CHAPTER

Data-Intensive Computing
MapReduce Programming

8

Data-intensive computing focuses on aa class of applications that deal with a large amount of data. Several application fields, ranging from computational science to social networking, produce large volumes of data that need to be efficiently stored, made accessible, indexed, and analyzed. These tasks become challenging as the quantity of information accumulates and increases over time at higher rates. Distributed computing is definitely of help in addressing these challenges by providing more scalable and efficient storage architectures and a better performance in terms of data computation and processing. Despite this fact, the use of parallel and distributed techniques as a support of data-intensive computing is not straightforward, but several challenges in the form of data representation, efficient algorithms, and scalable infrastructures need to be faced.

This chapter characterizes the nature of data-intensive computing and presents an overview of the challenges introduced by production of large volumes of data and how they are handled by storage systems and computing models. It describes *MapReduce*, which is a popular programming model for creating data-intensive applications and their deployment on clouds. Practical examples of MapReduce applications for data-intensive computing are demonstrated using the Aneka MapReduce Programming Model.

8.1 What is data-intensive computing?

Data-intensive computing is concerned with production, manipulation, and analysis of large-scale data in the range of hundreds of megabytes (MB) to petabytes (PB) and beyond [73]. The term *dataset* is commonly used to identify a collection of information elements that is relevant to one or more applications. Datasets are often maintained in *repositories*, which are infrastructures supporting the storage, retrieval, and indexing of large amounts of information. To facilitate the classification and search, relevant bits of information, called *metadata*, are attached to datasets.

Data-intensive computations occur in many application domains. Computational science is one of the most popular ones. People conducting scientific simulations and experiments are often keen to produce, analyze, and process huge volumes of data. Hundreds of gigabytes of data are produced every second by telescopes mapping the sky; the collection of images of the sky easily reaches the scale of petabytes over a year. Bioinformatics applications mine databases that may end up containing terabytes of data. Earthquake simulators process a massive amount of data, which is produced as a result of recording the vibrations of the Earth across the entire globe.

Besides scientific computing, several IT industry sectors require support for data-intensive computations. Customer data for any telecom company would easily be in the range of 10−100 terabytes. This volume of information is not only processed to generate billing statements, but it is also mined to identify scenarios, trends, and patterns that help these companies provide better service. Moreover, it is reported that U.S. handset mobile traffic has reached 8 petabytes per month and it is expected to grow up to 327 petabytes per month by 2015.[1] The scale of petabytes is even more common when we consider IT giants such as Google, which is reported to process about 24 petabytes of information per day [55] and to sort petabytes of data in hours.[2] Social networking and gaming are two other sectors in which data-intensive computing is now a reality. Facebook inbox search operations involve crawling about 150 terabytes of data, and the whole uncompressed data stored by the distributed infrastructure reach to 36 petabytes.[3] Zynga, *a* social gaming platform, moves 1 petabyte of data daily and it has been reported to add 1,000 servers every week to store the data generated by games like Farmville and Frontierville.[4]

8.1.1 Characterizing data-intensive computations

Data-intensive applications not only deal with huge volumes of data but, very often, also exhibit compute-intensive properties [74]. Figure 8.1 identifies the domain of data-intensive computing in the two upper quadrants of the graph.

Data-intensive applications handle datasets on the scale of multiple terabytes and petabytes. Datasets are commonly persisted in several formats and distributed across different locations. Such applications process data in multistep analytical pipelines, including transformation and fusion stages. The processing requirements scale almost linearly with the data size, and they can be easily processed in parallel. They also need efficient mechanisms for data management, filtering and fusion, and efficient querying and distribution [74].

8.1.2 Challenges ahead

The huge amount of data produced, analyzed, or stored imposes requirements on the supporting infrastructures and middleware that are hardly found in the traditional solutions for distributed computing. For example, the location of data is crucial as the need for moving terabytes of data becomes an obstacle for high-performing computations. Data partitioning as well as content replication and scalable algorithms help in improving the performance of data-intensive applications. Open challenges in data-intensive computing given by Ian Gorton et al. [74] are:

- Scalable algorithms that can search and process massive datasets
- New metadata management technologies that can scale to handle complex, heterogeneous, and distributed data sources

[1]Coda Research Consultancy, www.codaresearch.co.uk/usmobileinternet/index.htm.
[2]Google's Blog, http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html.
[3]OSCON 2010, David Recordon (Senior Open Programs Manager, Facebook): Today's LAMP Stack, Keynote Speech. Available at www.oscon.com/oscon2010/public/schedule/speaker/2442.
[4]http://techcrunch.com/2010/09/22/zynga-moves-1-petabyte-of-data-daily-adds-1000-servers-a-week/.

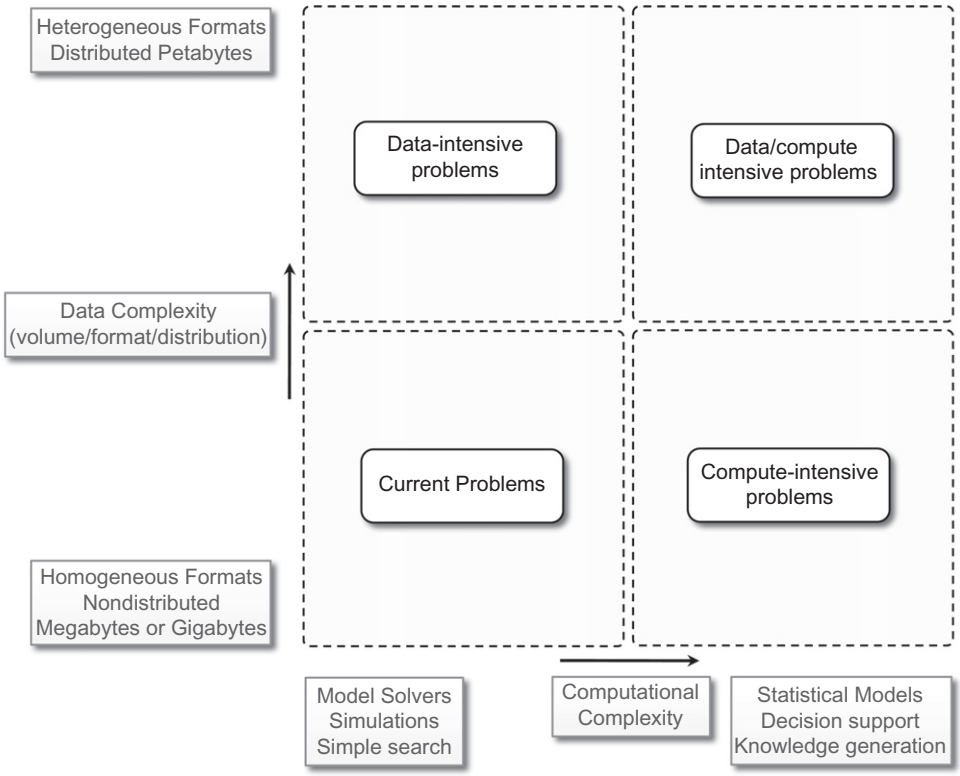

FIGURE 8.1

Data-intensive research issues.

- Advances in high-performance computing platforms aimed at providing a better support for accessing in-memory multiterabyte data structures
- High-performance, highly reliable, petascale distributed file systems
- Data signature-generation techniques for data reduction and rapid processing
- New approaches to software mobility for delivering algorithms that are able to move the computation to where the data are located
- Specialized hybrid interconnection architectures that provide better support for filtering multigigabyte datastreams coming from high-speed networks and scientific instruments
- Flexible and high-performance software integration techniques that facilitate the combination of software modules running on different platforms to quickly form analytical pipelines

8.1.3 Historical perspective

Data-intensive computing involves the production, management, and analysis of large volumes of data. Support for data-intensive computations is provided by harnessing storage, networking

technologies, algorithms, and infrastructure software all together. We track the evolution of this phenomenon by highlighting the most relevant contributions in the area of storage and networking and infrastructure software.

8.1.3.1 The early age: high-speed wide-area networking

The evolution of technologies, protocols, and algorithms for data transmission and streaming has been an enabler of data-intensive computations [75]. In 1989, the first experiments in high-speed networking as a support for remote visualization of scientific data led the way. Two years later, the potential of using high-speed wide area networks for enabling high-speed, TCP/IP-based distributed applications was demonstrated at Supercomputing 1991 (SC91). On that occasion, the remote visualization of large and complex scientific datasets (a high-resolution magnetic resonance image, or MRI, scan of the human brain) was set up between the Pittsburgh Supercomputing Center (PSC) and Albuquerque, New Mexico, the location of the conference.

A further step was made by the Kaiser project [76], which made available as remote data sources high data rate and online instrument systems. The project leveraged the Wide Area Large Data Object (WALDO) system [77], which was used to provide the following capabilities: automatic generation of metadata; automatic cataloguing of data and metadata while processing the data in real time; facilitation of cooperative research by providing local and remote users access to data; and mechanisms to incorporate data into databases and other documents.

The first data-intensive environment is reported to be the MAGIC project, a DARPA-funded collaboration working on distributed applications in large-scale, high-speed networks. Within this context, the *Distributed Parallel Storage System (DPSS)* was developed, later used to support *TerraVision* [78], a terrain visualization application that lets users explore and navigate a tridimensional real landscape.

Another important milestone was set with the Clipper project,[5] a collaborative effort of several scientific research laboratories, with the goal of designing and implementing a collection of independent but architecturally consistent service components to support data-intensive computing. The challenges addressed by the Clipper project included management of substantial computing resources, generation or consumption of high-rate and high-volume data flows, human interaction management, and aggregation of disperse resources (multiple data archives, distributed computing capacity, distributed cache capacity, and guaranteed network capacity). Clipper's main focus was to develop a coordinated collection of services that can be used by a variety of applications to build on-demand, large-scale, high-performance, wide-area problem-solving environments.

8.1.3.2 Data grids

With the advent of grid computing [8], huge computational power and storage facilities could be obtained by harnessing heterogeneous resources across different administrative domains. Within this context, *data grids* [79] emerge as infrastructures that support data-intensive computing. A data grid provides services that help users discover, transfer, and manipulate large datasets stored in distributed repositories as well as create and manage copies of them. Data grids offer two main functionalities: high-performance and reliable file transfer for moving large amounts of data, and scalable replica

[5] www.nersc.gov/news/annual_reports/annrep98/16clipper.html.

discovery and management mechanisms for easy access to distributed datasets [80]. Because data grids span different administration boundaries, access control and security are important concerns.

Data grids mostly provide storage and dataset management facilities as support for scientific experiments that produce huge volumes of data. The reference scenario might be one depicted in Figure 8.2. Huge amounts of data are produced by scientific instruments (telescopes, particle accelerators, etc.). The information, which can be locally processed, is then stored in repositories and made available for experiments and analysis to scientists, who can be local or, most likely, remote. Scientists can leverage specific discovery and information services, which help in determining the locations of the closest datasets of interest for their experiments. Datasets are replicated by the infrastructure to provide better availability. Since processing of this information also requires a large computational power, specific computing sites can be accessed to perform analysis and experiments.

Like any other grid infrastructure, heterogeneity of resources and different administrative domains constitute a fundamental aspect that needs to be properly addressed with security measures and the use of *virtual organizations (VO)*. Besides heterogeneity and security, data grids have their own characteristics and introduce new challenges [79]:

- *Massive datasets.* The size of datasets can easily be on the scale of gigabytes, terabytes, and beyond. It is therefore necessary to minimize latencies during bulk transfers, replicate content with appropriate strategies, and manage storage resources.
- *Shared data collections.* Resource sharing includes distributed collections of data. For example, repositories can be used to both store and read data.
- *Unified namespace.* Data grids impose a unified logical namespace where to locate data collections and resources. Every data element has a single logical name, which is eventually mapped to different physical filenames for the purpose of replication and accessibility.
- *Access restrictions.* Even though one of the purposes of data grids is to facilitate sharing of results and data for experiments, some users might want to ensure confidentiality for their data and restrict access to them to their collaborators. Authentication and authorization in data grids involve both coarse-grained and fine-grained access control over shared data collections.

With respect to the combination of several computing facilities through high-speed networking, data grids constitute a more structured and integrated approach to data-intensive computing. As a result, several scientific research fields, including high-energy physics, biology, and astronomy, leverage data grids, as briefly discussed here:

- *The LHC Grid.* A project funded by the European Union to develop a worldwide grid computing environment for use by high-energy physics researchers around the world who are collaborating on the Large Hadron Collider (LHC) experiment. It supports storage and analysis of large-scale datasets, from hundreds of terabytes to petabytes, generated by the LHC experiment (http://lhc.web.cern.ch/lhc/).
- *BioInformatics Research Network (BIRN).* BIRN is a national initiative to advance biomedical research through data sharing and online collaboration. Funded by the National Center for Research Resources (NCRR), a component of the U.S. National Institutes of Health (NIH), BIRN provides a data-sharing infrastructure, software tools, and strategies and advisory services (www.birncommunity.org).

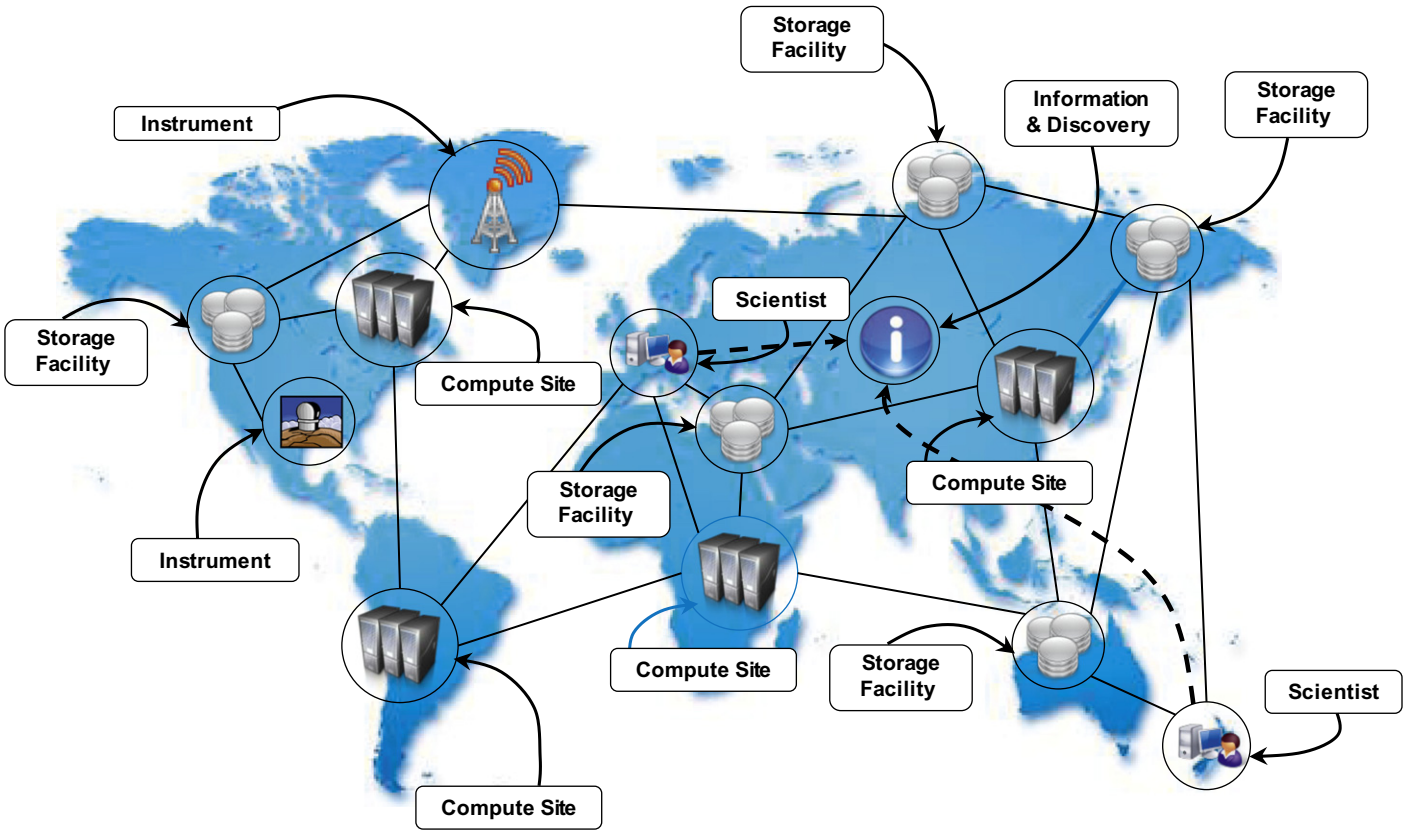

FIGURE 8.2

Data grid reference scenario.

- *International Virtual Observatory Alliance (IVOA)*. IVOA is an organization that aims to provide improved access to the ever-expanding astronomical data resources available online. It does so by promoting standards for *virtual observatories*, which are a collection of interoperating data archives and software tools that use the Internet to form a scientific research environment in which astronomical research programs can be conducted. This allows scientists to discover, access, analyze, and combine lab data from heterogeneous data collections (www.ivoa.net).

A complete taxonomy of Data Grids can be found in Venugopal et al. [79].

8.1.3.3 Data clouds and "Big Data"

Large datasets have mostly been the domain of scientific computing. This scenario has recently started to change as massive amounts of data are being produced, mined, and crunched by companies that provide Internet services such as searching, online advertising, and social media. It is critical for such companies to efficiently analyze these huge datasets because they constitute a precious source of information about their customers. *Log analysis* is an example of a data-intensive operation that is commonly performed in this context; companies such as Google have a massive amount of data in the form of logs that are daily processed using their distributed infrastructure. As a result, they settled upon an analytic infrastructure, which differs from the grid-based infrastructure used by the scientific community.

Together with the diffusion of cloud computing technologies that support data-intensive computations, the term *Big Data* [82] has become popular. This term characterizes the nature of data-intensive computations today and currently identifies datasets that grow so large that they become complex to work with using on-hand database management tools. Relational databases and desktop statistics/visualization packages become ineffective for that amount of information, instead requiring "massively parallel software running on tens, hundreds, or even thousands of servers" [82].

Big Data problems are found in nonscientific application domains such as weblogs, radio frequency identification (RFID), sensor networks, social networks, Internet text and documents, Internet search indexing, call detail records, military surveillance, medical records, photography archives, video archives, and large scale ecommerce. Other than the massive size, what characterizes all these examples is that new data are accumulated with time rather than replacing the old data. In general, the term *Big Data* applies to datasets of which the size is beyond the ability of commonly used software tools to capture, manage, and process within a tolerable elapsed time. Therefore, Big Data sizes are a constantly moving target, currently ranging from a few dozen terabytes to many petabytes of data in a single dataset [82].

Cloud technologies support data-intensive computing in several ways:

- By providing a large amount of compute instances on demand, which can be used to process and analyze large datasets in parallel.
- By providing a storage system optimized for keeping large blobs of data and other distributed data store architectures.
- By providing frameworks and programming APIs optimized for the processing and management of large amounts of data. These APIs are mostly coupled with a specific storage infrastructure to optimize the overall performance of the system.

A *data cloud* is a combination of these components. An example is the *MapReduce* framework [55], which provides the best performance for leveraging the Google File System [54] on top of Google's large computing infrastructure. Another example is the *Hadoop* system [83], the most mature, large, and open-source data cloud. It consists of the Hadoop Distributed File System (HDFS) and Hadoop's implementation of MapReduce. A similar approach is proposed by *Sector* [84], which consists of the Sector Distributed File System (SDFS) and a compute service called *Sphere* [84] that allows users to execute arbitrary user-defined functions (UDFs) over the data managed by SDFS. *Greenplum* uses a shared-nothing massively parallel processing (MPP) architecture based on commodity hardware. The architecture also integrates MapReduce-like functionality into its platform. A similar architecture has been deployed by *Aster*, which uses an MPP-based data-warehousing appliance that supports MapReduce and targets 1 PB of data.

