies for a range of file sizes, file availability, and number of client nodes, and the power requirements are reduced by as much as 31%.

Another technique is based on data migration. The system in [158] uses data storage in virtual nodes managed with a distributed hash table; the migration is controlled by two algorithms, a short-term optimization algorithm, used for gathering or spreading virtual nodes according to the daily variation of the workload so that the number of active physical nodes is reduced to a minimum, and a long-term optimization algorithm, used for coping with changes in the popularity of data over a longer period (e.g., a week).

The energy consumption of large-scale data centers and their costs for energy and for cooling are significant now and are expected to increase substantially in the future. In 2006, the 6,000 data centers in the United States reportedly consumed 61×10^9 KWh of energy, 1.5% of all electricity consumption in the country, at a cost of $4.5 billion [364].

The predictions have been dire: The energy consumed by the data centers was expected to double from 2006 to 2011; peak instantaneous demand was expected to increase from 7 GW in 2006 to 12 GW in 2011, requiring the construction of 10 new power plants. The energy consumption of data centers and the network infrastructure is predicted to reach 10, 300 TWh/year[18] in 2030, based on 2010 levels of efficiency [295]. These increases are expected in spite of the extraordinary reduction in energy requirements for computing activities; over the past 30 years the energy efficiency per transistor on a chip has improved by six orders of magnitude.

The effort to reduce energy use is focused on the computing, networking, and storage activities of a data center. A 2010 report shows that a typical Google cluster spends most of its time within the 10–50% CPU utilization range; there is a mismatch between server workload profile and server energy efficiency [6]. A similar behavior is also seen in the data center networks; these networks operate in a very narrow dynamic range, and the power consumed when the network is idle is significant compared to the power consumed when the network is fully utilized.

Many proposals argue that dynamic resource provisioning is necessary to minimize power consumption. Two main issues are critical for energy saving: the amount of resources allocated to each application and the placement of individual workloads. For example, a resource management framework combining a utility-based dynamic virtual machine provisioning manager with a dynamic VM placement manager to minimize power consumption and reduce SLA violations is presented in [358].

The support for network-centric content consumes a very large fraction of the network bandwidth; according to the CISCO VNI forecast, consumer traffic was responsible for around 80% of bandwidth use in 2009 and is expected to grow at a faster rate than business traffic. Data intensity for various activities ranges from 20 MB/minute for HDTV streaming to 10 MB/minute for standard TV streaming,

[17] Least recently used (LRU), most recently used (MRU), and least frequently used (LFU) are replacement policies used by memory hierarchies for caching and paging.

[18] One TWh (Tera Watt hour) is equal to 10^{12} Wh.

1.3 MB/minute for music streaming, 0.96 MB/minute for Internet radio, 0.35 MB/minute for Internet browsing, and 0.0025 MB/minute for ebook reading [295].

The same study reports that if the energy demand for bandwidth is 4 Watts-hour per MB[19] and if the demand for network bandwidth is 3.2 GB/day/person or 2,572 EB/year for the entire world population, then the energy required for this activity will be 1, 175 GW. These estimates do not count very high-bandwidth applications that may emerge in the future, such as 3D TV, personalized immersive entertainment such as Second Life, or massively multiplayer online games.

The power consumption required by different types of human activities is partially responsible for the world's greenhouse gas emissions. According to a recent study [295], the greenhouse gas emissions due to data centers are estimated to increase from 116×10^6 tons of CO_2 in 2007 to 257 tons in 2020, due primarily to increased consumer demand. Environmentally opportunistic computing is a macro-scale computing idea that exploits the physical and temporal mobility of modern computer processes. A prototype called a Green Cloud is described in [376].

3.8 Service- and compliance-level agreements

A *service-level agreement (SLA)* is a negotiated contract between two parties, the customer and the service provider. The agreement can be legally binding or informal and specifies the services that the customer receive rather than how the service provider delivers the services. The objectives of the agreement are:

- Identify and define customers' needs and constraints, including the level of resources, security, timing, and quality of service.
- Provide a framework for understanding. A critical aspect of this framework is a clear definition of classes of service and costs.
- Simplify complex issues; for example, clarify the boundaries between the responsibilities of the clients and those of the provider of service in case of failures.
- Reduce areas of conflict.
- Encourage dialogue in the event of disputes.
- Eliminate unrealistic expectations.