8.1.3.4 Databases and data-intensive computing

Traditionally, distributed databases [85] have been considered the natural evolution of database management systems as the scale of the datasets becomes unmanageable with a single system. Distributed databases are a collection of data stored at different sites of a computer network. Each site might expose a degree of autonomy, providing services for the execution of local applications, but also participating in the execution of a global application. A distributed database can be created by splitting and scattering the data of an existing database over different sites or by federating together multiple existing databases. These systems are very robust and provide distributed transaction processing, distributed query optimization, and efficient management of resources. However, they are mostly concerned with datasets that can be expressed using the relational model [86], and the need to enforce ACID properties on data limits their abilities to scale as data clouds and grids do.

8.2 Technologies for data-intensive computing

Data-intensive computing concerns the development of applications that are mainly focused on processing large quantities of data. Therefore, storage systems and programming models constitute a natural classification of the technologies supporting data-intensive computing.

8.2.1 Storage systems

Traditionally, database management systems constituted the *de facto* storage support for several types of applications. Due to the explosion of unstructured data in the form of blogs, Web pages, software logs, and sensor readings, the relational model in its original formulation does not seem to be the preferred solution for supporting data analytics on a large scale [88]. Research on databases and the data management industry are indeed at a turning point, and new opportunities arise. Some factors contributing to this change are:

- *Growing of popularity of Big Data.* The management of large quantities of data is no longer a rare case but instead has become common in several fields: scientific computing, enterprise applications, media entertainment, natural language processing, and social network analysis. The large volume of data imposes new and more efficient techniques for data management.

- *Growing importance of data analytics in the business chain.* The management of data is no longer considered a cost but a key element of business profit. This situation arises in popular social networks such as Facebook, which concentrate their focus on the management of user profiles, interests, and connections among people. This massive amount of data, which is constantly mined, requires new technologies and strategies to support data analytics.
- *Presence of data in several forms, not only structured.* As previously mentioned, what constitutes relevant information today exhibits a heterogeneous nature and appears in several forms and formats. Structured data are constantly growing as a result of the continuous use of traditional enterprise applications and system, but at the same time the advances in technology and the democratization of the Internet as a platform where everyone can pull information has created a massive amount of information that is unstructured and does not naturally fit into the relational model.
- *New approaches and technologies for computing.* Cloud computing promises access to a massive amount of computing capacity on demand. This allows engineers to design software systems that incrementally scale to arbitrary degrees of parallelism. It is no longer rare to build software applications and services that are dynamically deployed on hundreds or thousands of nodes, which might belong to the system for a few hours or days. Classical database infrastructures are not designed to provide support to such a volatile environment.

All these factors identify the need for new data management technologies. This not only implies a new research agenda in database technologies and a more holistic approach to the management of information but also leaves room for alternatives (or complements) to the relational model. In particular, advances in distributed file systems for the management of raw data in the form of files, distributed object stores, and the spread of the NoSQL movement constitute the major directions toward support for data-intensive computing.

8.2.1.1 High-performance distributed file systems and storage clouds

Distributed file systems constitute the primary support for data management. They provide an interface whereby to store information in the form of files and later access them for read and write operations. Among the several implementations of file systems, few of them specifically address the management of huge quantities of data on a large number of nodes. Mostly these file systems constitute the data storage support for large computing clusters, supercomputers, massively parallel architectures, and lately, storage/computing clouds.

Lustre. The Lustre file system is a massively parallel distributed file system that covers the needs of a small workgroup of clusters to a large-scale computing cluster. The file system is used by several of the Top 500 supercomputing systems, including the one rated the most powerful supercomputer in the June 2012 list.[6] Lustre is designed to provide access to petabytes (PBs) of storage to serve thousands of clients with an I/O throughput of hundreds of gigabytes per second (GB/s). The system is composed of a metadata server that contains the metadata about the file system and a collection of object storage servers that are in charge of providing storage. Users access the file system via a POSIX-compliant client, which can be either mounted as a module in the

[6]Top 500 supercomputers list: www.top500.org (accessed in June 2012).

kernel or through a library. The file system implements a robust failover strategy and recovery mechanism, making server failures and recoveries transparent to clients.

IBM General Parallel File System (GPFS). GPFS [88] is the high-performance distributed file system developed by IBM that provides support for the RS/6000 supercomputer and Linux computing clusters. GPFS is a multiplatform distributed file system built over several years of academic research and provides advanced recovery mechanisms. GPFS is built on the concept of shared disks, in which a collection of disks is attached to the file system nodes by means of some switching fabric. The file system makes this infrastructure transparent to users and stripes large files over the disk array by replicating portions of the file to ensure high availability. By means of this infrastructure, the system is able to support petabytes of storage, which is accessed at a high throughput and without losing consistency of data. Compared to other implementations, GPFS distributes the metadata of the entire file system and provides transparent access to it, thus eliminating a single point of failure.

Google File System (GFS). GFS [54] is the storage infrastructure that supports the execution of distributed applications in Google's computing cloud. The system has been designed to be a fault-tolerant, highly available, distributed file system built on commodity hardware and standard Linux operating systems. Rather than a generic implementation of a distributed file system, GFS specifically addresses Google's needs in terms of distributed storage for applications, and it has been designed with the following assumptions:

- The system is built on top of commodity hardware that often fails.
- The system stores a modest number of large files; multi-GB files are common and should be treated efficiently, and small files must be supported, but there is no need to optimize for that.
- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads.
- The workloads also have many large, sequential writes that append data to files.
- High-sustained bandwidth is more important than low latency.

The architecture of the file system is organized into a single master, which contains the metadata of the entire file system, and a collection of chunk servers, which provide storage space. From a logical point of view the system is composed of a collection of software daemons, which implement either the master server or the chunk server. A file is a collection of chunks for which the size can be configured at file system level. Chunks are replicated on multiple nodes in order to tolerate failures. Clients look up the master server and identify the specific chunk of a file they want to access. Once the chunk is identified, the interaction happens between the client and the chunk server. Applications interact through the file system with a specific interface supporting the usual operations for file creation, deletion, read, and write. The interface also supports *snapshots* and *record append* operations that are frequently performed by applications. GFS has been conceived by considering that failures in a large distributed infrastructure are common rather than a rarity; therefore, specific attention has been given to implementing a highly available, lightweight, and fault-tolerant infrastructure. The potential single point of failure of the single-master architecture has been addressed by giving the possibility of replicating the master node on any other node belonging to the infrastructure. Moreover, a stateless daemon and extensive logging capabilities facilitate the system's recovery from failures.

Sector. Sector [84] is the storage cloud that supports the execution of data-intensive applications defined according to the Sphere framework. It is a user space file system that can be deployed on commodity hardware across a wide-area network. Compared to other file systems, Sector does not partition a file into blocks but replicates the entire files on multiple nodes, allowing users to customize the replication strategy for better performance. The system's architecture is composed of four nodes: a security server, one or more master nodes, slave nodes, and client machines. The security server maintains all the information about access control policies for user and files, whereas master servers coordinate and serve the I/O requests of clients, which ultimately interact with slave nodes to access files. The protocol used to exchange data with slave nodes is UDT [89], which is a lightweight connection-oriented protocol optimized for wide-area networks.

Amazon Simple Storage Service (S3). Amazon S3 is the online storage service provided by Amazon. Even though its internal details are not revealed, the system is claimed to support high availability, reliability, scalability, infinite storage, and low latency at commodity cost. The system offers a flat storage space organized into buckets, which are attached to an Amazon Web Services (AWS) account. Each bucket can store multiple objects, each identified by a unique key. Objects are identified by unique URLs and exposed through HTTP, thus allowing very simple *get-put* semantics. Because of the use of HTTP, there is no need for any specific library for accessing the storage system, the objects of which can also be retrieved through the Bit Torrent protocol.[7] Despite its simple semantics, a POSIX-like client library has been developed to mount S3 buckets as part of the local file system. Besides the minimal semantics, security is another limitation of S3. The visibility and accessibility of objects are linked to AWS accounts, and the owner of a bucket can decide to make it visible to other accounts or the public. It is also possible to define authenticated URLs, which provide public access to anyone for a limited (and configurable) period of time.

Besides these examples of storage systems, there exist other implementations of distributed file systems and storage clouds that have architecture that is similar to the models discussed here. Except for the S3 service, it is possible to sketch a general reference architecture in all the systems presented that identifies two major roles into which all the nodes can be classified. Metadata or master nodes contain the information about the location of files or file chunks, whereas slave nodes are used to provide direct access to the storage space. The architecture is completed by client libraries, which provide a simple interface for accessing the file system, which is to some extent or completely compliant to the POSIX specification. Variations of the reference architecture can include the ability to support multiple masters, to distribute the metadata over multiple nodes, or to easily interchange the role of nodes. The most important aspect common to all these different implementations is the ability to provide fault-tolerant and highly available storage systems.

8.2.1.2 NoSQL systems

The term *Not Only SQL (NoSQL)* was originally coined in 1998 to identify a relational database that did not expose a SQL interface to manipulate and query data but relied on a set of UNIX shell scripts and commands to operate on text files containing the actual data. In a very strict sense, NoSQL cannot be considered a relational database since it is not a monolithic piece of software organizing information according to the relational model, but rather is a collection of scripts that

[7] Bit Torrent is a P2P file-sharing protocol used to distribute large amounts of data. The key characteristic of the protocol is the ability to allow users to download a file in parallel from multiple hosts.

allow users to manage most of the simplest and more common database tasks by using text files as information stores. Later, in 2009, the term *NoSQL* was reintroduced with the intent of labeling all those database management systems that did not use a relational model but provided simpler and faster alternatives for data manipulation. Nowadays, the term *NoSQL* is a big umbrella encompassing all the storage and database management systems that differ in some way from the relational model. Their general philosophy is to overcome the restrictions imposed by the relational model and to provide more efficient systems. This often implies the use of tables without fixed schemas to accommodate a larger range of data types or avoid joins to increase the performance and scale horizontally.

Two main factors have determined the growth of the NoSQL movement: in many cases simple data models are enough to represent the information used by applications, and the quantity of information contained in unstructured formats has grown considerably in the last decade. These two factors made software engineers look to alternatives that were more suitable to specific application domains they were working on. As a result, several different initiatives explored the use of nonrelational storage systems, which considerably differ from each other. A broad classification is reported by Wikipedia,[8] which distinguishes NoSQL implementations into:

- *Document stores* (Apache Jackrabbit, Apache CouchDB, SimpleDB, Terrastore).
- *Graphs* (AllegroGraph, Neo4j, FlockDB, Cerebrum).
- *Key-value stores*. This is a macro classification that is further categorized into key-value stores on disk, key-value caches in RAM, hierarchically key-value stores, eventually consistent key-value stores, and ordered key-value store.
- *Multivalue databases* (OpenQM, Rocket U2, OpenInsight).
- *Object databases* (ObjectStore, JADE, ZODB).
- *Tabular stores* (Google BigTable, Hadoop HBase, Hypertable).
- *Tuple stores* (Apache River).

Let us now examine some prominent implementations that support data-intensive applications.

Apache CouchDB and MongoDB. Apache CouchDB [91] and MongoDB [90] are two examples of document stores. Both provide a schema-less store whereby the primary objects are documents organized into a collection of key-value fields. The value of each field can be of type string, integer, float, date, or an array of values. The databases expose a RESTful interface and represent data in JSON format. Both allow querying and indexing data by using the MapReduce programming model, expose JavaScript as a base language for data querying and manipulation rather than SQL, and support large files as documents. From an infrastructure point of view, the two systems support data replication and high availability. CouchDB ensures ACID properties on data. MongoDB supports *sharding*, which is the ability to distribute the content of a collection among different nodes.

Amazon Dynamo. Dynamo [92] is the distributed key-value store that supports the management of information of several of the business services offered by Amazon Inc. The main goal of Dynamo is to provide an incrementally scalable and highly available storage system. This goal helps in achieving reliability at a massive scale, where thousands of servers and network components build an infrastructure serving 10 million requests per day. Dynamo provides a simplified

[8] http://en.wikipedia.com/wiki/NoSQL.

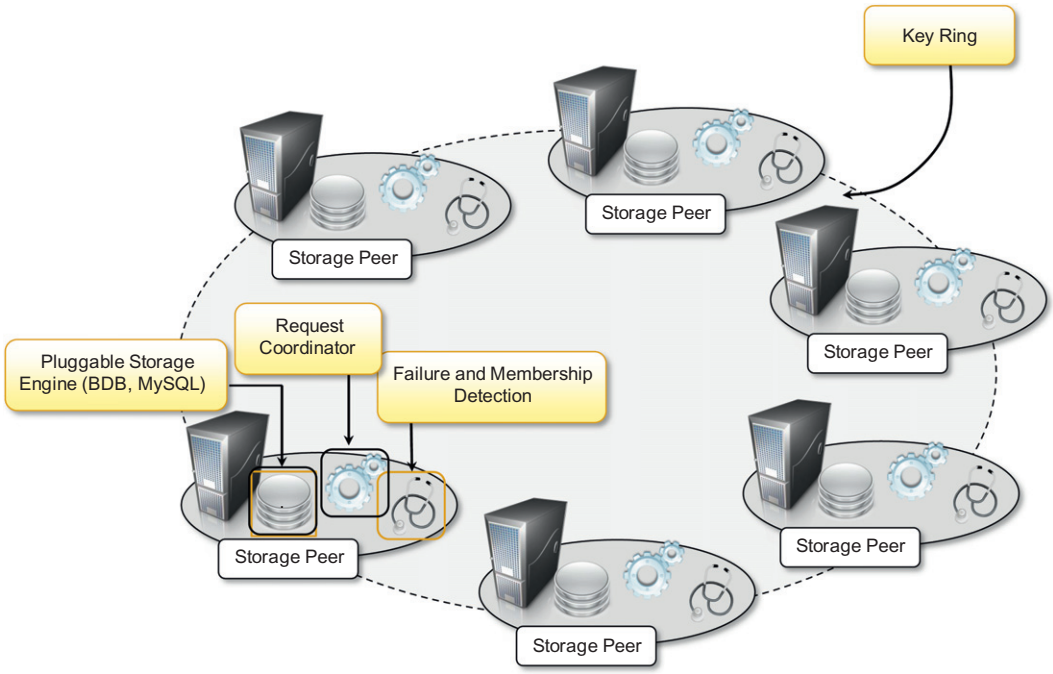

FIGURE 8.3

Amazon Dynamo architecture.

interface based on *get/put* semantics, where objects are stored and retrieved with a unique identifier (key). The main goal of achieving an extremely reliable infrastructure has imposed some constraints on the properties of these systems. For example, ACID properties on data have been sacrificed in favor of a more reliable and efficient infrastructure. This creates what it is called an *eventually consistent* model, which means that in the long term all the users will see the same data.

The architecture of the Dynamo system, shown in Figure 8.3, is composed of a collection of storage peers organized in a ring that shares the key space for a given application. The key space is partitioned among the storage peers, and the keys are replicated across the ring, avoiding adjacent peers. Each peer is configured with access to a local storage facility where original objects and replicas are stored. Furthermore, each node provides facilities for distributing the updates among the rings and to detect failures and unreachable nodes. With some relaxation of the consistency model applied to replicas and the use of object versioning, Dynamo implements the capability of being an *always-writable store*, where consistency of data is resolved in the background. The downside of such an approach is the simplicity of the storage model, which requires applications to build their own data models on top of the simple building blocks provided by the store. For example, there are no referential integrity constraints, relationships are not embedded in the storage model, and therefore join operations are not supported. These restrictions are not prohibitive in the case of Amazon services for which the single key-value model is acceptable.

Google Bigtable. Bigtable [93] is the distributed storage system designed to scale up to petabytes of data across thousands of servers. Bigtable provides storage support for several Google applications that expose different types of workload: from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. Bigtable's key design goals are wide applicability, scalability, high performance, and high availability. To achieve these goals, Bigtable organizes the data storage in tables of which the rows are distributed over the distributed file system supporting the middleware, which is the Google File System. From a logical point of view, a table is a multidimensional sorted map indexed by a key that is represented by a string of arbitrary length. A table is organized into rows and columns; columns can be grouped in column family, which allow for specific optimization for better access control, the storage and the indexing of data. A simple data access model constitutes the interface for client applications that can address data at the granularity level of the single column of a row. Moreover, each column value is stored in multiple versions that can be automatically time-stamped by Bigtable or by the client applications.

Besides the basic data access, Bigtable APIs also allow more complex operations such as single row transactions and advanced data manipulation by means of the Sazwall[9] [95] scripting language or the MapReduce APIs.

Figure 8.4 gives an overview of the infrastructure that enables Bigtable. The service is the result of a collection of processes that coexist with other processes in a cluster-based environment. Bigtable identifies two kinds of processes: master processes and tablet server processes. A tablet server is responsible for serving the requests for a given tablet that is a contiguous partition of rows of a table. Each server can manage multiple tablets (commonly from 10 to 1,000). The master server is responsible for keeping track of the status of the tablet servers and of the allocation of tablets to tablet servers. The server constantly monitors the tablet servers to check whether they are alive, and in case they are not reachable, the allocated tablets are reassigned and eventually partitioned to other servers.

Chubby [96]—a distributed, highly available, and persistent lock service—supports the activity of the master and tablet servers. System monitoring and data access are filtered through Chubby, which is also responsible for managing replicas and providing consistency among them. At the very bottom layer, the data are stored in the Google File System in the form of files, and all the update operations are logged into the file for the easy recovery of data in case of failures or when tablets need to be reassigned to other servers. Bigtable uses a specific file format for storing the data of a tablet, which can be compressed for optimizing the access and storage of data.

Bigtable is the result of a study of the requirements of several distributed applications in Google. It serves as a storage back-end for 60 applications (such as Google Personalized Search, Google Anlytics, Google Finance, and Google Earth) and manages petabytes of data.

Apache Cassandra. Cassandra [94] is a distributed object store for managing large amounts of structured data spread across many commodity servers. The system is designed to avoid a single point of failure and offer a highly reliable service. Cassandra was initially developed by Facebook; now it is part of the Apache incubator initiative. Currently, it provides storage support for several very large Web applications such as Facebook itself, Digg, and Twitter. Cassandra is defined as a

[9] *Sazwall* is an interpreted procedural programming language developed at Google for the manipulation of large quantities of tabular data. It includes specific capabilities for supporting statistical aggregation of values read or computed from the input and other features that simplify the parallel processing of petabytes of data.

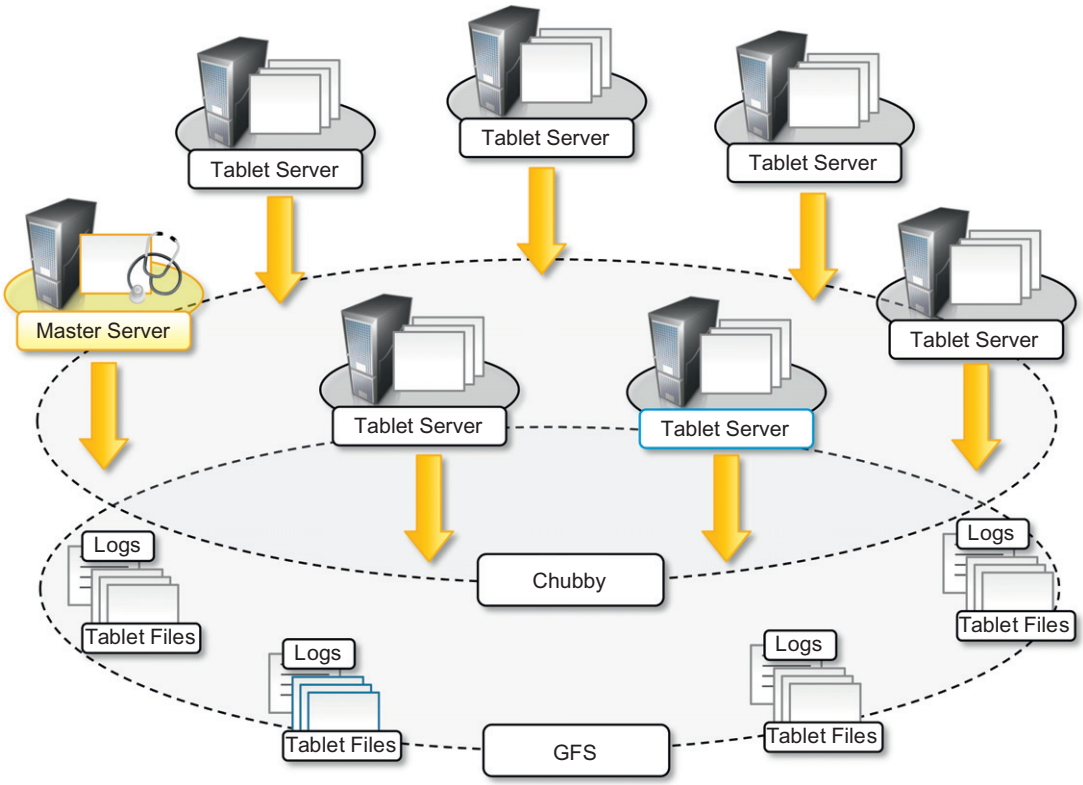

FIGURE 8.4

Bigtable architecture.

second-generation distributed database that builds on the concept of Amazon Dynamo, which follows a fully distributed design, and Google Bigtable, from which it inherits the "column family" concept. The data model exposed by Cassandra is based on the concept of a table that is implemented as a distributed multidimensional map indexed by a key. The value corresponding to a key is a highly structured object and constitutes the row of a table. Cassandra organizes the row of a table into columns, and sets of columns can be grouped into column families. The APIs provided by the system to access and manipulate the data are very simple: insertion, retrieval, and deletion. The insertion is performed at the row level; retrieval and deletion can operate at the column level.

In terms of the infrastructure, Cassandra is very similar to Dynamo. It has been designed for incremental scaling, and it organizes the collection of nodes sharing a key space into a ring. Each node manages multiple and discontinuous portions of the key space and replicates its data up to N other nodes. Replication uses different strategies; it can be *rack aware*, *data center aware*, or *rack unaware*, meaning that the policies can take into account whether the replication needs to be made within the same cluster or datacenter or not to consider the geo-location of nodes. As in Dynamo,

node membership information is based on gossip protocols.[10] Cassandra makes also use of this information diffusion mode for other tasks, such as disseminating the system control state. The local file system of each node is used for data persistence, and Cassandra makes extensive use of commit logs, which makes the system able to recover from transient failures. Each write operation is applied in memory only after it has been logged on disk so that it can be easily reproduced in case of failures. When the data in memory trespasses a specified size, it is dumped to disk. Read operations are performed in-memory first and then on disk. To speed up the process, each file includes a summary of the keys it contains so that it is possible to avoid unnecessary file scanning to search for a key.

As noted earlier, Cassandra builds on the concepts designed in Dynamo and Bigtable and puts them together to achieve a completely distributed and highly reliable storage system. The largest Cassandra deployment to our knowledge manages 100 TB of data distributed over a cluster of 150 machines.

Hadoop HBase. HBase is the distributed database that supports the storage needs of the Hadoop distributed programming platform. HBase is designed by taking inspiration from Google Bigtable; its main goal is to offer real-time read/write operations for tables with billions of rows and millions of columns by leveraging clusters of commodity hardware. The internal architecture and logic model of HBase is very similar to Google Bigtable, and the entire system is backed by the Hadoop Distributed File System (HDFS), which mimics the structure and services of GFS.

In this section, we discussed the storage solutions that support the management of data-intensive applications, especially those referred as *Big Data*. Traditionally, database systems, most likely based on the relational model, have been the primary solution for handling large quantities of data. As we discussed, when it comes to extremely huge quantities of unstructured data, relational databases become impractical and provide poor performance. Alternative and more effective solutions have significantly reviewed the fundamental concepts at the base of distributed file systems and storage systems. The next level comprises providing programming platforms that, by leveraging the discussed storage systems, can capitalize on developers' efforts to handle massive amounts of data. Among them, MapReduce and all its variations play a fundamental role.

8.2.2 Programming platforms

Platforms for programming data-intensive applications provide abstractions that help express the computation over a large quantity of information and runtime systems able to efficiently manage huge volumes of data. Traditionally, database management systems based on the relational model have been used to express the structure and connections between the entities of a data model. This approach has proven unsuccessful in the case of Big Data, where information is mostly found unstructured or semistructured and where data are most likely to be organized in files of large size or a huge number of medium-sized files rather than rows in a database. Distributed workflows have often been used to analyze and process large amounts of data [66,67]. This approach introduced a plethora of frameworks for workflow management systems, as discussed in Section 7.2.4, which

[10] A *gossip protocol* is a style of communication protocol inspired by the form of gossip seen in social networks. Gossip protocols are used in distributed systems as an alternative to distributed and propagate information that is efficient compared to flooding or other kinds of algorithms.

eventually incorporated capabilities to leverage the elastic features offered by cloud computing [70]. These systems are fundamentally based on the abstraction of a *task*, which puts a big burden on the developer, who needs to deal with data management and, often, data transfer issues.

Programming platforms for data-intensive computing provide higher-level abstractions, which focus on the processing of data and move into the runtime system the management of transfers, thus making the data always available where needed. This is the approach followed by the MapReduce [55] programming platform, which expresses the computation in the form of two simple functions—map and reduce—and hides the complexities of managing large and numerous data files into the distributed file system supporting the platform. In this section, we discuss the characteristics of MapReduce and present some variations of it, which extend its capabilities for wider purposes.

8.2.2.1 The MapReduce programming model

MapReduce [55] is a programming platform Google introduced for processing large quantities of data. It expresses the computational logic of an application in two simple functions: *map* and *reduce*. Data transfer and management are completely handled by the distributed storage infrastructure (i.e., the Google File System), which is in charge of providing access to data, replicating files, and eventually moving them where needed. Therefore, developers no longer have to handle these issues and are provided with an interface that presents data at a higher level: as a collection of key-value pairs. The computation of MapReduce applications is then organized into a workflow of *map* and *reduce* operations that is entirely controlled by the runtime system; developers need only specify how the *map* and *reduce* functions operate on the key-value pairs.

More precisely, the MapReduce model is expressed in the form of the two functions, which are defined as follows:

$$map\ (k1, v1) \rightarrow list(k2, v2)$$
$$reduce(k2, list(v2)) \rightarrow list(v2)$$

The *map* function reads a key-value pair and produces a list of key-value pairs of different types. The *reduce* function reads a pair composed of a key and a list of values and produces a list of values of the same type. The types (*k1,v1,k2,kv2*) used in the expression of the two functions provide hints as to how these two functions are connected and are executed to carry out the computation of a MapReduce job: The output of map tasks is aggregated together by grouping the values according to their corresponding keys and constitutes the input of *reduce* tasks that, for each of the keys found, reduces the list of attached values to a single value. Therefore, the input of a MapReduce computation is expressed as a collection of key-value pairs $< k1,v1 >$, and the final output is represented by a list of values: *list(v2)*.

Figure 8.5 depicts a reference workflow characterizing MapReduce computations. As shown, the user submits a collection of files that are expressed in the form of a list of $< k1,v1 >$ pairs and specifies the *map* and *reduce* functions. These files are entered into the distributed file system that supports MapReduce and, if necessary, partitioned in order to be the input of map tasks. Map tasks generate intermediate files that store collections of $< k2,\ list(v2) >$ pairs, and these files are saved into the distributed file system. The MapReduce runtime might eventually aggregate the values corresponding to the same keys. These files constitute the input of reduce tasks, which finally produce output files in the form of *list(v2)*. The operation performed by reduce tasks is generally expressed

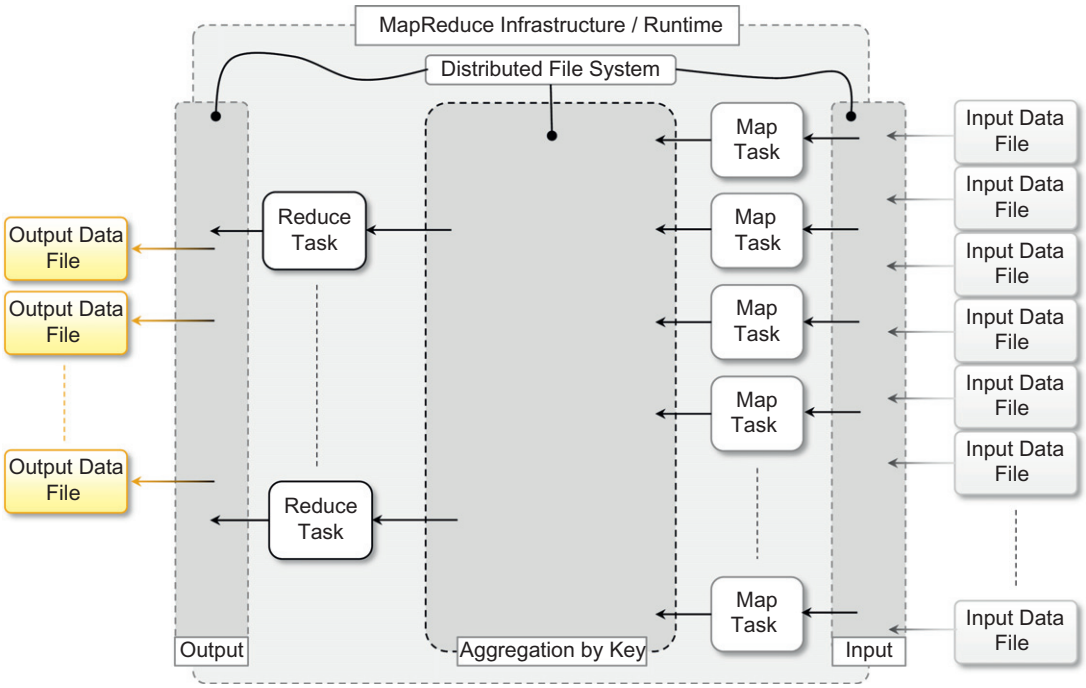

FIGURE 8.5

MapReduce computation workflow.

as an aggregation of all the values that are mapped by a specific key. The number of map and reduce tasks to create, the way files are partitioned with respect to these tasks, and the number of map tasks connected to a single reduce task are the responsibilities of the MapReduce runtime. In addition, the way files are stored and moved is the responsibility of the distributed file system that supports MapReduce.

The computation model expressed by MapReduce is very straightforward and allows greater productivity for people who have to code the algorithms for processing huge quantities of data. This model has proven successful in the case of Google, where the majority of the information that needs to be processed is stored in textual form and is represented by Web pages or log files. Some of the examples that show the flexibility of MapReduce are the following [55]:

- *Distributed grep.* The *grep* operation, which performs the recognition of patterns within text streams, is performed across a wide set of files. MapReduce is leveraged to provide a parallel and faster execution of this operation. In this case, the input file is a plain text file, and the *map* function emits a line into the output each time it recognizes the given pattern. The reduce task aggregates all the lines emitted by the map tasks into a single file.
- *Count of URL-access frequency.* MapReduce is used to distribute the execution of Web server log parsing. In this case, the *map* function takes as input the log of a Web server and emits into the output file a key-value pair $< URL, 1 >$ for each page access recorded in the log. The

reduce function aggregates all these lines by the corresponding URL, thus summing the single accesses, and outputs a $<$*URL, total-count*$>$ pair.

- *Reverse Web-link graph.* The Reverse Web-link graph keeps track of all the possible Web pages that might lead to a given link. In this case input files are simple HTML pages that are scanned by map tasks emitting $<$*target, source*$>$ pairs for each of the links found in the Web page *source*. The reduce task will collate all the pairs that have the same target into a $<$*target, list (source)*$>$ pair. The final result is given one or more files containing these mappings.
- *Term vector per host.* A term vector recaps the most important words occurring in a set of documents in the form of *list(*$<$*word, frequency*$>$*)*, where the number of occurrences of a word is taken as a measure of its importance. MapReduce is used to provide a mapping between the origin of a set of document, obtained as the host component of the URL of a document, and the corresponding term vector. In this case, the map task creates a pair $<$*host, term-vector*$>$ for each text document retrieved, and the reduce task aggregates the term vectors corresponding to documents retrieved from the same host.
- *Inverted index.* The inverted index contains information about the presence of words in documents. This information is useful to allow fast full-text searches compared to direct document scans. In this case, the map task takes as input a document, and for each document it emits a collection of $<$*word, document-id*$>$. The *reduce* function aggregates the occurrences of the same word, producing a pair $<$*word, list(document-id)*$>$.
- *Distributed sort.* In this case, MapReduce is used to parallelize the execution of a *sort* operation over a large number of records. This application mostly relies on the properties of the MapReduce runtime, which sorts and creates partitions of the intermediate files, rather than in the operations performed in the map and reduce tasks. Indeed, these are very simple: The map task extracts the key from a record and emits a $<$*key, record*$>$ pair for each record; the reduce task will simply copy through all the pairs. The actual sorting process is performed by the MapReduce runtime, which will emit and partition the key-value pair by ordering them according to the key.

The reported example are mostly concerned with text-based processing. MapReduce can also be used, with some adaptation, to solve a wider range of problems. An interesting example is its application in the field of machine learning [97], where statistical algorithms such as *Support Vector Machines (SVM)*, *Linear Regression (LR)*, *Naïve Bayes (NB)*, and *Neural Network (NN)*, are expressed in the form of *map* and *reduce* functions. Other interesting applications can be found in the field of compute-intensive applications, such as the computation of Pi with a high degree of precision. It has been reported that the Yahoo! Hadoop cluster has been used to compute the $10^{15} + 1$ bit of Pi.[11] Hadoop is an open-source implementation of the MapReduce platform.

In general, any computation that can be expressed in the form of two major stages can be represented in the terms of MapReduce computation. These stages are:

- *Analysis.* This phase operates directly on the data input file and corresponds to the operation performed by the map task. Moreover, the computation at this stage is expected to be embarrassingly parallel, since map tasks are executed without any sequencing or ordering.

[11]The full details of this computation can be found in the Yahoo! Developer Network blog in the following blog post: http://developer.yahoo.com/blogs/hadoop/posts/2009/05/hadoop_computes_the_10151st_bi/.

- *Aggregation.* This phase operates on the intermediate results and is characterized by operations that are aimed at aggregating, summing, and/or elaborating the data obtained at the previous stage to present the data in their final form. This is the task performed by the *reduce* function.

Adaptations to this model are mostly concerned with identifying the appropriate keys, creating reasonable keys when the original problem does not have such a model, and finding ways to partition the computation between *map* and *reduce* functions. Moreover, more complex algorithms can be decomposed into multiple MapReduce programs, where the output of one program constitutes the input of the following program.

The abstraction proposed by MapReduce provides developers with a very minimal interface that is strongly focused on the algorithm to implement rather than the infrastructure on which it is executed. This is a very effective approach, but at the same time it demands a lot of common tasks, which are of concern in the management of a distributed application to the MapReduce runtime, allowing the user to specify only configuration parameters to control the behavior of applications. These tasks are managing data transfer and scheduling map and reduce tasks over a distributed infrastructure. Figure 8.6 gives a more complete overview of a MapReduce infrastructure, according to the implementation proposed by Google [55].

As depicted, the user submits the execution of MapReduce jobs by using the client libraries that are in charge of submitting the input data files, registering the *map* and *reduce* functions, and returning control to the user once the job is completed. A generic distributed infrastructure (i.e., a cluster) equipped with job-scheduling capabilities and distributed storage can be used to run MapReduce applications. Two different kinds of processes are run on the distributed infrastructure: a master process and a worker process.

The master process is in charge of controlling the execution of map and reduce tasks, partitioning, and reorganizing the intermediate output produced by the map task in order to feed the reduce tasks. The worker processes are used to host the execution of map and reduce tasks and provide basic I/O facilities that are used to interface the map and reduce tasks with input and output files. In a MapReduce computation, input files are initially divided into splits (generally 16 to 64 MB) and stored in the distributed file system. The master process generates the map tasks and assigns input splits to each of them by balancing the load.

Worker processes have input and output buffers that are used to optimize the performance of map and reduce tasks. In particular, output buffers for map tasks are periodically dumped to disk to create intermediate files. Intermediate files are partitioned using a user-defined function to evenly split the output of map tasks. The locations of these pairs are then notified to the master process, which forwards this information to the reduce tasks, which are able to collect the required input via a remote procedure call in order to read from the map tasks' local storage. The key range is then sorted and all the same keys are grouped together. Finally, the reduce task is executed to produce the final output, which is stored in the global file system. This process is completely automatic; users may control it through configuration parameters that allow specifying (besides the *map* and *reduce* functions) the number of map tasks, the number of partitions into which to separate the final output, and the *partition* function for the intermediate key range.

Besides orchestrating the execution of map and reduce tasks as previously described, the MapReduce runtime ensures a reliable execution of applications by providing a fault-tolerant

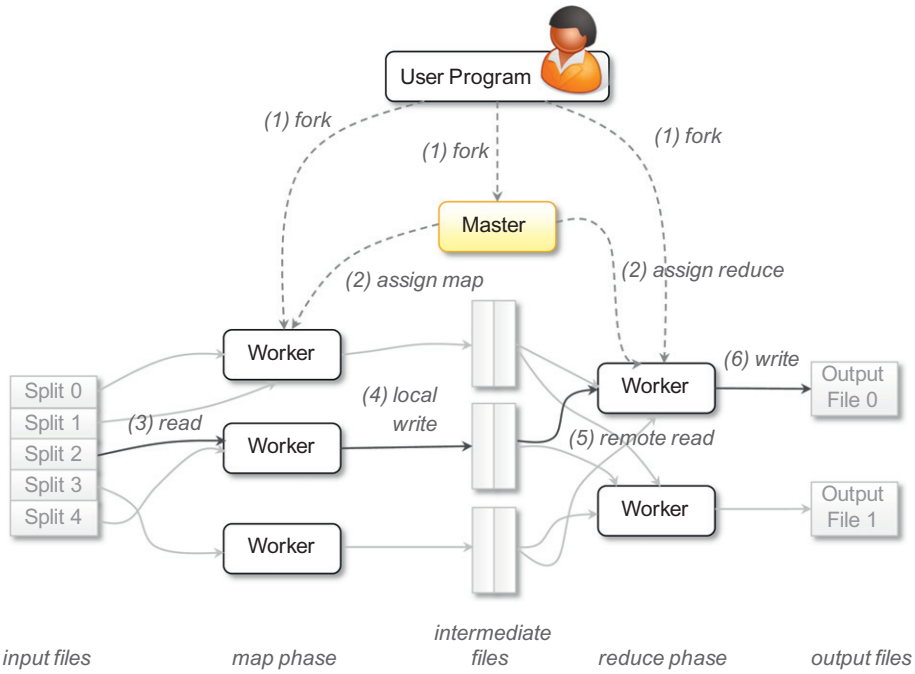

FIGURE 8.6

Google MapReduce infrastructure overview.

infrastructure. Failures of both master and worker processes are handled, as are machine failures that make intermediate outputs inaccessible. Worker failures are handled by rescheduling map tasks somewhere else. This is also the technique that is used to address machine failures since the valid intermediate output of map tasks has become inaccessible. Master process failure is instead addressed using checkpointing, which allows restarting the MapReduce job with a minimum loss of data and computation.

8.2.2.2 Variations and extensions of MapReduce

MapReduce constitutes a simplified model for processing large quantities of data and imposes constraints on the way distributed algorithms should be organized to run over a MapReduce infrastructure. Although the model can be applied to several different problem scenarios, it still exhibits limitations, mostly due to the fact that the abstractions provided to process data are very simple, and complex problems might require considerable effort to be represented in terms of *map* and *reduce* functions only. Therefore, a series of extensions to and variations of the original MapReduce model have been proposed. They aim at extending the MapReduce application space and providing developers with an easier interface for designing distributed algorithms. In this

section, we briefly present a collection of MapReduce-like frameworks and discuss how they differ from the original MapReduce model.

Hadoop. Apache Hadoop [83] is a collection of software projects for reliable and scalable distributed computing. Taken together, the entire collection is an open-source implementation of the MapReduce framework supported by a GFS-like distributed file system. The initiative consists of mostly two projects: Hadoop Distributed File System (HDFS) and Hadoop MapReduce. The former is an implementation of the Google File System [54]; the latter provides the same features and abstractions as Google MapReduce. Initially developed and supported by Yahoo!, Hadoop now constitutes the most mature and large data cloud application and has a very robust community of developers and users supporting it. Yahoo! now runs the world's largest Hadoop cluster, composed of 40,000 machines and more than 300,000 cores, made available to academic institutions all over the world. Besides the core projects of Hadoop, a collection of other projects related to it provides services for distributed computing.