An SLA records a common understanding in several areas: (i) services, (ii) priorities, (iii) responsibilities, (iv) guarantees, and (v) warranties. An agreement usually covers: services to be delivered, performance, tracking and reporting, problem management, legal compliance and resolution of disputes, customer duties and responsibilities, security, handling of confidential information, and termination.

Each area of service in cloud computing should define a "target level of service" or a "minimum level of service" and specify the levels of availability, serviceability, performance, operation, or other attributes of the service, such as billing. Penalties may also be specified in the case of noncompliance with the SLA. It is expected that any service-oriented architecture (SOA) will eventually include middleware supporting SLA management. The *Framework 7* project supported by the European Union is researching this area (see `http://sla-at-soi.eu/`).

[19] In the United States, in 2006, the energy consumed to download data from a data center across the Internet was in the range of 9 to 16 Watts hour per MB.

The common metrics specified by an SLA are service-specific. For example, the metrics used by a *call center* usually are: (i) abandonment rate: percentage of calls abandoned while waiting to be answered; (ii) average speed to answer: average time before the service desk answers a call; (iii) time service factor: percentage of calls answered within a definite time frame; (iv) first-call resolution: percentage of incoming calls that can be resolved without a callback; and (v) turnaround time: time to complete a certain task.

There are two well-differentiated phases in SLA management: the negotiation of the contract and the monitoring of its fulfillment in real time. In turn, automated negotiation has three main components: (i) the *object of negotiation*, which defines the attributes and constraints under negotiation; (ii) the *negotiation protocols*, which describe the interaction between negotiating parties; and (iii) the *decision models* responsible for processing proposals and generating counterproposals.

The concept of compliance in cloud computing is discussed in [55] in the context of the user's ability to select a provider of service. The selection process is subject to customizable compliance with user requirements, such as security, deadlines, and costs. The authors propose an infrastructure called *Compliant Cloud Computing* (C3) consisting of: (i) a language to express user requirements and the compliance level agreements (CLAs) and (ii) the middleware for managing CLAs.

The Web Service Agreement Specification (WS-Agreement) [20] uses an XML-based language to define a protocol for creating an agreement using a predefined template with some customizable aspects. It only supports one-round negotiation without counterproposals. A policy-based framework for automated SLA negotiation for a virtual computing environment is described in [379].

3.9 Responsibility sharing between user and cloud service provider

After reviewing cloud services provided by Amazon, Google, and Microsoft, we are in a better position to understand the differences among *SaaS*, *IaaS*, and *PaaS*. There is no confusion about *SaaS*; the service provider supplies both the hardware and the application software, and the user has direct access to these services through a Web interface and has no control over cloud resources. Typical examples are Google with Gmail, Google Docs, Google Calendar, Google Groups, and Picasa and Microsoft with the Online Services.

In the case of *IaaS*, the service provider supplies the hardware (servers, storage, networks) and system software (operating systems, databases); in addition, the provider ensures system attributes such as security, fault tolerance, and load balancing. The representative of *IaaS* is Amazon *AWS*.

PaaS provides only a platform, including the hardware and system software, such as operating systems and databases; the service provider is responsible for system updates, patches, and software maintenance. *PaaS* does not allow any user control of the operating system, security features, or the ability to install applications. Typical examples are *Google App Engine, Microsoft Azure*, and `Force.com`, provided by `Salesforce.com`.

The level of user control over the system in *IaaS* is different form *PaaS*. *IaaS* provides total control, whereas *PaaS* typically provides no control. Consequently, *IaaS* incurs administration costs similar to a traditional computing infrastructure, whereas the administrative costs are virtually zero for *PaaS*.

It is critical for a cloud user to carefully `read` the SLA and to understand the limitations of the liability a cloud provider is willing to accept. In many instances the liabilities do not apply to damages caused by a third party or to failures attributed either to the customer's hardware and software or to hardware and software from a third party.

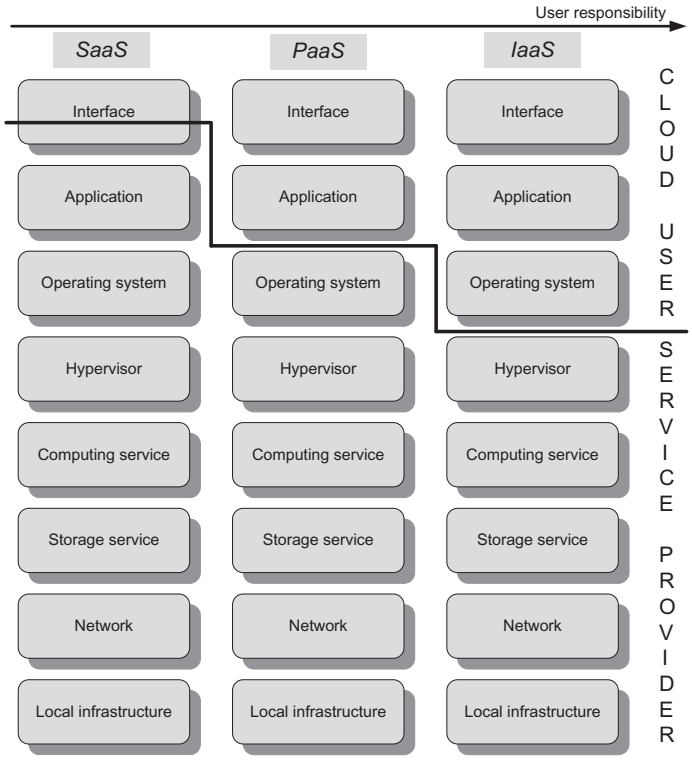

FIGURE 3.7

The limits of responsibility between a cloud user and the cloud service provider.

The limits of responsibility between the cloud user and the cloud service provider are different for the three service-delivery models, as we can see in Figure 3.7. In the case of *SaaS* the user is partially responsible for the interface; the user responsibility increases in the case of *PaaS* and includes the interface and the application. In the case of *IaaS* the user is responsible for all the events occurring in the virtual machine running the application.

For example, if a distributed denial-of-service attack (DDoS) causes the entire *IaaS* infrastructure to fail, the cloud service provider is responsible for the consequences of the attack. The user is responsible if the DDoS affects only several instances, including the ones running the user application. A recent posting describes the limits of responsibility illustrated in Figure 3.7 and argues that security should be a major concern for *IaaS* cloud users, (see `www.sans.org/cloud/2012/07/19/can-i-outsource-my-security-to-the-cloud`).

3.10 User experience

There have been a few studies of user experience based on a large population of cloud computing users. An empirical study of the experience of a small group of users of the Finish Cloud Computing

Consortium is reported in [279]. The main user concerns are security threats, the dependence on fast Internet connections that forced version updates, data ownership, and user behavior monitoring. All users reported that trust in the cloud services is important, two-thirds raised the point of fuzzy boundaries of liability between cloud user and provider, about half did not fully comprehend the cloud functions and its behavior, and about one-third were concerned about security threats.

The security threats perceived by this group of users are: (i) abuse and villainous use of the cloud; (ii) APIs that are not fully secure; (iii) malicious insiders; (iv) account hijacking; (iv) data leaks; and (v) issues related to shared resources. Identity theft and privacy were major concerns for about half of the users questioned; availability, liability, and data ownership and copyright were raised by a third of respondents.

The suggested solutions to these problems are as follows: SLAs and tools to monitor usage should be deployed to prevent abuse of the cloud; data encryption and security testing should enhance the API security; an independent security layer should be added to prevent threats caused by malicious insiders; strong authentication and authorization should be enforced to prevent account hijacking; data decryption in a secure environment should be implemented to prevent data leakage; and compartmentalization of components and firewalls should be deployed to limit the negative effect of resource sharing.