Pig. Pig[12] is a platform that allows the analysis of large datasets. Developed as an Apache project, Pig consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The Pig infrastructure's layer consists of a compiler for a high-level language that produces a sequence of MapReduce jobs that can be run on top of distributed infrastructures such as Hadoop. Developers can express their data analysis programs in a textual language called *Pig Latin*, which exposes a SQL-like interface and is characterized by major expressiveness, reduced programming effort, and a familiar interface with respect to MapReduce.

Hive. Hive[13] is another Apache initiative that provides a data warehouse infrastructure on top of Hadoop MapReduce. It provides tools for easy data summarization, *ad hoc* queries, and analysis of large datasets stored in Hadoop MapReduce files. Whereas the framework provides the same capabilities as a classical data warehouse, it does not exhibit the same performance, especially in terms of query latency, and for this reason does not constitute a valid solution for online transaction processing. Hive's major advantages reside in the ability to scale out, since it is based on the Hadoop framework, and in the ability to provide a data warehouse infrastructure in environments where there is already a Hadoop system running.

Map-Reduce-Merge. Map-Reduce-Merge [98] is an extension of the MapReduce model, introducing a third phase to the standard MapReduce pipeline—the Merge phase—that allows efficiently merging data already partitioned and sorted (or hashed) by map and reduce modules. The Map-Reduce-Merge framework simplifies the management of heterogeneous related datasets and provides an abstraction able to express the common relational algebra operators as well as several join algorithms.

Twister. Twister [99] is an extension of the MapReduce model that allows the creation of iterative executions of MapReduce jobs. With respect to the normal MapReduce pipeline, the model proposed by Twister proposes the following extensions:

1. `Configure Map`
2. `Configure Reduce`

[12] http://pig.apache.org/.
[13] http://hive.apache.org/.

3. `While Condition Holds True Do`
   a. `Run MapReduce`
   b. `Apply Combine Operation to Result`
   c. `Update Condition`
4. `Close`

Besides the iterative MapReduce computation, Twister provides additional features such as the ability for *map* and *reduce* tasks to refer to static and in-memory data; the introduction of an additional phase called *combine*, run at the end of the MapReduce job, that aggregates the output together; and other tools for management of data.

8.2.2.3 Alternatives to MapReduce

MapReduce, other abstractions provide support for processing large datasets and execute data-intensive workloads. To different extents, these alternatives exhibit some similarities to the MapReduce approach.

Sphere. Sphere [84] is the distributed processing engine that leverages the Sector Distributed File System (SDFS). Rather than being a variation of MapReduce, Sphere implements the stream processing model (*Single Program, Multiple Data*) and allows developers to express the computation in terms of *user-defined functions (UDFs)*, which are run against the distributed infrastructure. A specific combination of UDFs allows Sphere to express MapReduce computations. Sphere strongly leverages the Sector distributed file systems, and it is built on top of Sector's API for data access. UDFs are expressed in terms of programs that read and write streams. A *stream* is a data structure that provides access to a collection of data segments mapping one or more files in the SDFS. The collective execution of UDFs is achieved through the distributed execution of *Sphere Process Engines (SPEs)*, which are assigned with a given stream segment. The execution model is a master-slave model that is client controlled; a Sphere client sends a request for processing to the master node, which returns the list of available slaves, and the client will choose the slaves on which to execute Sphere processes and orchestrate the entire distributed execution.

All-Pairs. All-Pairs [100] is an abstraction and a runtime environment for the optimized execution of data-intensive workloads. It provides a simple abstraction—in terms of the *All-pairs* function—that is common in many scientific computing domains:

$$All\text{-}pairs(A{:}set, B{:}set, F{:}function) \rightarrow M{:}matrix$$

Examples of problems that can be represented in this model can be found in the field of biometrics, where similarity matrices are composed as a result of the comparison of several images that contain subject pictures. Another example is several applications and algorithms in data mining. The model expressed by the *All-pairs* function can be easily solved by the following algorithm:

1. `For each $i in A`
2. `For each $j in B`
3. `Submit job F $i $j`

This implementation is quite naïve and produces poor performance in general. Moreover, other problems, such as data distribution, dispatch latency, number of available compute nodes, and probability of failure, are not handled specifically. The All-Pairs model tries to address these issues by

introducing a specification for the nature of the problem and an engine that, according to this specification, optimizes the distribution of tasks over a conventional cluster or grid infrastructure. The execution of a distributed application is controlled by the engine and develops in four stages: (1) model the system; (2) distribute the data; (3) dispatch batch jobs; and (4) clean up the system. The interesting aspect of this model is mostly concentrated on the first two phases, where the performance model of the system is built and the data are opportunistically distributed in order to create the optimal number of tasks to assign to each node and optimize the utilization of the infrastructure.

DryadLINQ. Dryad [101] is a Microsoft Research project that investigates programming models for writing parallel and distributed programs to scale from a small cluster to a large datacenter. Dryad's aim is to provide an infrastructure for automatically parallelizing the execution of applications without requiring the developer to know about distributed and parallel programming.

In Dryad, developers can express distributed applications as a set of sequential programs that are connected by means of channels. More precisely, a Dryad computation can be expressed in terms of a directed acyclic graph in which nodes are the sequential programs and vertices represent the channels connecting such programs. Because of this structure, Dryad is considered a superset of the MapReduce model, since its general application model allows expressing graphs representing MapReduce computation as well. An interesting feature exposed by Dryad is the capability of supporting dynamic modification of the graph (to some extent) and of partitioning, if possible, the execution of the graph into stages. This infrastructure is used to serve different applications and tools for parallel programming. Among them, DryadLINQ [102] is a programming environment that produces Dryad computations from the Language Integrated Query (LINQ) extensions to C# [103]. The resulting framework provides a solution that is completely integrated into the .NET framework and able to express several distributed computing models, including MapReduce.

8.3 Aneka MapReduce programming

Aneka provides an implementation of the MapReduce abstractions by following the reference model introduced by Google and implemented by Hadoop. MapReduce is supported as one of the available programming models that can be used to develop distributed applications.

8.3.1 Introducing the MapReduce programming model

The *MapReduce Programming Model* defines the abstractions and runtime support for developing MapReduce applications on top of Aneka. Figure 8.7 provides an overview of the infrastructure supporting MapReduce in Aneka. A MapReduce job in Google MapReduce or Hadoop corresponds to the execution of a MapReduce application in Aneka. The application instance is specialized, with components that identify the *map* and *reduce* functions to use. These functions are expressed in terms of *Mapper* and *Reducer* classes that are extended from the Aneka MapReduce APIs. The runtime support is composed of three main elements:

- *MapReduce Scheduling Service*, which plays the role of the master process in the Google and Hadoop implementation
- *MapReduce Execution Service*, which plays the role of the worker process in the Google and Hadoop implementation
- A specialized distributed file system that is used to move data files

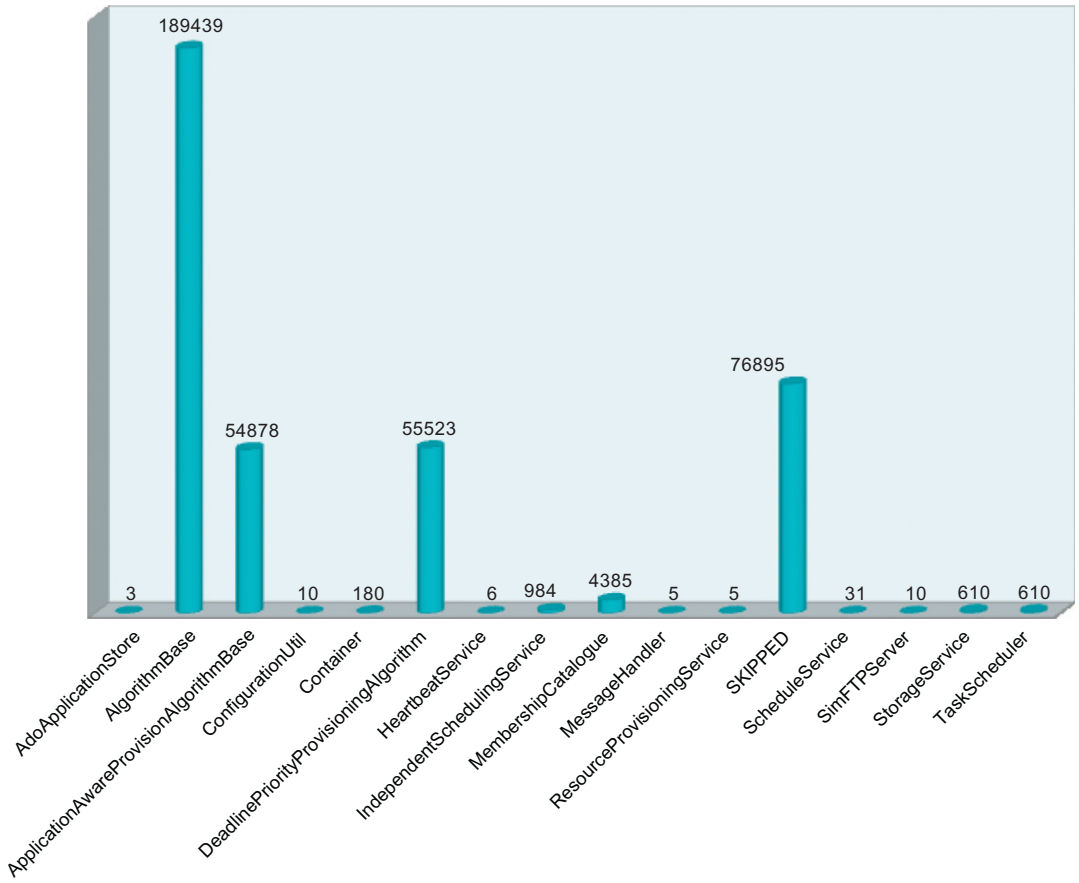

FIGURE 8.13

Component entries distribution.

and task programming models in Aneka, we discussed the programming abstractions supporting the design and implementation of MapReduce applications. We presented the structure and organization of Aneka's runtime services for the execution of MapReduce jobs. Finally, we included step-by-step examples of how to design and implement applications using Aneka MapReduce APIs.

Review questions

1. What is a data-intensive computing? Describe the characteristics that define this term.
2. Provide an historical perspective on the most important technologies that support data-intensive computing.

3. What are the characterizing features of so-called Big Data?
4. List some of the important storage technologies that support data-intensive computing and describe one of them.
5. Describe the architecture of the Google File System.
6. What does the term NoSQL mean?
7. Describe the characteristics of Amazon Simple Storage Service (S3).
8. What is Google Bigtable?
9. What are the requirements of a programming platform that supports data-intensive computations?
10. What is MapReduce?
11. Describe the kinds of problems MapReduce can solve and give some real examples.
12. List some of the variations on or extensions to MapReduce.
13. What are the major components of the Aneka MapReduce Programming Model?
14. How does the MapReduce model differ from the other models supported by Aneka and discussed in this book?
15. Describe the components of the Scheduling and Execution Services that constitute the runtime infrastructure supporting MapReduce.
16. Describe the architecture of the data storage layer designed for Aneka MapReduce and the I/O APIs for handling MapReduce files.
17. Design and implement a simple program that uses MapReduce for the computation of Pi.

CHAPTER

Cloud Platforms in Industry

9

Cloud computing allows end users and developers to leverage large distributed computing infrastructures. This is made possible thanks to infrastructure management software and distributed computing platforms offering on-demand compute, storage, and, on top of these, more advanced services. There are several different options for building enterprise cloud computing applications or for using cloud computing technologies to integrate and extend existing industrial applications. An overview of a few prominent cloud computing platforms and a brief description of the types of service they offer are shown in Table 9.1. A cloud computing system can be developed using either a single technology and vendor or a combination of them.

This chapter presents some of the representative cloud computing solutions offered as Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) services in the market. It provides some insights into and practical issues surrounding the architecture of the major cloud computing technologies and their service offerings.

9.1 Amazon web services

Amazon Web Services (AWS) is a platform that allows the development of flexible applications by providing solutions for elastic infrastructure scalability, messaging, and data storage. The platform is accessible through SOAP or RESTful Web service interfaces and provides a Web-based console where users can handle administration and monitoring of the resources required, as well as their expenses computed on a pay-as-you-go basis.

Figure 9.1 shows all the services available in the AWS ecosystem. At the base of the solution stack are services that provide raw compute and raw storage: *Amazon Elastic Compute (EC2)* and *Amazon Simple Storage Service (S3)*. These are the two most popular services, which are generally complemented with other offerings for building a complete system. At the higher level, *Elastic MapReduce* and *AutoScaling* provide additional capabilities for building smarter and more elastic computing systems. On the data side, *Elastic Block Store (EBS)*, *Amazon SimpleDB*, *Amazon RDS*, and *Amazon ElastiCache* provide solutions for reliable data snapshots and the management of structured and semistructured data. Communication needs are covered at the networking level by *Amazon Virtual Private Cloud (VPC)*, *Elastic Load Balancing*, *Amazon Route 53*, and *Amazon Direct Connect*. More advanced services for connecting applications are *Amazon Simple Queue*

Table 9.1 Some Example Cloud Computing Offerings

Vendor/Product	Service Type	Description
Amazon Web Services	IaaS, PaaS, SaaS	Amazon Web Services (AWS) is a collection of Web services that provides developers with compute, storage, and more advanced services. AWS is mostly popular for IaaS services and primarily for its elastic compute service EC2.
Google AppEngine	PaaS	Google AppEngine is a distributed and scalable runtime for developing scalable Web applications based on Java and Python runtime environments. These are enriched with access to services that simplify the development of applications in a scalable manner.
Microsoft Azure	PaaS	Microsoft Azure is a cloud operating system that provides services for developing scalable applications based on the proprietary Hyper-V virtualization technology and the .NET framework.
SalesForce.com and Force.com	SaaS, PaaS	SalesForce.com is a Software-as-a-Service solution that allows prototyping of CRM applications. It leverages the Force.com platform, which is made available for developing new components and capabilities for CRM applications.
Heroku	PaaS	Heroku is a scalable runtime environment for building applications based on Ruby.
RightScale	IaaS	Rightscale is a cloud management platform with a single dashboard to manage public and hybrid clouds.

Service (SQS), *Amazon Simple Notification Service (SNS)*, and *Amazon Simple E-mail Service (SES)*. Other services include:

- *Amazon CloudFront* content delivery network solution
- *Amazon CloudWatch* monitoring solution for several Amazon services
- *Amazon Elastic BeanStalk* and *CloudFormation* flexible application packaging and deployment

As shown, AWS comprise a wide set of services. We discuss the most important services by examining the solutions proposed by AWS regarding compute, storage, communication, and complementary services.

9.1.1 Compute services

Compute services constitute the fundamental element of cloud computing systems. The fundamental service in this space is Amazon EC2, which delivers an IaaS solution that has served as a reference model for several offerings from other vendors in the same market segment. Amazon EC2 allows deploying servers in the form of virtual machines created as instances of a specific image. Images come with a preinstalled operating system and a software stack, and instances can be configured for memory, number of processors, and storage. Users are provided with credentials to remotely access the instance and further configure or install software if needed.

9.1.1.1 Amazon machine images

Amazon Machine Images (AMIs) are templates from which it is possible to create a virtual machine. They are stored in Amazon S3 and identified by a unique identifier in the form of *ami-xxxxx* and

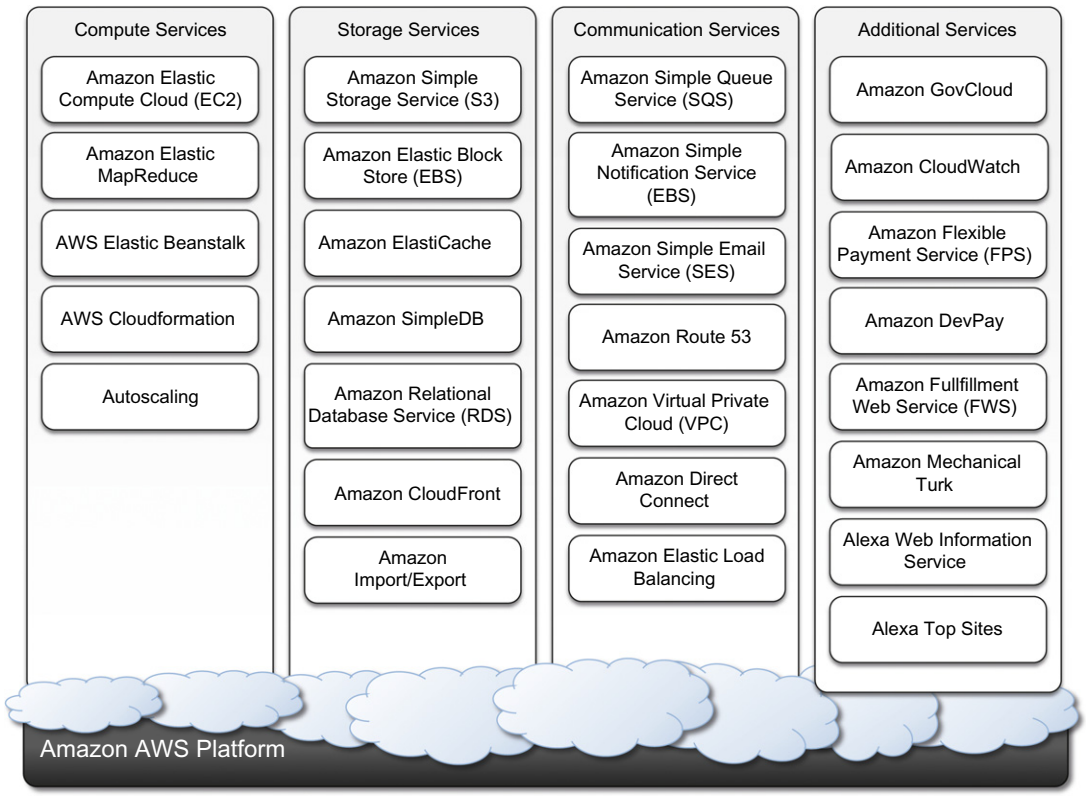

FIGURE 9.1

Amazon Web Services ecosystem.

a manifest XML file. An AMI contains a physical file system layout with a predefined operating system installed. These are specified by the *Amazon Ramdisk Image* (*ARI*, id: *ari-yyyyyy*) and the *Amazon Kernel Image* (*AKI*, id: *aki-zzzzzz*), which are part of the configuration of the template. AMIs are either created from scratch or "bundled" from existing EC2 instances. A common practice is to prepare new AMIs to create an instance from a preexisting AMI, log into it once it is booted and running, and install all the software needed. Using the tools provided by Amazon, we can convert the instance into a new image. Once an AMI is created, it is stored in an S3 bucket and the user can decide whether to make it available to other users or keep it for personal use. Finally, it is also possible to associate a product code with a given AMI, thus allowing the owner of the AMI to get revenue every time this AMI is used to create EC2 instances.