A broad set of concerns identified by the NIST working group on cloud security includes:

- Potential loss of control/ownership of data.
- Data integration, privacy enforcement, data encryption.
- Data remanence after deprovisioning.
- Multitenant data isolation.
- Data location requirements within national borders.
- Hypervisor security.
- Audit data integrity protection.
- Verification of subscriber policies through provider controls.
- Certification/accreditation requirements for a given cloud service.

A 2010 study conducted by IBM [176] aims to identify barriers to public and private cloud adoption. The study is based on interviews with more than 1,000 individuals responsible for IT decision making around the world. Seventy-seven percent of the respondents cited cost savings as the key argument in favor of public cloud adoption, though only 30% of them believed that public clouds are "very appealing or appealing" for their line of business, versus 64% for private clouds and 34% for hybrid ones.

The reasons driving the decision to use public clouds and the percentage of responders who considered each element critical are shown in Table 3.6. In view of the high energy costs for operating a data center (discussed in Section 3.7), it seems strange that only 29% of the respondents seem to be concerned about lower energy costs.

The top workloads mentioned by the users involved in this study are data mining and other analytics (83%), application streaming (83%), help desk services (80%), industry-specific applications (80%), and development environments (80%).

The study also identified workloads that are not good candidates for migration to a public cloud environment:

Table 3.6 The reasons driving the decision to use public clouds.

Reason	Respondents Who Agree
Improved system reliability and availability	50%
Pay only for what you use	50%
Hardware savings	47%
Software license savings	46%
Lower labor costs	44%
Lower maintenance costs	42%
Reduced IT support needs	40%
Ability to take advantage of the latest functionality	40%
Less pressure on internal resources	39%
Solve problems related to updating/upgrading	39%
Rapid deployment	39%
Ability to scale up resources to meet needs	39%
Ability to focus on core competencies	38%
Take advantage of the improved economies of scale	37%
Reduced infrastructure management needs	37%
Lower energy costs	29%
Reduced space requirements	26%
Create new revenue streams	23%

- Sensitive data such as employee and health care records.
- Multiple codependent services (e.g., online transaction processing).
- Third-party software without cloud licensing.
- Workloads requiring auditability and accountability.
- Workloads requiring customization.

Such studies help identify the concerns of potential cloud users and the critical issues for cloud research.

3.11 Software Licensing

Software licensing for cloud computing is an enduring problem without a universally accepted solution at this time. The license management technology is based on the old model of computing centers with licenses given on the basis of named users or as site licenses. This licensing technology, developed for a centrally managed environment, cannot accommodate the distributed service infrastructure of cloud computing or of grid computing.

Only very recently IBM reached an agreement allowing some of its software products to be used on *EC2*. Furthermore, MathWorks developed a business model for the use of MATLAB in grid environments [63]. The *Software-as-a-Service (SaaS)* deployment model is gaining acceptance because it allows users to pay only for the services they use.

There is significant pressure to change the traditional software licensing model and find nonhardware-based solutions for cloud computing. The increased negotiating power of users, coupled with the increase in software piracy, has renewed interest in alternative schemes such as those proposed by the *SmartLM* research project (www.smartlm.eu). *SmartLM* license management requires a complex software infrastructure involving SLA, negotiation protocols, authentication, and other management functions.

A commercial product based on the ideas developed by this research project is *elasticLM*, which provides license and billing for Web-based services [63]. The architecture of the *elasticLM* license service has several layers: coallocation, authentication, administration, management, business, and persistency. The authentication layer authenticates communication between the license service and the billing service as well as the individual applications; the persistence layer stores the usage records. The main responsibility of the business layer is to provide the licensing service with the licenses prices, and the management coordinates various components of the automated billing service.

When a user requests a license from the license service, the terms of the license usage are negotiated and they are part of an SLA document. The negotiation is based on application-specific templates and the license cost becomes part of the SLA. The SLA describes all aspects of resource usage, including the ID of application, duration, number of processors, and guarantees, such as the maximum cost and deadlines. When multiple negotiation steps are necessary, the WS-Agreement Negotiation protocol is used.