9.1.1.2 EC2 instances

EC2 instances represent virtual machines. They are created using AMI as templates, which are specialized by selecting the number of cores, their computing power, and the installed memory. The processing power is expressed in terms of virtual cores and EC2 Compute Units (ECUs). The ECU

is a measure of the computing power of a virtual core; it is used to express a predictable quantity of real CPU power that is allocated to an instance. By using compute units instead of real frequency values, Amazon can change over time the mapping of such units to the underlying real amount of computing power allocated, thus keeping the performance of EC2 instances consistent with standards set by the times. Over time, the hardware supporting the underlying infrastructure will be replaced by more powerful hardware, and the use of ECUs helps give users a consistent view of the performance offered by EC2 instances. Since users rent computing capacity rather than buying hardware, this approach is reasonable. One ECU is defined as giving the same performance as a 1.0−1.2 GHz 2007 Opteron or 2007 Xeon processor.[1]

Table 9.2 shows all the currently available configurations for EC2 instances. We can identify six major categories:

- *Standard instances*. This class offers a set of configurations that are suitable for most applications. EC2 provides three different categories of increasing computing power, storage, and memory.
- *Micro instances*. This class is suitable for those applications that consume a limited amount of computing power and memory and occasionally need bursts in CPU cycles to process surges in the workload. Micro instances can be used for small Web applications with limited traffic.
- *High-memory instances*. This class targets applications that need to process huge workloads and require large amounts of memory. Three-tier Web applications characterized by high traffic are the target profile. Three categories of increasing memory and CPU are available, with memory proportionally larger than computing power.
- *High-CPU instances.* This class targets compute-intensive applications. Two configurations are available where computing power proportionally increases more than memory.
- *Cluster Compute instances*. This class is used to provide virtual cluster services. Instances in this category are characterized by high CPU compute power and large memory and an extremely high I/O and network performance, which makes it suitable for HPC applications.
- *Cluster GPU instances*. This class provides instances featuring graphic processing units (GPUs) and high compute power, large memory, and extremely high I/O and network performance. This class is particularly suited for cluster applications that perform heavy graphic computations, such as rendering clusters. Since GPU can be used for general-purpose computing, users of such instances can benefit from additional computing power, which makes this class suitable for HPC applications.

EC2 instances are priced hourly according to the category they belong to. At the beginning of every hour of usage, the user will be charged the cost of the entire hour. The hourly expense charged for one instance is constant. Instance owners are responsible for providing their own backup strategies, since there is no guarantee that the instance will run for the entire hour. Another alternative is represented by *spot instances*. These instances are much more dynamic in terms of pricing and lifetime since they are made available to the user according to the load of EC2 and the availability of resources. Users define an upper bound for a price they want to pay for these instances; as long as the current price (the spot price) remains under the given bound, the instance is kept running. The price is sampled at the beginning of each hour. Spot instances are more volatile than normal instances; whereas for normal instances EC2 will try as much as possible to keep

[1] http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it.

Table 9.2 Amazon EC2 (On-Demand) Instances Characteristics

Instance Type	ECU	Platform	Memory	Disk Storage	Price (U.S. East) (USD/hour)
Standard instances					
Small	1(1 × 1)	32 bit	1.7 GB	160 GB	$0.085 Linux $0.12 Windows
Large	4(2 × 2)	64 bit	7.5 GB	850 GB	$0.340 Linux $0.48 Windows
Extra Large	8(4 × 2)	64 bit	15 GB	1,690 GB	$0.680 Linux $0.96 Windows
Micro instances					
Micro	< = 2	32/64 bit	613 MB	EBS Only	$0.020 Linux $0.03 Windows
High-Memory instances					
Extra Large	6.5(2 × 3.25)	64 bit	17.1 GB	420 GB	$0.500 Linux $0.62 Windows
Double Extra Large	13(4 × 3.25)	64 bit	34.2 GB	850 GB	$1.000 Linux $1.24 Windows
Quadruple Extra Large	26(8 × 3.25)	64 bit	68.4 GB	1,690 GB	$2.000 Linux $2.48 Windows
High-CPU instances					
Medium	5(2 × 2.5)	32 bit	1.7 GB	350 GB	$0.170 Linux $0.29 Windows
Extra Large	20(8 × 2.5)	64 bit	7 GB	1,690 GB	$0.680 Linux $1.16 Windows
Cluster instances					
Quadruple Extra Large	33.5	64 bit	23 GB	1,690 GB	$1.600 Linux $1.98 Windows
Cluster GPU instances					
Quadruple Extra Large	33.5	64 bit	22 GB	1,690 GB	$2.100 Linux $2.60 Windows

them active, there is no such guarantee for spot instances. Therefore, implementing backup and checkpointing strategies is inevitable.

EC2 instances can be run either by using the command-line tools provided by Amazon, which connects the Amazon Web Service that provides remote access to the EC2 infrastructure, or via the AWS console, which allows the management of other services, such as S3. By default an EC2 instance is created with the kernel and the disk associated to the AMI. These define the architecture (32 bit or 64 bit) and the space of disk available to the instance. This is an ephemeral disk; once the instance is shut down, the content of the disk will be lost. Alternatively, it is possible to attach an EBS volume to the instance, the content of which will be stored in S3. If the default AKI and ARI are not suitable, EC2 provides capabilities to run EC2 instances by specifying a different AKI and ARI, thus giving flexibility in the creation of instances.

9.1.1.3 EC2 environment

EC2 instances are executed within a virtual environment, which provides them with the services they require to host applications. The EC2 environment is in charge of allocating addresses, attaching storage volumes, and configuring security in terms of access control and network connectivity.

By default, instances are created with an internal IP address, which makes them capable of communicating within the EC2 network and accessing the Internet as clients. It is possible to associate an *Elastic IP* to each instance, which can then be remapped to a different instance over time. Elastic IPs allow instances running in EC2 to act as servers reachable from the Internet and, since they are not strictly bound to specific instances, to implement failover capabilities. Together with an external IP, EC2 instances are also given a domain name that generally is in the form *ec2-xxx-xxx-xxx.compute-x.amazonaws.com*, where *xxx-xxx-xxx* normally represents the four parts of the external IP address separated by a dash, and *compute-x* gives information about the availability zone where instances are deployed. Currently, there are five availability zones that are priced differently: two in the United States (Virginia and Northern California), one in Europe (Ireland), and two in Asia Pacific (Singapore and Tokyo).

Instance owners can partially control where to deploy instances. Instead, they have a finer control over the security of the instances as well as their network accessibility. Instance owners can associate a key pair to one or more instances when these instances are created. A key pair allows the owner to remotely connect to the instance once this is running and gain root access to it. Amazon EC2 controls the accessibility of a virtual instance with basic firewall configuration, allowing the specification of source address, port, and protocols (TCP, UDP, ICMP). Rules can also be attached to security groups, and instances can be made part of one or more groups before their deployment. Security groups and firewall rules constitute a flexible way of providing basic security for EC2 instances, which has to be complemented by appropriate security configuration within the instance itself.

9.1.1.4 Advanced compute services

EC2 instances and AMIs constitute the basic blocks for building an IaaS computing cloud. On top of these, Amazon Web Services provide more sophisticated services that allow the easy packaging and deploying of applications and a computing platform that supports the execution of MapReduce-based applications.

AWS CloudFormation

AWS CloudFormation constitutes an extension of the simple deployment model that characterizes EC2 instances. CloudFormation introduces the concepts of *templates*, which are JSON formatted text files that describe the resources needed to run an application or a service in EC2 together with the relations between them. CloudFormation allows easily and explicitly linking EC2 instances together and introducing dependencies among them. Templates provide a simple and declarative way to build complex systems and integrate EC2 instances with other AWS services such as S3, SimpleDB, SQS, SNS, Route 53, Elastic Beanstalk, and others.

AWS elastic beanstalk

AWS Elastic Beanstalk constitutes a simple and easy way to package applications and deploy them on the AWS Cloud. This service simplifies the process of provisioning instances and deploying

application code and provides appropriate access to them. Currently, this service is available only for Web applications developed with the Java/Tomcat technology stack. Developers can conveniently package their Web application into a WAR file and use Beanstalk to automate its deployment on the AWS Cloud.

With respect to other solutions that automate cloud deployment, Beanstalk simplifies tedious tasks without removing the user's capability of accessing—and taking over control of—the underlying EC2 instances that make up the virtual infrastructure on top of which the application is running. With respect to AWS CloudFormation, AWS Elastic Beanstalk provides a higher-level approach for application deployment on the cloud, which does not require the user to specify the infrastructure in terms of EC2 instances and their dependencies.

Amazon elastic MapReduce

Amazon Elastic MapReduce provides AWS users with a cloud computing platform for MapReduce applications. It utilizes Hadoop as the MapReduce engine, deployed on a virtual infrastructure composed of EC2 instances, and uses Amazon S3 for storage needs.

Apart from supporting all the application stack connected to Hadoop (Pig, Hive, etc.), Elastic MapReduce introduces elasticity and allows users to dynamically size the Hadoop cluster according to their needs, as well as select the appropriate configuration of EC2 instances to compose the cluster (Small, High-Memory, High-CPU, Cluster Compute, and Cluster GPU). On top of these services, basic Web applications allowing users to quickly run data-intensive applications without writing code are offered.

9.1.2 Storage services

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

9.1.2.1 S3 key concepts

As the name suggests, S3 has been designed to provide a simple storage service that's accessible through a Representational State Transfer (REST) interface, which is quite similar to a distributed file system but which presents some important differences that allow the infrastructure to be highly efficient:

- *The storage is organized in a two-level hierarchy*. S3 organizes its storage space into buckets that cannot be further partitioned. This means that it is not possible to create directories or other kinds of physical groupings for objects stored in a bucket. Despite this fact, there are few limitations in naming objects, and this allows users to simulate directories and create logical groupings.
- *Stored objects cannot be manipulated like standard files*. S3 has been designed to essentially provide storage for objects that will not change over time. Therefore, it does not allow renaming, modifying, or relocating an object. Once an object has been added to a bucket, its

content and position is immutable, and the only way to change it is to remove the object from the store and add it again.

- *Content is not immediately available to users.* The main design goal of S3 is to provide an eventually consistent data store. As a result, because it is a large distributed storage facility, changes are not immediately reflected. For instance, S3 uses replication to provide redundancy and efficiently serve objects across the globe; this practice introduces latencies when adding objects to the store—especially large ones—which are not available instantly across the entire globe.
- *Requests will occasionally fail.* Due to the large distributed infrastructure being managed, requests for object may occasionally fail. Under certain conditions, S3 can decide to drop a request by returning an internal server error. Therefore, it is expected to have a small failure rate during day-to-day operations, which is generally not identified as a persistent failure.

Access to S3 is provided with RESTful Web services. These express all the operations that can be performed on the storage in the form of HTTP requests (*GET, PUT, DELETE, HEAD*, and *POST*), which operate differently according to the element they address. As a rule of thumb *PUT/POST* requests add new content to the store, *GET/HEAD* requests are used to retrieve content and information, and *DELETE* requests are used to remove elements or information attached to them.

Resource naming

Buckets, objects, and attached metadata are made accessible through a REST interface. Therefore, they are represented by *uniform resource identifiers (URIs)* under the *s3.amazonaws.com* domain. All the operations are then performed by expressing the entity they are directed to in the form of a request for a URI.

Amazon offers three different ways of addressing a bucket:

- *Canonical form: http://s3.amazonaws.com/bukect_name/.* The bucket name is expressed as a path component of the domain name s3.amazonaws.com. This is the naming convention that has less restriction in terms of allowed characters, since all the characters that are allowed for a path component can be used.
- *Subdomain form: http://bucketname.s3.amazon.com/.* Alternatively, it is also possible to reference a bucket as a subdomain of s3.amazonaws.com. To express a bucket name in this form, the name has to do all of the following:
  - Be between 3 and 63 characters long
  - Contain only letters, numbers, periods, and dashes
  - Start with a letter or a number
  - Contain at least one letter
  - Have no fragments between periods that start with a dash or end with a dash or that are empty strings

  This form is equivalent to the previous one when it can be used, but it is the one to be preferred since it works more effectively for all the geographical locations serving resources stored in S3.
- *Virtual hosting form: http://bucket-name.com/.* Amazon also allows referencing of its resources with custom URLs. This is accomplished by entering a CNAME record into the DNS that points to the subdomain form of the bucket URI.

Since S3 is logically organized as a flat data store, all the buckets are managed under the s3.amazonaws.com domain. Therefore, the names of buckets must be unique across all the users.

Objects are always referred as resources local to a given bucket. Therefore, they always appear as a part of the resource component of a URI. Since a bucket can be expressed in three different ways, objects indirectly inherit this flexibility:

- Canonical form: http://s3.amazonaws.com/bukect_name/object_name
- Subdomain form: http://bucket-name/s3.amzonaws.com/object_name
- Virtual hosting form: http://bucket-name.com/object_name

Except for the *?*, which separates the resource path of a URI from the set of parameters passed with the request, all the characters that follow the */* after the bucket reference constitute the name of the object. For instance, path separator characters expressed as part of the object name do not have corresponding physical layout within the bucket store. Despite this fact, they can still be used to create logical groupings that look like directories.

Finally, specific information about a given object, such as its access control policy or the server logging settings defined for a bucket, can be referenced using a specific parameter. More precisely:

- Object ACL: http://s3.amazonaws.com/bukect_name/object_name?acl
- Bucket server logging: http://s3.amzonaws.com/bucket_name?logging

Object metadata are not directly accessible through a specific URI, but they are manipulated by adding attributes in the request of the URL and are not part of the identifier.

Buckets

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

A bucket is located in a specific geographic location and eventually replicated for fault tolerance and better content distribution. Users can select the location at which to create buckets, which by default are created in Amazon's U.S. datacenters. Once a bucket is created, all the objects that belong to the bucket will be stored in the same availability zone of the bucket. Users create a bucket by sending a *PUT* request to http://s3.amazonaws.com/ with the name of the bucket and, if they want to specify the availability zone, additional information about the preferred location. The content of a bucket can be listed by sending a *GET* request specifying the name of the bucket. Once created, the bucket cannot be renamed or relocated. If it is necessary to do so, the bucket needs to be deleted and recreated. The deletion of a bucket is performed by a *DELETE* request, which can be successful if and only if the bucket is empty.

Objects and metadata

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

characters normally used as path separators are allowed. This actually compensates for the lack of a structured file system, since directories can be emulated by properly naming objects.

Users create an object via a *PUT* request that specifies the name of the object together with the bucket name, its contents, and additional properties. The maximum size of an object is 5 GB. Once an object is created, it cannot be modified, renamed, or moved into another bucket. It is possible to retrieve an object via a *GET* request; deleting an object is performed via a *DELETE* request.

Objects can be tagged with metadata, which are passed as properties of the *PUT* request. Such properties are retrieved either with a *GET* request or with a *HEAD* request, which only returns the object's metadata without the content. Metadata are both system and user defined: the first ones are used by S3 to control the interaction with the object, whereas the second ones are meaningful to the user, who can store up to 2 KB per metadata property represented by a key-value pair of strings.

Access control and security

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

- *READ* allows the grantee to retrieve an object and its metadata and to list the content of a bucket as well as getting its metadata.
- *WRITE* allows the grantee to add an object to a bucket as well as modify and remove it.
- *READ_ACP* allows the grantee to read the ACP of a resource.
- *WRITE_ACP* allows the grantee to modify the ACP of a resource.
- *FULL_CONTROL* grants all of the preceding permissions.

Grantees can be either single users or groups. Users can be identified by their canonical IDs or the email addresses they provided when they signed up for S3. For groups, only three options are available: all users, authenticated users, and log delivery users.[2]

Once a resource is created, S3 attaches a default ACP granting full control permissions to its owner only. Changes to the ACP can be made by using the request to the resource URI followed by *?acl*. A *GET* method allows retrieval of the ACP; a *PUT* method allows uploading of a new ACP to replace the existing one. Alternatively, it is possible to use a predefined set of permissions called *canned policies* to set the ACP at the time a resource is created. These policies represent the most common access patterns for S3 resources.

ACPs provide a set of powerful rules to control S3 users' access to resources, but they do not exhibit fine grain in the case of nonauthenticated users, who cannot be differentiated and are considered as a group. To provide a finer grain in this scenario, S3 allows defining *signed URIs*, which grant access to a resource for a limited amount of time to all the requests that can provide a temporary access token.

[2]This group identifies a specific group of accounts that automated processes use to perform bucket access logging.

Advanced features

Besides the management of buckets, objects, and ACPs, S3 offers other additional features that can be helpful. These features are server access logging and integration with the *BitTorrent* file-sharing network.

Server access logging allows bucket owners to obtain detailed information about the request made for the bucket and all the objects it contains. By default, this feature is turned off; it can be activated by issuing a *PUT* request to the bucket URI followed by *?logging*. The request should include an XML file specifying the target bucket in which to save the logging files and the file name prefix. A *GET* request to the same URI allows the user to retrieve the existing logging configuration for the bucket.

The second feature of interest is represented by the capability of exposing S3 objects to the *BitTorrent* network, thus allowing files stored in S3 to be downloaded using the *BitTorrent* protocol. This is done by appending *?torrent* to the URI of the S3 object. To actually download the object, its ACP must grant read permission to everyone.

9.1.2.2 Amazon elastic block store

The Amazon Elastic Block Store (EBS) allows AWS users to provide EC2 instances with persistent storage in the form of volumes that can be mounted at instance startup. They accommodate up to 1 TB of space and are accessed through a block device interface, thus allowing users to format them according to the needs of the instance they are connected to (raw storage, file system, or other). The content of an EBS volume survives the instance life cycle and is persisted into S3. EBS volumes can be cloned, used as boot partitions, and constitute durable storage since they rely on S3 and it is possible to take incremental snapshots of their content.

EBS volumes normally reside within the same availability zone of the EC2 instances that will use them to maximize the I/O performance. It is also possible to connect volumes located in different availability zones. Once mounted as volumes, their content is lazily loaded in the background and according to the request made by the operating system. This reduces the number of I/O requests that go to the network. Volume images cannot be shared among instances, but multiple (separate) active volumes can be created from them. In addition, it is possible to attach multiple volumes to a single instance or create a volume from a given snapshot and modify its size, if the formatted file system allows such an operation.

The expense related to a volume comprises the cost generated by the amount of storage occupied in S3 and by the number of I/O requests performed against the volume. Currently, Amazon charges $0.10/GB/month of allocated storage and $0.10 per 1 million requests made to the volume.

9.1.2.3 Amazon ElastiCache

ElastiCache is an implementation of an elastic in-memory cache based on a cluster of EC2 instances. It provides fast data access from other EC2 instances through a Memcached-compatible protocol so that existing applications based on such technology do not need to be modified and can transparently migrate to ElastiCache.

ElastiCache is based on a cluster of EC2 instances running the caching software, which is made available through Web services. An ElastiCache cluster can be dynamically resized according to the demand of the client applications. Furthermore, automatic patch management and failure

detection and recovery of cache nodes allow the cache cluster to keep running without administrative intervention from AWS users, who have only to elastically size the cluster when needed.

ElastiCache nodes are priced according to the EC2 costing model, with a small price difference due to the use of the caching service installed on such instances. It is possible to choose between different types of instances; Table 9.3 provides an overview of the pricing options.

The prices indicated in Table 9.3 are related to the Amazon offerings during 2011–2012, and the amount of memory specified represents the memory available after taking system software overhead into account.

9.1.2.4 Structured storage solutions

Enterprise applications quite often rely on databases to store data in a structured form, index, and perform analytics against it. Traditionally, RDBMS have been the common data back-end for a wide range of applications, even though recently more scalable and lightweight solutions have been proposed. Amazon provides applications with structured storage services in three different forms: preconfigured EC2 AMIs, *Amazon Relational Data Storage (RDS)*, and *Amazon SimpleDB*.

Preconfigured EC2 AMIs

Preconfigured EC2 AMIs are predefined templates featuring an installation of a given database management system. EC2 instances created from these AMIs can be completed with an EBS volume for storage persistence. Available AMIs include installations of IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase, and Vertica. Instances are priced hourly according to the EC2 cost model. This solution poses most of the administrative burden on the EC2 user, who has to configure, maintain, and manage the relational database, but offers the greatest variety of products to choose from.

Amazon RDS

RDS is relational database service that relies on the EC2 infrastructure and is managed by Amazon. Developers do not have to worry about configuring the storage for high availability, designing

Table 9.3 Amazon EC2 (On-Demand) Cache Instances Characteristics, 2011–2012

Instance Type	ECU	Platform	Memory	I/O Capacity	Price (U.S. East) (USD/hour)
Standard instances					
Small	1(1 × 1)	64 bit	1.3 GB	Moderate	$0.095
Large	4(2 × 2)	64 bit	7.1 GB	High	$0.380
Extra Large	8(4 × 2)	64 bit	14.6 GB	High	$0.760
High-Memory instances					
Extra Large	6.5(2 × 3.25)	64 bit	16.7 GB	High	$0.560
Double Extra Large	13(4 × 3.25)	64 bit	33.8 GB	High	$1.120
Quadruple Extra Large	26(8 × 3.25)	64 bit	68 GB	High	$2.240
High-CPU instances					
Extra Large	26(8 × 3.25)	64 bit	6.6 GB	High	$0.760

failover strategies, or keeping the servers up-to-date with patches. Moreover, the service provides users with automatic backups, snapshots, point-in-time recoveries, and facilities for implementing replications. These and the common database management services are available through the AWS console or a specific Web service. Two relational engines are available: MySQL and Oracle.

Two key advanced features of RDS are *multi-AZ deployment* and *read replicas*. The first option provides users with a failover infrastructure for their RDBMS solutions. The high-availability solution is implemented by keeping in standby synchronized copies of the services in different availability zones that are activated if the primary service goes down. The second option provides users with increased performance for applications that are heavily based on database reads. In this case, Amazon deploys copies of the primary service that are only available for database reads, thus cutting down the response time of the service.

The available options and the relative pricing of the service during 2011–2012 are shown in Table 9.4. The table shows the costing details of the on-demand instances. There is also the possibility of using reserved instances for long terms (one to three years) by paying up-front at discounted hourly rates.

With respect to the previous solution, users are not responsible for managing, configuring, and patching the database management software, but these operations are performed by the AWS. In addition, support for elastic management of servers is simplified. Therefore, this solution is optimal for applications based on the Oracle and MySQL engines, which are migrated on the AWS infrastructure and require a scalable database solution.

Amazon SimpleDB

Amazon SimpleDB is a lightweight, highly scalable, and flexible data storage solution for applications that do not require a fully relational model for their data. SimpleDB provides support for semistructured data, the model for which is based on the concept of *domains*, *items*, and *attributes*. With respect to the relational model, this model provides fewer constraints on the structure of data entries, thus obtaining improved performance in querying large quantities of data. As happens for Amazon RDS, this service frees AWS users from performing configuration, management, and high-availability design for their data stores.

Table 9.4 Amazon RDS (On-Demand) Instances Characteristics, 2011–2012

Instance Type	ECU	Platform	Memory	I/O Capacity	Price (U.S. East) (USD/hour)
Standard instances					
Small	1(1 × 1)	64 bit	1.7 GB	Moderate	$0.11
Large	4(2 × 2)	64 bit	7.5 GB	High	$0.44
Extra Large	8(4 × 2)	64 bit	15 GB	High	$0.88
High-Memory instances					
Extra Large	6.5(2 × 3.25)	64 bit	17.1 GB	High	$0.65
Double Extra Large	13(4 × 3.25)	64 bit	34 GB	High	$1.30
Quadruple Extra Large	26(8 × 3.25)	64 bit	68 GB	High	$2.60

SimpleDB uses *domains* as top-level elements to organize a data store. These domains are roughly comparable to tables in the relational model. Unlike tables, they allow items not to have all the same column structure; each item is therefore represented as a collection of attributes expressed in the form of a key-value pair. Each domain can grow up to 10 GB of data, and by default a single user can allocate a maximum of 250 domains. Clients can create, delete, modify, and make snapshots of domains. They can insert, modify, delete, and query items and attributes. Batch insertion and deletion are also supported. The capability of querying data is one of the most relevant functions of the model, and the *select* clause supports the following test operators: $=$, $!=$, $<$, $>$, $<=$, $>=$, *like*, *not like*, *between*, *is null*, *is not null*, and *every()*. Here is a simple example on how to query data:

```
select * from domain_name where every(attribute_name) = 'value'
```

Moreover, the *select* operator can extend its query beyond the boundaries of a single domain, thus allowing users to query effectively a large amount of data.

To efficiently provide AWS users with a scalable and fault-tolerant service, SimpleDB implements a relaxed constraint model, which leads to *eventually consistent* data. The adverb *eventually* denotes the fact that multiple accesses on the same data might not read the same value in the very short term, but they will eventually converge over time. This is because SimpleDB does not lock all the copies of the data during an update, which is propagated in the background. Therefore, there is a transient period of time in which different clients can access different copies of the same data that have different values. This approach is very scalable with minor drawbacks, and it is also reasonable, since the application scenario for SimpleDB is mostly characterized by querying and indexing operations on data. Alternatively, it is possible to change the default behavior and ensure that all the readers are blocked during an update.

Even though SimpleDB is not a transactional model, it allows clients to express conditional insertions or deletions, which are useful to prevent lost updates in multiple-writer scenarios. In this case, the operation is executed if and only if the condition is verified. This condition can be used to check preexisting values of attributes for an item.

Table 9.5 provides an overview of the pricing options for the SimpleDB service for data transfer during 2011–2012. The service charges either for data transfer or stored data. Data transfer within the AWS network is not charged. In addition, SimpleDB also charges users for machine usage. The first 25 SimpleDB instances per month are free; after this threshold there is an hourly charge ($0.140 hour in the U.S. East region).

If we compare this cost model with the one characterizing S3, it becomes evident that S3 is a cheaper option for storing large objects. This is useful information for clarifying the different nature of SimpleDB with respect to S3: The former has been designed to provide fast access to semistructured collections of small objects and not for being a long-term storage option for large objects.

9.1.2.5 Amazon CloudFront

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

Table 9.5 Amazon SimpleDB Data Transfer Charges, 2011–2012

Instance Type	Price (U.S. East) (USD)
Data Transfer In	
All data transfer in	$0.000
Data Transfer Out	
1st GB/month	$0.000
Up to 10 TB/month	$0.120
Next 40 TB/month	$0.090
Next 100 TB/month	$0.070
Next 350 TB/month	$0.050
Next 524 TB/month	Special arrangements
Next 4 PB/month	Special arrangements
Greater than 5 PB/month	Special arrangements

AWS provides users with simple Web service APIs to manage CloudFront. To make available content through CloudFront, it is necessary to create a distribution. This identifies an origin server, which contains the original version of the content being distributed, and it is referenced by a DNS domain under the *Cloudfront.net* domain name (i.e., my-distribution.Cloudfront.net). It is also possible to map a given domain name to a distribution. Once the distribution is created, it is sufficient to reference the distribution name, and the CloudFront engine will redirect the request to the closest replica and eventually download the original version from the origin server if the content is not found or expired on the selected edge server.

The content that can be delivered through CloudFront is static (HTTP and HTTPS) or streaming (Real Time Messaging Protocol, or RMTP). The origin server hosting the original copy of the distributed content can be an S3 bucket, an EC2 instance, or a server external to the Amazon network. Users can restrict access to the distribution to only one or a few of the available protocols, or they can set up access rules for finer control. It is also possible to invalidate content to remove it from the distribution or force its update before expiration.

Table 9.6 provides a breakdown of the pricing during 2011–2012. Note that CloudFront is cheaper than S3. This reflects its different purpose: CloudFront is designed to optimize the distribution of very popular content that is frequently downloaded, potentially from the entire globe and not only the Amazon network.

9.1.3 Communication services

Amazon provides facilities to structure and facilitate the communication among existing applications and services residing within the AWS infrastructure. These facilities can be organized into two major categories: *virtual networking* and *messaging*.

9.1.3.1 Virtual networking

Virtual networking comprises a collection of services that allow AWS users to control the connectivity to and between compute and storage services. *Amazon Virtual Private Cloud (VPC)* and

Table 9.6 Amazon CloudFront On-Demand Pricing, 2011–2012

Pricing Item	United States	Europe	Hong Kong and Singapore	Japan	South America
Requests					
Per 10,000 HTTP requests	$0.0075	$0.0090	$0.0090	$0.0095	$0.0160
Per 10,000 HTTPS requests	$0.0100	$0.0120	$0.0120	$0.0130	$0.0220
Regional Data Transfer Out					
First 10 TB/month	$0.120/GB	$0.120/GB	$0.190/GB	$0.201/GB	$0.250/GB
Next 40 TB/month	$0.080/GB	$0.080/GB	$0.140/GB	$0.148/GB	$0.200/GB
Next 100 TB/month	$0.060/GB	$0.060/GB	$0.120/GB	$0.127/GB	$0.180/GB
Next 350 TB/month	$0.040/GB	$0.040/GB	$0.100/GB	$0.106/GB	$0.160/GB
Next 524 TB/month	$0.030/GB	$0.030/GB	$0.080/GB	$0.085/GB	$0.140/GB
Next 4 PB/month	$0.025/GB	$0.025/GB	$0.070/GB	$0.075/GB	$0.130/GB
Greater than 5 PB/month	$0.020/GB	$0.020/GB	$0.060/GB	$0.065/GB	$0.125/GB

Amazon Direct Connect provide connectivity solutions in terms of infrastructure; *Route 53* facilitates connectivity in terms of naming.

Amazon VPC provides a great degree of flexibility in creating virtual private networks within the Amazon infrastructure and beyond. The service providers prepare either templates covering most of the usual scenarios or a fully customizable network service for advanced configurations. Prepared templates include public subnets, isolated networks, private networks accessing Internet through network address translation (NAT), and hybrid networks including AWS resources and private resources. Also, it is possible to control connectivity between different services (EC2 instances and S3 buckets) by using the *Identity Access Management (IAM)* service. During 2011, the cost of Amazon VPC was $0.50 per connection hour.

Amazon Direct Connect allows AWS users to create dedicated networks between the user private network and Amazon Direct Connect locations, called *ports*. This connection can be further partitioned in multiple logical connections and give access to the public resources hosted on the Amazon infrastructure. The advantage of using Direct Connect versus other solutions is the consistent performance of the connection between the users' premises and the Direct Connect locations. This service is compatible with other services such as EC2, S3, and Amazon VPC and can be used in scenarios requiring high bandwidth between the Amazon network and the outside world. There are only two available ports located in the United States, but users can leverage external providers that offer guaranteed high bandwidth to these ports. Two different bandwidths can be chosen: 1 Gbps, priced at $0.30 per hour, and 10 Gbps, priced at $2.25 per hour. Inbound traffic is free; outbound traffic is priced at $0.02 per GB.

Amazon Route 53 implements dynamic DNS services that allow AWS resources to be reached through domain names different from the amazon.com domain. By leveraging the large and globally distributed network of Amazon DNS servers, AWS users can expose EC2 instances or S3 buckets as resources under a domain of their property, for which Amazon DNS servers become

authoritative.[3] EC2 instances are likely to be more dynamic than the physical machines, and S3 buckets might also exist for a limited time. To cope with such a volatile nature, the service provides AWS users with the capability of dynamically mapping names to resources as instances are launched on EC2 or as new buckets are created in S3. By interacting with the Route 53 Web service, users can manage a set of *hosted zones*, which represent the user domains controlled by the service, and edit the resources made available through it. Currently, a single user can have up to 100 zones. The costing model includes a fixed amount ($1 per zone per month) and a dynamic component that depends on the number of queries resolved by the service for the hosted zones ($0.50 per million queries for the first billion of queries a month, $0.25 per million queries over 1 billion of queries a month).

9.1.3.2 Messaging

Messaging services constitute the next step in connecting applications by leveraging AWS capabilities. The three different types of messaging services offered are *Amazon Simple Queue Service (SQS)*, *Amazon Simple Notification Service (SNS)*, and *Amazon Simple Email Service (SES)*.

Amazon SQS constitutes disconnected model for exchanging messages between applications by means of message queues, hosted within the AWS infrastructure. Using the AWS console or directly the underlying Web service AWS, users can create an unlimited number of message queues and configure them to control their access. Applications can send messages to any queue they have access to. These messages are securely and redundantly stored within the AWS infrastructure for a limited period of time, and they can be accessed by other (authorized) applications. While a message is being read, it is kept locked to avoid spurious processing from other applications. Such a lock will expire after a given period.

Amazon SNS provides a publish-subscribe method for connecting heterogeneous applications. With respect to Amazon SQS, where it is necessary to continuously poll a given queue for a new message to process, Amazon SNS allows applications to be notified when new content of interest is available. This feature is accessible through a Web service whereby AWS users can create a topic, which other applications can subscribe to. At any time, applications can publish content on a given topic and subscribers can be automatically notified. The service provides subscribers with different notification models (HTTP/HTTPS, email/email JSON, and SQS).

Amazon SES provides AWS users with a scalable email service that leverages the AWS infrastructure. Once users are signed up for the service, they have to provide an email that SES will use to send emails on their behalf. To activate the service, SES will send an email to verify the given address and provide the users with the necessary information for the activation. Upon verification, the user is given an SES sandbox to test the service, and he can request access to the production version. Using SES, it is possible to send either SMTP-compliant emails or raw emails by specifying email headers and Multipurpose Internet Mail Extension (MIME) types. Emails are queued for

[3] A DNS server is responsible for resolving a name to a corresponding IP address. Since DNS servers implement a distributed database without a single global control, a single DNS server does not have the complete knowledge of all the mappings between names and IP addresses, but it has direct knowledge only of a small subset of them. Such a DNS server is therefore authoritative for these names because it can directly resolve the names. For resolving the other names, the nearest authoritative DNS is contacted.

delivery, and the users are notified of any failed delivery. SES also provides a wide range of statistics that help users to improve their email campaigns for effective communication with customers.

With regard to the costing, all three services do not require a minimum commitment but are based on a pay-as-you go model. Currently, users are not charged until they reach a minimum threshold. In addition, data transfer-in is not charged, but data transfer-out is charged by ranges.

9.1.4 Additional services

Besides compute, storage, and communication services, AWS provides a collection of services that allow users to utilize services in aggregation. The two relevant services are *Amazon CloudWatch* and *Amazon Flexible Payment Service (FPS)*.

Amazon CloudWatch is a service that provides a comprehensive set of statistics that help developers understand and optimize the behavior of their application hosted on AWS. CloudWatch collects information from several other AWS services: EC2, S3, SimpleDB, CloudFront, and others. Using CloudWatch, developers can see a detailed breakdown of their usage of the service they are renting on AWS and can devise more efficient and cost-saving applications. Earlier services of CloudWatch were offered only through subscription, but now it is made available for free to all the AWS users.

Amazon FPS infrastructure allows AWS users to leverage Amazon's billing infrastructure to sell goods and services to other AWS users. Using Amazon FPS, developers do not have to set up alternative payment methods, and they can charge users via a billing service. The payment models available through FPS include one-time payments and delayed and periodic payments, required by subscriptions and usage-based services, transactions, and aggregate multiple payments.

9.1.5 Summary

Amazon provides a complete set of services for developing, deploying, and managing cloud computing systems by leveraging the large and distributed AWS infrastructure. Developers can use EC2 to control and configure the computing infrastructure hosted in the cloud. They can leverage other services, such as AWS CloudFormation, Elastic Beanstalk, or Elastic MapReduce, if they do not need complete control over the computing stack. Applications hosted in the AWS Cloud can leverage S3, SimpleDB, or other storage services to manage structured and unstructured data. These services are primarily meant for storage, but other options, such as Amazon SQS, SNS, and SES, provide solutions for dynamically connecting applications from both inside and outside the AWS Cloud. Network connectivity to AWS applications is addressed by Amazon VPC and Amazon Direct Connect.

9.2 Google AppEngine

Google AppEngine is a PaaS implementation that provides services for developing and hosting scalable Web applications. AppEngine is essentially a distributed and scalable runtime environment that leverages Google's distributed infrastructure to scale out applications facing a large number of requests by allocating more computing resources to them and balancing the load among them. The runtime is completed by a collection of services that allow developers to design and implement

applications that naturally scale on AppEngine. Developers can develop applications in Java, Python, and Go, a new programming language developed by Google to simplify the development of Web applications. Application usage of Google resources and services is metered by AppEngine, which bills users when their applications finish their free quotas.

9.2.1 Architecture and core concepts

AppEngine is a platform for developing scalable applications accessible through the Web (see Figure 9.2). The platform is logically divided into four major components: infrastructure, the runtime environment, the underlying storage, and the set of scalable services that can be used to develop applications.

9.2.1.1 Infrastructure

AppEngine hosts Web applications, and its primary function is to serve users requests efficiently. To do so, AppEngine's infrastructure takes advantage of many servers available within Google datacenters. For each HTTP request, AppEngine locates the servers hosting the application that processes the request, evaluates their load, and, if necessary, allocates additional resources (i.e., servers) or redirects the request to an existing server. The particular design of applications, which does not expect any state information to be implicitly maintained between requests to the same application, simplifies the work of the infrastructure, which can redirect each of the requests to any of the servers hosting the target application or even allocate a new one.

The infrastructure is also responsible for monitoring application performance and collecting statistics on which the billing is calculated.

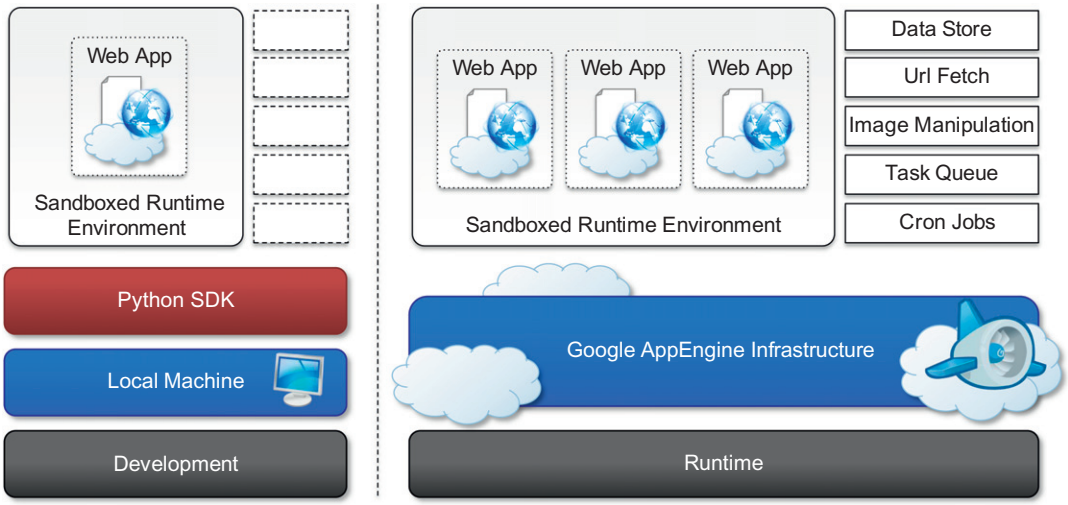

FIGURE 9.2

Google AppEngine platform architecture.

9.2.1.2 Runtime environment

The runtime environment represents the execution context of applications hosted on AppEngine. With reference to the AppEngine infrastructure code, which is always active and running, the runtime comes into existence when the request handler starts executing and terminates once the handler has completed.

Sandboxing

One of the major responsibilities of the runtime environment is to provide the application environment with an isolated and protected context in which it can execute without causing a threat to the server and without being influenced by other applications. In other words, it provides applications with a *sandbox*.

Currently, AppEngine supports applications that are developed only with managed or interpreted languages, which by design require a runtime for translating their code into executable instructions. Therefore, sandboxing is achieved by means of modified runtimes for applications that disable some of the common features normally available with their default implementations. If an application tries to perform any operation that is considered potentially harmful, an exception is thrown and the execution is interrupted. Some of the operations that are not allowed in the sandbox include writing to the server's file system; accessing computer through network besides using *Mail*, *UrlFetch*, and *XMPP*; executing code outside the scope of a request, a queued task, and a cron job; and processing a request for more than 30 seconds.

Supported runtimes

Currently, it is possible to develop AppEngine applications using three different languages and related technologies: *Java*, *Python*, and *Go*.

AppEngine currently supports Java 6, and developers can use the common tools for Web application development in Java, such as the *Java Server Pages (JSP)*, and the applications interact with the environment by using the *Java Servlet* standard. Furthermore, access to AppEngine services is provided by means of Java libraries that expose specific interfaces of provider-specific implementations of a given abstraction layer. Developers can create applications with the AppEngine Java SDK, which allows developing applications with either Java 5 or Java 6 and by using any Java library that does not exceed the restrictions imposed by the sandbox.

Support for Python is provided by an optimized Python 2.5.2 interpreter. As with Java, the runtime environment supports the Python standard library, but some of the modules that implement potentially harmful operations have been removed, and attempts to import such modules or to call specific methods generate exceptions. To support application development, AppEngine offers a rich set of libraries connecting applications to AppEngine services. In addition, developers can use a specific Python Web application framework, called *webapp*, simplifying the development of Web applications.