To understand the complexity of the issues related to software licensing, we point out some of the difficulties related to authorization. To verify the authorization to use a license, an application must have the certificate of an authority. This certificate must be available locally to the application because the application may be executed in an environment with restricted network access. This opens up the possibility for an administrator to hijack the license mechanism by exchanging the local certificate.

3.12 Further reading

Information about cloud computing at Amazon, Google, Microsoft, HP, and Oracle is available from the following sites:

- Amazon: http://aws.amazon.com/ec2/
- Google: http://code.google.com/appengine/
- Microsoft: www.microsoft.com/windowsazure/
- HP: www.hp.com/go/cloud
- Oracle: http://cloud.oracle.com

Several sites provide additional information about the open-source platforms *Eucalyptus*, *OpenNebula*, and *Nimbus*:

- Eucalyptus: www.eucalyptus.com
- Open-Nebula: www.opennebula.org
- Nimbus: www.nimbusproject.org

A white paper on SLA specification can be found at www.itsm.info, a toolkit at www.service-level-agreement.net, and a Web service-level agreement (WSLA) at www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

Energy use and ecological impact are discussed in [6,158,295,358,364].

Information about the *OpenStack*, an open-source cloud operating system, is available from the project site www.openstack.org. The Intercloud is discussed in several papers, including [47–49]. Alternative architectures for cloud computing have been proposed [111].

Several other references including [221,278,283,302,310,378] cover important aspects of the cloud infrastructure.

3.13 History notes

Amazon was one of the first providers of cloud computing. One year after the beta release of *EC2* in 2006, two new instance types (Large and Extra-Large) were added, followed in 2008 by two more types, *High-CPU Medium* and *High-CPU Extra-Large*. New features include static IP addresses, availability zones, and user-selectable kernels as well as the Block Store (EBS). Amazon *EC2* has been in full production mode since October 2008 and supports an SLA and the *Microsoft Windows* operating system as well as the Microsoft SQL Server.

3.14 Exercises and problems

Problem 1. Several desirable properties of a large-scale distributed system were discussed in Section 2.3. The list includes transparency of access, location, concurrency, replication, failure, migration, performance, and scaling. Analyze how each one of these properties applies to *AWS*.

Problem 2. Compare the Oracle Cloud offerings (see https://cloud.oracle.com) with the cloud services provided by Amazon, Google, and Microsoft.

Problem 3. Read the IBM report [176] and discuss the workload preferences for private and public clouds and the reasons for the preferences.

Problem 4. In Section 3.7 we introduced the concept of energy-proportional systems and we saw that different system components have different dynamic ranges. Sketch a strategy to reduce the power consumption in a lightly loaded cloud, and discuss the steps for placing a computational server in standby mode and then for bringing it back up to active mode.

Problem 5. Read the paper that introduced the concept of dataspaces [127] and analyze the benefits and the problems with this new idea. Research the literature for potential application of dataspaces for scientific data management in a domain of your choice, be it the search for the Higgs boson at CERN, structural biology, cancer research, or another important research topic that involves data-intensive applications.

Problem 6. In Section 3.7 it was mentioned that *InfiniBand* can be considered an energy-proportional network. The network is used by supercomputers (see `http://i.top500.org/`); the InfiniBand fabric is also used to connect compute nodes, compute nodes with storage servers, and Exadata and Exalogic systems at Oracle data centers. Analyze the features of *InfiniBand* that are critical to reduction of energy consumption.

Problem 7. Many organizations operate one or more computer clusters and contemplate the migration to private clouds. What are the arguments for and against such an effort?

Problem 8. Evaluate the SLA toolkit at `www.service-level-agreement.net`. Is the interactive guide useful? What does it miss? Does the SLA template include all the clauses that are important in your view? If not, what is missing? Are the examples helpful?

Problem 9. Software licensing is a major problem in cloud computing. Discuss several ideas to prevent an administrator from hijacking the authorization to use a software license.

Problem 10. Annotation schemes are widely used by popular services such as the Flickr photo-sharing service, which supports annotation of photos. Sketch the organization of a cloud service used for sharing medical X-ray, tomography, CAT scan, and other medical images and discuss the main challenges for its implementation.