The Go runtime environment allows applications developed with the Go programming language to be hosted and executed in AppEngine. Currently the release of Go that is supported by AppEngine is r58.1. The SDK includes the compiler and the standard libraries for developing applications in Go and interfacing it with AppEngine services. As with the Python environment, some of the functionalities have been removed or generate a runtime exception. In addition, developers can include third-party libraries in their applications as long as they are implemented in pure Go.

9.2.1.3 Storage

AppEngine provides various types of storage, which operate differently depending on the volatility of the data. There are three different levels of storage: in memory-cache, storage for semistructured data, and long-term storage for static data. In this section, we describe *DataStore* and the use of static file servers. We cover *MemCache* in the application services section.

Static file servers

Web applications are composed of dynamic and static data. Dynamic data are a result of the logic of the application and the interaction with the user. Static data often are mostly constituted of the components that define the graphical layout of the application (CSS files, plain HTML files, JavaScript files, images, icons, and sound files) or data files. These files can be hosted on static file servers, since they are not frequently modified. Such servers are optimized for serving static content, and users can specify how dynamic content should be served when uploading their applications to AppEngine.

DataStore

DataStore is a service that allows developers to store semistructured data. The service is designed to scale and optimized to quickly access data. DataStore can be considered as a large object database in which to store objects that can be retrieved by a specified key. Both the type of the key and the structure of the object can vary.

With respect to the traditional Web applications backed by a relational database, DataStore imposes less constraint on the regularity of the data but, at the same time, does not implement some of the features of the relational model (such as reference constraints and join operations). These design decisions originated from a careful analysis of data usage patterns for Web applications and were taken in order to obtain a more scalable and efficient data store. The underlying infrastructure of *DataStore* is based on *Bigtable* [93], a redundant, distributed, and semistructured data store that organizes data in the form of tables (see Section 8.2.1).

DataStore provides high-level abstractions that simplify interaction with Bigtable. Developers define their data in terms of *entity* and *properties*, and these are persisted and maintained by the service into tables in *Bigtable*. An entity constitutes the level of granularity for the storage, and it identifies a collection of properties that define the data it stores. Properties are defined according to one of the several primitive types supported by the service. Each entity is associated with a key, which is either provided by the user or created automatically by AppEngine. An entity is associated with a *named kind* that AppEngine uses to optimize its retrieval from Bigtable. Although entities and properties seem to be similar to rows and tables in SQL, there are a few differences that have to be taken into account. Entities of the same kind might not have the same properties, and properties of the same name might contain values of different types. Moreover, properties can store different versions of the same values. Finally, keys are immutable elements and, once created, they cannot be changed.

DataStore also provides facilities for creating indexes on data and to update data within the context of a transaction. Indexes are used to support and speed up queries. A query can return zero or more objects of the same kind or simply the corresponding keys. It is possible to query the data store by specifying either the key or conditions on the values of the properties. Returned result sets can be sorted by key value or properties value. Even though the queries are quite similar to SQL

queries, their implementation is substantially different. DataStore has been designed to be extremely fast in returning result sets; to do so it needs to know in advance all the possible queries that can be done for a given kind, because it stores for each of them a separate index. The indexes are provided by the user while uploading the application to AppEngine and can be automatically defined by the development server. When the developer tests the application, the server monitors all the different types of queries made against the simulated data store and creates an index for them. The structure of the indexes is saved in a configuration file and can be further changed by the developer before uploading the application. The use of precomputed indexes makes the query execution time-independent from the size of the stored data but only influenced by the size of the result set.

The implementation of transaction is limited in order to keep the store scalable and fast. AppEngine ensures that the update of a single entity is performed atomically. Multiple operations on the same entity can be performed within the context of a transaction. It is also possible to update multiple entities atomically. This is only possible if these entities belong to the same *entity group.* The entity group to which an entity belongs is specified at the time of entity creation and cannot be changed later. With regard to concurrency, AppEngine uses an *optimistic concurrency control*: If one user tries to update an entity that is already being updated, the control returns and the operation fails. Retrieving an entity never incurs into exceptions.

9.2.1.4 Application services

Applications hosted on AppEngine take the most from the services made available through the runtime environment. These services simplify most of the common operations that are performed in Web applications: access to data, account management, integration of external resources, messaging and communication, image manipulation, and asynchronous computation.

UrlFetch

Web 2.0 has introduced the concept of composite Web applications. Different resources are put together and organized as meshes within a single Web page. Meshes are fragments of HTML generated in different ways. They can be directly obtained from a remote server or rendered from an XML document retrieved from a Web service, or they can be rendered by the browser as the result of an embedded and remote component. A common characteristic of all these examples is the fact that the resource is not local to the server and often not even in the same administrative domain. Therefore, it is fundamental for Web applications to be able to retrieve remote resources.

The sandbox environment does not allow applications to open arbitrary connections through sockets, but it does provide developers with the capability of retrieving a remote resource through HTTP/HTTPS by means of the *UrlFetch* service. Applications can make synchronous and asynchronous Web requests and integrate the resources obtained in this way into the normal request-handling cycle of the application. One of the interesting features of UrlFetch is the ability to set deadlines for requests so that they can be completed (or aborted) within a given time. Moreover, the ability to perform such requests asynchronously allows the applications to continue with their logic while the resource is retrieved in the background. UrlFetch is not only used to integrate meshes into a Web page but also to leverage remote Web services in accordance with the SOA reference model for distributed applications.

MemCache

AppEngine provides developers with access to fast and reliable storage, which is DataStore. Despite this, the main objective of the service is to serve as a scalable and long-term storage, where data are persisted to disk redundantly in order to ensure reliability and availability of data against failures. This design poses a limit on how much faster the store can be compared to other solutions, especially for objects that are frequently accessed—for example, at each Web request.

AppEngine provides caching services by means of *MemCache*. This is a distributed in-memory cache that is optimized for fast access and provides developers with a volatile store for the objects that are frequently accessed. The caching algorithm implemented by MemCache will automatically remove the objects that are rarely accessed. The use of MemCache can significantly reduce the access time to data; developers can structure their applications so that each object is first looked up into MemCache and if there is a miss, it will be retrieved from DataStore and put into the cache for future lookups.

Mail and instant messaging

Communication is another important aspect of Web applications. It is common to use email for following up with users about operations performed by the application. Email can also be used to trigger activities in Web applications. To facilitate the implementation of such tasks, AppEngine provides developers with the ability to send and receive mails through *Mail*. The service allows sending email on behalf of the application to specific user accounts. It is also possible to include several types of attachments and to target multiple recipients. Mail operates asynchronously, and in case of failed delivery the sending address is notified through an email detailing the error.

AppEngine provides also another way to communicate with the external world: the Extensible Messaging and Presence Protocol (XMPP). Any chat service that supports XMPP, such as Google Talk, can send and receive chat messages to and from the Web application, which is identified by its own address. Even though the chat is a communication medium mostly used for human interactions, XMPP can be conveniently used to connect the Web application with chat bots or to implement a small administrative console.

Account management

Web applications often keep various data that customize their interaction with users. These data normally go under the user profile and are attached to an account. AppEngine simplifies account management by allowing developers to leverage Google account management by means of *Google Accounts*. The integration with the service also allows Web applications to offload the implementation of authentication capabilities to Google's authentication system.

Using Google Accounts, Web applications can conveniently store profile settings in the form of key-value pairs, attach them to a given Google account, and quickly retrieve them once the user authenticates. With respect to a custom solution, the use of Google Accounts requires users to have a Google account, but it does not require any further implementation. The use of Google Accounts is particularly advantageous for developing Web applications within a corporate environment using Google Apps. In this case, the applications can be easily integrated with all the other services (and profile settings) included in Google Apps.

Image manipulation

Web applications render pages with graphics. Often simple operations, such as adding watermarks or applying simple filters, are required. AppEngine allows applications to perform image resizing, rotation, mirroring, and enhancement by means of *Image Manipulation*, a service that is also used in other Google products. Image Manipulation is mostly designed for lightweight image processing and is optimized for speed.

9.2.1.5 Compute services

Web applications are mostly designed to interface applications with users by means of a ubiquitous channel, that is, the Web. Most of the interaction is performed synchronously: Users navigate the Web pages and get instantaneous feedback in response to their actions. This feedback is often the result of some computation happening on the Web application, which implements the intended logic to serve the user request. Sometimes this approach is not applicable—for example, in long computations or when some operations need to be triggered at a given point in time. A good design for these scenarios provides the user with immediate feedback and a notification once the required operation is completed. AppEngine offers additional services such as *Task Queues* and *Cron Jobs* that simplify the execution of computations that are off-bandwidth or those that cannot be performed within the timeframe of the Web request.

Task queues

Task Queues allow applications to submit a task for a later execution. This service is particularly useful for long computations that cannot be completed within the maximum response time of a request handler. The service allows users to have up to 10 queues that can execute tasks at a configurable rate.

In fact, a task is defined by a Web request to a given URL, and the queue invokes the request handler by passing the payload as part of the Web request to the handler. It is the responsibility of the request handler to perform the "task execution," which is seen from the queue as a simple Web request. The queue is designed to reexecute the task in case of failure in order to avoid transient failures preventing the task from a successful completion.

Cron jobs

Sometimes the length of computation might not be the primary reason that an operation is not performed within the scope of the Web request. It might be possible that the required operation needs to be performed at a specific time of the day, which does not coincide with the time of the Web request. In this case, it is possible to schedule the required operation at the desired time by using the *Cron Jobs* service. This service operates similarly to Task Queues but invokes the request handler specified in the task at a given time and does not reexecute the task in case of failure. This behavior can be useful to implement maintenance operations or send periodic notifications.

9.2.2 Application life cycle

AppEngine provides support for almost all the phases characterizing the life cycle of an application: testing and development, deployment, and monitoring. The SDKs released by Google provide

developers with most of the functionalities required by these tasks. Currently there are two SDKs available for development: Java SDK and Python SDK.

9.2.2.1 Application development and testing

Developers can start building their Web applications on a local development server. This is a self-contained environment that helps developers tune applications without uploading them to AppEngine. The development server simulates the AppEngine runtime environment by providing a mock implementation of DataStore, MemCache, UrlFetch, and the other services leveraged by Web applications. Besides hosting Web applications, the development server contains a complete set of monitoring features that are helpful to profile the behavior of applications, especially regarding access to the DataStore service and the queries performed against it. This is a particularly important feature that will be of relevance in deploying the application to AppEngine. As discussed earlier, AppEngine builds indexes for each of the queries performed by a given application in order to speed up access to the relevant data. This capability is enabled by *a priori* knowledge about all the possible queries made by the application; such knowledge is made available to AppEngine by the developer while uploading the application. The development server analyzes application behavior while running and traces all the queries made during testing and development, thus providing the required information about the indexes to be built.

Java SDK

The Java SDK provides developers with the facility for building applications with the Java 5 and Java 6 runtime environments. Alternatively, it is possible to develop applications within the Eclipse development environment by using the Google AppEngine plug-in, which integrates the features of the SDK within the powerful Eclipse environment. Using the Eclipse software installer, it is possible to download and install Java SDK, Google Web Toolkit, and Google AppEngine plug-ins into Eclipse. These three components allow developers to program powerful and rich Java applications for AppEngine.

The SDK supports the development of applications by using the *servlet* abstraction, which is a common development model. Together with servlets, many other features are available to build applications. Moreover, developers can easily create Web applications by using the *Eclipse Web Platform*, which provides a set of tools and components.

The plug-in allows developing, testing, and deploying applications on AppEngine. Other tasks, such as retrieving the log of applications, are available by means of command-line tools that are part of the SDK.

Python SDK

The Python SDK allows developing Web applications for AppEngine with Python 2.5. It provides a standalone tool, called *GoogleAppEngineLauncher*, for managing Web applications locally and deploying them to AppEngine. The tool provides a convenient user interface that lists all the available Web applications, controls their execution, and integrates them with the default code editor for editing application files. In addition, the launcher provides access to some important services for application monitoring and analysis, such as the logs, the SDK console, and the dashboard. The log console captures all the information that is logged by the application while it is running. The console SDK provides developers with a Web interface via which they can see the application profile

in terms of utilized resource. This feature is particularly useful because it allows developers to preview the behavior of the applications once they are deployed on AppEngine, and it can be used to tune applications made available through the runtime.

The Python implementation of the SDK also comes with an integrated Web application framework called *webapp* that includes a set of models, components, and tools that simplify the development of Web applications and enforce a set of coherent practices. This is not the only Web framework that can be used to develop Web applications. There are dozens of available Python Web frameworks that can be used. However, due to the restrictions enforced by the sandboxed environment, all of them cannot be used seamlessly. The *webapp* framework has been reimplemented and made available in the Python SDK so that it can be used with AppEngine. Another Web framework that is known to work well is *Django*.[4]

The SDK is completed by a set of command-line tools that allows developers to perform all the operations available through the launcher and more from the command shell.

9.2.2.2 Application deployment and management

Once the application has been developed and tested, it can be deployed on AppEngine with a simple click or command-line tool. Before performing such task, it is necessary to create an application identifier, which will be used to locate the application from the Web browser by typing the address *http://<application-id>.appspot.com*. Alternatively, it is also possible to map the application with a registered DNS domain name. This is particularly useful for commercial development, where users want to make the application available through a more appropriate name.

An application identifier is mandatory because it allows unique identification of the application while it's interacting with AppEngine. Developers use an app identifier to upload and update applications. Besides being unique, it also needs to be compliant to the rules that are enforced for domain names. It is possible to register an application identifier by logging into AppEngine and selecting the "Create application" option. It is also possible to provide an application title that is descriptive of the application; the title can be changed over time.

Once an application identifier has been created, it is possible to deploy an application on AppEngine. This task can be done using either the respective development environment (*GoogleAppEngineLauncher* and *Google AppEngine* plug-in) or the command-line tools. Once the application is uploaded, nothing else needs to be done to make it available. AppEngine will take care of everything. Developers can then manage the application by using the administrative console. This is the primary tool used for application monitoring and provides users with insight into resource usage (CPU, bandwidth) and services and other useful counters. It is also possible to manage multiple versions of a single application, select the one available for the release, and manage its billing-related issues.

9.2.3 Cost model

AppEngine provides a free service with limited quotas that get reset every 24 hours. Once the application has been tested and tuned for AppEngine, it is possible to set up a billing account and obtain more allowance and be charged on a pay-per-use basis. This allows developers to identify the appropriate daily budget that they want to allocate for a given application.

[4] www.djangoproject.com.

An application is measured against *billable quotas*, *fixed quotas*, and *per-minute quotas*. Google AppEngine uses these quotas to ensure that users do not spend more than the allocated budget and that applications run without being influenced by each other from a performance point of view. Billable quotas identify the daily quotas that are set by the application administrator and are defined by the daily budget allocated for the application. AppEngine will ensure that the application does not exceed these quotas. Free quotas are part of the billable quota and identify the portion of the quota for which users are not charged. Fixed quotas are internal quotas set by AppEngine that identify the infrastructure boundaries and define operations that the application can carry out on the infrastructure (services and runtime). These quotas are generally bigger than billable quotas and are set by AppEngine to avoid applications impacting each other's performance or overloading the infrastructure. The costing model also includes per-minute quotas, which are defined in order to avoid applications consuming all their credit in a very limited period of time, monopolizing a resource, and creating service interruption for other applications.

Once an application reaches the quota for a given resource, the resource is depleted and will not be available to the application until the quota is replenished. Once a resource is depleted, subsequent requests to that resource will generate an error or an exception. Resources such as CPU time and incoming or outgoing bandwidth will return an "HTTP 403" error page to users; all the other resources and services will generate an exception that can be trapped in code to provide more useful feedback to users.

Resources and services quotas are organized into free default quotas and billing-enabled default quotas. For these two categories, a daily limit and a maximum rate are defined. A detailed explanation of how quotas work, their limits, and the amount that is charged to the user can be found on the AppEngine Website at the following Internet address: http://code.google.com/appengine/docs/quotas.html.

9.2.4 Observations

AppEngine, a framework for developing scalable Web applications, leverages Google's infrastructure. The core components of the service are a scalable and sandboxed runtime environment for executing applications and a collection of services that implement most of the common features required for Web development and that help developers build applications that are easy to scale. One of the characteristic elements of AppEngine is the use of simple interfaces that allow applications to perform specific operations that are optimized and designed to scale. Building on top of these blocks, developers can build applications and let AppEngine scale them out when needed.

With respect to the traditional approach to Web development, the implementation of rich and powerful applications requires a change of perspective and more effort. Developers have to become familiar with the capabilities of AppEngine and implement the required features in a way that conforms with the AppEngine application model.

9.3 Microsoft Azure

Microsoft Windows Azure is a cloud operating system built on top of Microsoft datacenters' infrastructure and provides developers with a collection of services for building applications with cloud technology. Services range from compute, storage, and networking to application connectivity,

access control, and business intelligence. Any application that is built on the Microsoft technology can be scaled using the Azure platform, which integrates the scalability features into the common Microsoft technologies such as Microsoft Windows Server 2008, SQL Server, and ASP.NET.

Figure 9.3 provides an overview of services provided by Azure. These services can be managed and controlled through the *Windows Azure Management Portal*, which acts as an administrative console for all the services offered by the Azure platform. In this section, we present the core features of the major services available with Azure.

9.3.1 Azure core concepts

The Windows Azure platform is made up of a foundation layer and a set of developer services that can be used to build scalable applications. These services cover compute, storage, networking, and identity management, which are tied together by middleware called *AppFabric*. This scalable computing environment is hosted within Microsoft datacenters and accessible through the Windows Azure Management Portal. Alternatively, developers can recreate a Windows Azure environment (with limited capabilities) on their own machines for development and testing purposes. In this section, we provide an overview of the Azure middleware and its services.

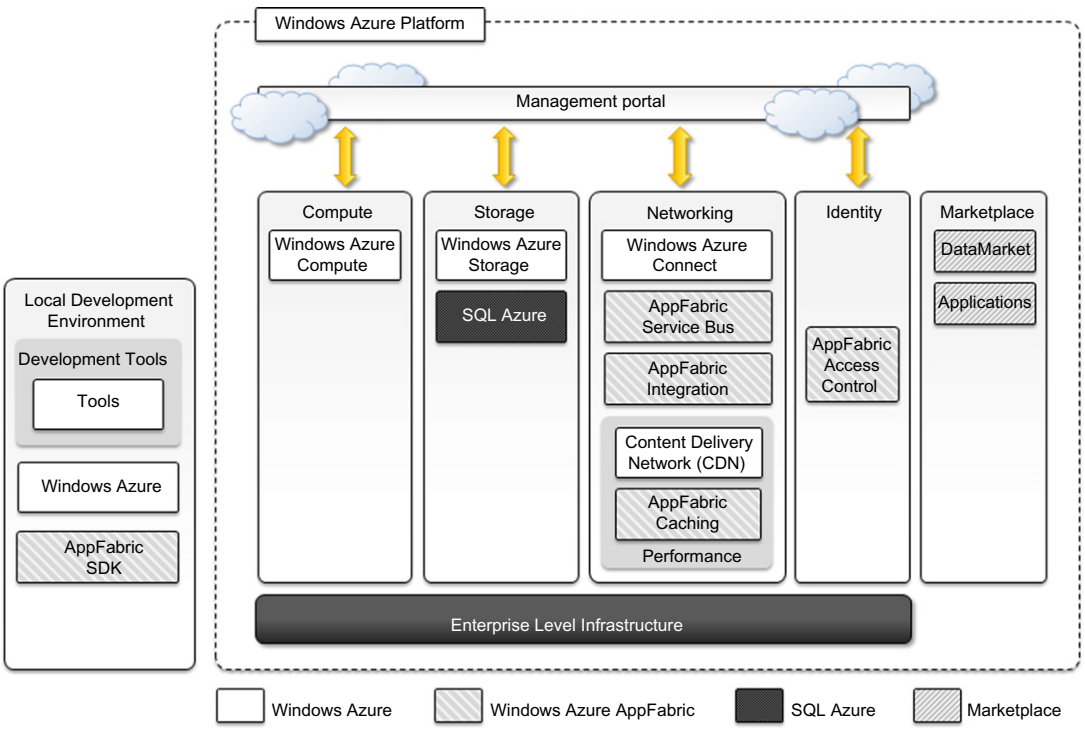

FIGURE 9.3

Microsoft Windows Azure Platform Architecture.

9.3.1.1 Compute services

Compute services are the core components of Microsoft Windows Azure, and they are delivered by means of the abstraction of *roles*. A role is a runtime environment that is customized for a specific compute task. Roles are managed by the Azure operating system and instantiated on demand in order to address surges in application demand. Currently, there are three different roles: *Web role*, *Worker role*, and *Virtual Machine (VM) role*.

Web role

The *Web role* is designed to implement scalable Web applications. Web roles represent the units of deployment of Web applications within the Azure infrastructure. They are hosted on the IIS 7 Web Server, which is a component of the infrastructure that supports Azure. When Azure detects peak loads in the request made to a given application, it instantiates multiple Web roles for that application and distributes the load among them by means of a load balancer.

Since version 3.5, the .NET technology natively supports Web roles; developers can directly develop their applications in Visual Studio, test them locally, and upload to Azure. It is possible to develop ASP.NET (*ASP.NET Web Role* and *ASP.NET MVC 2 Web Role*) and WCF (*WCF Service Web Role*) applications. Since IIS 7 also supports the PHP runtime environment by means of the FastCGI module, Web roles can be used to run and scale PHP Web applications on Azure (*CGI Web Role*). Other Web technologies that are not integrated with IIS can still be hosted on Azure (i.e., Java Server Pages on Apache Tomcat), but there is no advantage to using a Web role over a Worker role.

Worker role

Worker roles are designed to host general compute services on Azure. They can be used to quickly provide compute power or to host services that do not communicate with the external world through HTTP. A common practice for Worker roles is to use them to provide background processing for Web applications developed with Web roles.

Developing a worker role is like a developing a service. Compared to a Web role whose computation is triggered by the interaction with an HTTP client (i.e., a browser), a Worker role runs continuously from the creation of its instance until it is shut down. The Azure SDK provides developers with convenient APIs and libraries that allow connecting the role with the service provided by the runtime and easily controlling its startup as well as being notified of changes in the hosting environment. As with Web roles, the .NET technology provides complete support for Worker roles, but any technology that runs on a Windows Server stack can be used to implement its core logic. For example, Worker roles can be used to host Tomcat and serve JSP-based applications.

Virtual machine role

The *Virtual Machine role* allows developers to fully control the computing stack of their compute service by defining a custom image of the Windows Server 2008 R2 operating system and all the service stack required by their applications. The Virtual Machine role is based on the Windows Hyper-V virtualization technology (see Section 3.6.3), which is natively integrated in the Windows server technology at the base of Azure. Developers can image a Windows server installation complete with all the required applications and components, save it into a Virtual Hard Disk (VHD)

Table 9.7 Windows Azure Compute Instances Characteristics, 2011–2012

Compute Instance Type	CPU	Memory	Instance Storage	I/O Performance	Hourly Cost (USD)
Extra Small	1.0 GHz	768 MB	20 GB	Low	$0.04
Small	1.6 GHz	1.75 GB	225 GB	Moderate	$0.12
Medium	2 × 1.6 GHz	3.5 GB	490 GB	High	$0.24
Large	4 × 1.6 GHz	7 GB	1,000 GB	High	$0.48
Extra Large	8 × 1.6 GHz	14 GB	2,040 GB	High	$0.96

file, and upload it to Windows Azure to create compute instances on demand. Different types of instances are available, and Table 9.7 provides an overview of the options offered during 2011–2012.

Compared to the Worker and Web roles, the VM role provides finer control of the compute service and resource that are deployed on the Azure Cloud. An additional administrative effort is required for configuration, installation, and management of services.

9.3.1.2 Storage services

Compute resources are equipped with local storage in the form of a directory on the local file system that can be used to temporarily store information that is useful for the current execution cycle of a role. If the role is restarted and activated on a different physical machine, this information is lost.

Windows Azure provides different types of storage solutions that complement compute services with a more durable and redundant option compared to local storage. Compared to local storage, these services can be accessed by multiple clients at the same time and from everywhere, thus becoming a general solution for storage.

Blobs

Azure allows storing large amount of data in the form of binary large objects (BLOBs) by means of the *blobs* service. This service is optimal to store large text or binary files. Two types of blobs are available:

- *Block blobs.* Block blobs are composed of blocks and are optimized for sequential access; therefore they are appropriate for media streaming. Currently, blocks are of 4 MB, and a single block blob can reach 200 GB in dimension.
- *Page blobs.* Page blobs are made of pages that are identified by an offset from the beginning of the blob. A page blob can be split into multiple pages or constituted of a single page. This type of blob is optimized for random access and can be used to host data different from streaming. Currently, the maximum dimension of a page blob can be 1 TB.

Blobs storage provides users with the ability to describe the data by adding metadata. It is also possible to take snapshots of a blob for backup purposes. Moreover, to optimize its distribution, blobs storage can leverage the Windows Azure CDN so that blobs are kept close to users requesting them and can be served efficiently.

Azure drive

Page blobs can be used to store an entire file system in the form of a single *Virtual Hard Drive (VHD)* file. This can then be mounted as a part of the NTFS file system by Azure compute resources, thus providing persistent and durable storage. A page blob mounted as part of an NTFS tree is called an *Azure Drive*.

Tables

Tables constitute a semistructured storage solution, allowing users to store information in the form of entities with a collection of properties. Entities are stored as rows in the table and are identified by a key, which also constitutes the unique index built for the table. Users can insert, update, delete, and select a subset of the rows stored in the table. Unlike SQL tables, there are no schema enforcing constraints on the properties of entities and there is no facility for representing relationships among entities. For this reason, tables are more similar to spreadsheets rather than SQL tables.

The service is designed to handle large amounts of data and queries returning huge result sets. This capability is supported by partial result sets and table partitions. A partial result set is returned together with a continuation token, allowing the client to resume the query for large result sets. Table partitions allow tables to be divided among several servers for load-balancing purposes. A partition is identified by a key, which is represented by three of the columns of the table.

Currently, a table can contain up to 100 TB of data, and rows can have up to 255 properties, with a maximum of 1 MB for each row. The maximum dimension of a row key and partition keys is 1 KB.

Queues

Queue storage allows applications to communicate by exchanging messages through durable queues, thus avoiding lost or unprocessed messages. Applications enter messages into a queue, and other applications can read them in a first-in, first-out (FIFO) style.

To ensure that messages get processed, when an application reads a message it is marked as invisible; hence it will not be available to other clients. Once the application has completed processing the message, it needs to explicitly delete the message from the queue. This two-phase process ensures that messages get processed before they are removed from the queue, and the client failures do not prevent messages from being processed. At the same time, this is also a reason that the queue does not enforce a strict FIFO model: Messages that are read by applications that crash during processing are made available again after a timeout, during which other messages can be read by other clients. An alternative to reading a message is *peeking*, which allows retrieving the message but letting it stay visible in the queue. Messages that are peeked are not considered processed.

All the services described are geo-replicated three times to ensure their availability in case of major disasters. *Geo-replication* involves the copying of data into a different datacenter that is hundreds or thousands of miles away from the original datacenter.

9.3.1.3 Core infrastructure: AppFabric

AppFabric is a comprehensive middleware for developing, deploying, and managing applications on the cloud or for integrating existing applications with cloud services. AppFabric implements an optimized infrastructure supporting scaling out and high availability; sandboxing and

multitenancy; state management; and dynamic address resolution and routing. On top of this infrastructure, the middleware offers a collection of services that simplify many of the common tasks in a distributed application, such as communication, authentication and authorization, and data access. These services are available through language-agnostic interfaces, thus allowing developers to build heterogeneous applications.

Access control

AppFabric provides the capability of encoding access control to resources in Web applications and services into a set of rules that are expressed outside the application code base. These rules give a great degree of flexibility in terms of the ability to secure components of the application and define access control policies for users and groups.

Access control services also integrate several authentication providers into a single coherent identity management framework. Applications can leverage Active Directory, Windows Live, Google, Facebook, and other services to authenticate users. This feature also allows easy building of hybrid systems, with some parts existing in the private premises and others deployed in the public cloud.

Service bus

Service Bus constitutes the messaging and connectivity infrastructure provided with AppFabric for building distributed and disconnected applications in the Azure Cloud and between the private premises and the Azure Cloud. Service Bus allows applications to interact with different protocols and patterns over a reliable communication channel that guarantees delivery.

The service is designed to allow transparent network traversal and to simplify the development of loosely coupled applications, without renouncing security and reliability and letting developers focus on the logic of the interaction rather than the details of its implementation. Service Bus allows services to be available by simple URLs, which are untied from their deployment location. It is possible to support publish-subscribe models, full-duplex communications point to point as well as in a peer-to-peer environment, unicast and multicast message delivery in one-way communications, and asynchronous messaging to decouple application components.

In order to leverage these features, applications need to be connected to the bus, which provides these services. A connection is the Service Bus element that is priced by Azure on a pay-as-you-go basis. Users are billed on a connections-per-month basis, and they can buy advance "connection packs," which have a discounted price, if they can estimate their needs in advance.

Azure cache

Windows Azure provides a set of durable storage solutions that allow applications to persist their data. These solutions are based on disk storage, which might constitute a bottleneck for the applications that need to gracefully scale along the clients' requests and dataset size dimensions.

Azure Cache is a service that allows developers to quickly access data persisted on Windows Azure storage or in SQL Azure. The service implements a distributed in-memory cache of which the size can be dynamically adjusted by applications according to their needs. It is possible to store any .NET managed object as well as many common data formats (table rows, XML, and binary data) and control its access by applications. Azure Cache is delivered as a service, and it can be

easily integrated with applications. This is a particularly true for ASP.NET applications, which already integrate providers for session state and page output caching based on Azure Cache.

The service is priced according the size of cache allocated by applications per month, despite their effective use of the cache. Currently, several cache sizes are available, ranging from 128 MB ($45/month) to 4 GB ($325/month).

9.3.1.4 Other services

Compute, storage, and middleware services constitute the core components of the Windows Azure platform. Besides these, other services and components simplify the development and integration of applications with the Azure Cloud. An important area for these services is applications connectivity, including virtual networking and content delivery.

Windows Azure virtual network

Networking services for applications are offered under the name *Windows Azure Virtual Network*, which includes *Windows Azure Connect* and *Windows Azure Traffic Manager*.

Windows Azure Connect allows easy setup of IP-based network connectivity among machines hosted on the private premises and the roles deployed on the Azure Cloud. This service is particularly useful in the case of VM roles, where machines hosted in the Azure Cloud become part of the private network of the enterprise and can be managed with the same tools used in the private premises.

Windows Azure Traffic Manager provides load-balancing features for services listening to the HTTP or HTTPS ports and hosted on multiple roles. It allows developers to choose from three different load-balancing strategies: Performance, Round-Robin, and Failover.

Currently, the two services are still in beta phase and are available for free only by invitation.

Windows Azure content delivery network

Windows Azure Content Delivery Network (CDN) is the content delivery network solution that improves the content delivery capabilities of Windows Azure Storage and several other Microsoft services, such as *Microsoft Windows Update* and *Bing* maps. The service allows serving of Web objects (images, static HTML, CSS, and scripts) as well as streaming content by using a network of 24 locations distributed across the world.

9.3.2 SQL Azure

SQL Azure is a relational database service hosted on Windows Azure and built on the SQL Server technologies. The service extends the capabilities of SQL Server to the cloud and provides developers with a scalable, highly available, and fault-tolerant relational database. SQL Azure is accessible from either the Windows Azure Cloud or any other location that has access to the Azure Cloud. It is fully compatible with the interface exposed by SQL Server, so applications built for SQL Server can transparently migrate to SQL Azure. Moreover, the service is fully manageable using REST APIs, allowing developers to control databases deployed in the Azure Cloud as well as the firewall rules set up for their accessibility.

Figure 9.4 shows the architecture of SQL Azure. Access to SQL Azure is based on the Tabular Data Stream (TDS) protocol, which is the communication protocol underlying all the different

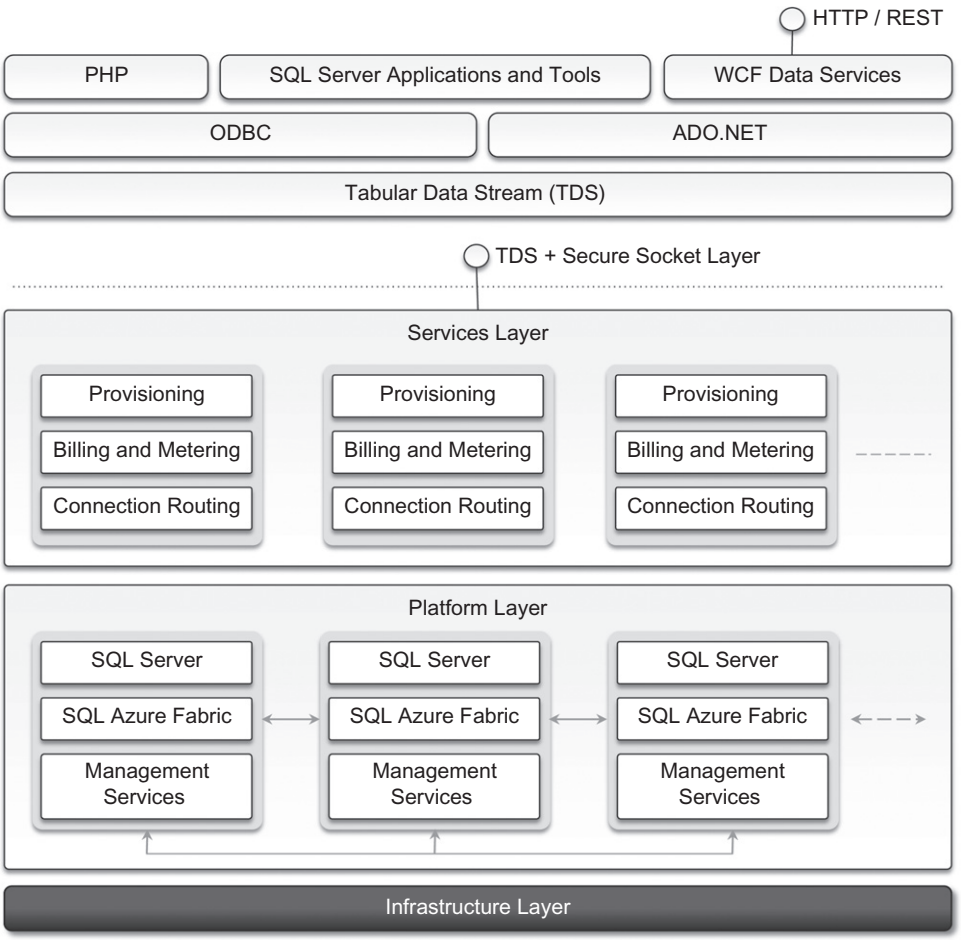

FIGURE 9.4

SQL Azure architecture.

interfaces used by applications to connect to a SQL Server-based installation such as ODBC and ADO.NET. On the SQL Azure side, access to data is mediated by the service layer, which provides provisioning, billing, and connection-routing services. These services are logically part of server instances, which are managed by SQL Azure Fabric. This is the distributed database middleware that constitutes the infrastructure of SQL Azure and that is deployed on Microsoft datacenters.

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

that closely resemble physical SQL Servers: They have a fully qualified domain name under the *database.windows.net* (i.e., *server-name.database.windows.net*) domain name. This simplifies the management tasks and the interaction with SQL Azure from client applications. SQL Azure ensures that multiple copies of each server are maintained within the Azure Cloud and that these copies are kept synchronized when client applications insert, update, and delete data on them.

Currently, the SQL Azure service is billed according to space usage and the type of edition. Currently, two different editions are available: Web Edition and Business Edition. The former is suited for small Web applications and supports databases with a maximum size of 1 GB or 5 GB. The latter is suited for independent software vendors, line-of-business applications, and enterprise applications and supports databases with a maximum size from 10 GB to 50 GB, in increments of 10 GB. Moreover, a bandwidth fee applies for any data transfer trespassing the Windows Azure Cloud or the region where the database is located. A monthly fee per user/database is also charged and is based on the peak size the database reaches during the month.

9.3.3 Windows Azure platform appliance

The Windows Azure platform can also be deployed as an appliance on third-party data centers and constitutes the cloud infrastructure governing the physical servers of the datacenter. The Windows Azure Platform Appliance includes Windows Azure, SQL Azure, and Microsoft-specified configuration of network, storage, and server hardware. The appliance is a solution that targets governments and service providers who want to have their own cloud computing infrastructure.

As introduced earlier, Azure already provides a development environment that allows building applications for Azure in their own premises. The local development environment is not intended to be production middleware, but it is designed for developing and testing the functionalities of applications that will eventually be deployed on Azure. The Azure appliance is instead a full-featured implementation of Windows Azure. Its goal is to replicate Azure on a third-party infrastructure and make available its services beyond the boundaries of the Microsoft Cloud. The appliance addresses two major scenarios: institutions that have very large computing needs (such as government agencies) and institutions that cannot afford to transfer their data outside their premises.

9.3.4 Observations

Windows Azure is Microsoft's solution for developing cloud computing applications. Azure is an implementation of the PaaS layer and provides the developer with a collection of services and scalable middleware hosted on Microsoft datacenters that address compute, storage, networking, and identity management needs of applications. The services Azure offers can be used either individually or all together for building both applications that integrate cloud features and elastic computing systems completely hosted in the cloud.

The core components of the platform are composed of compute services, storage services, and middleware. Compute services are based on the abstraction of roles, which identify a sandboxed environment where developers can build their distributed and scalable components. These roles are useful for Web applications, back-end processing, and virtual computing. Storage services include

solutions for static and dynamic content, which is organized in the form of tables with fewer constraints than those imposed by the relational model. These and other services are implemented and made available through AppFabric, which constitutes the distributed and scalable middleware of Azure.

SQL Azure is another important element of Windows Azure and provides support for relational data in the cloud. SQL Azure is an extension of the capabilities of SQL Server adapted for the cloud environment and designed for dynamic scaling.

The platform is mostly based on the .NET technology and Windows systems, even though other technologies and systems can be supported. For this reason, Azure constitutes the solution of choice for migrating to the cloud applications that are already based on the .NET technology.

SUMMARY

This chapter introduced some cloud platforms that are widely used in industry for building real commercial applications: Amazon Web Services, Google AppEngine, and Microsoft Windows Azure.

Amazon Web Services (AWS) provides solutions for building infrastructure in the Amazon Cloud. Amazon EC2 and Amazon S3 represent AWS's core value offering. The former allows developers to create virtual servers and customize their computing stack as required. The latter is a storage solution that allows users to store documents of any size. These core services are then complemented by a wide collection of services, covering networking, data management, content distribution, computing middleware, and communication, which make AWS a complete solution for developing entire cloud computing systems on top of the Amazon infrastructure.

Google AppEngine is a distributed and scalable platform for building Web applications in the Cloud. AppEngine is a scalable runtime that offers developers a collection of services for simplifying the development of Web applications. These services are designed with scalability in mind and constitute functional blocks that can be reused to define applications. Developers can build their applications in either Java or Python, first locally using the AppEngine SDK. Once the applications have been completed and fully tested, they can deploy the application on AppEngine.

Windows Azure is the cloud operating system deployed on Microsoft datacenters for building dynamically scalable applications. Azure's core components are represented by compute services expressed in terms of roles, storage services, and the AppFabric, the middleware that ties together all these services and constitutes the infrastructure of Azure. A role is a sandboxed runtime environment specialized for a specific development scenario: Web applications, background processing, and virtual computing. Developers define their Azure applications in terms of roles and then deploy these roles on Azure. Storage services represent a natural complement to roles. Besides storage for static data and semistructured data, Windows Azure also provides storage for relational data by means of the SQL Azure service.

AppEngine and Windows Azure are PaaS solutions. AWS extends its services across all three layers of the Cloud Computing Reference Model, although it is well known for its IaaS offerings, represented by EC2 and S3.

Review questions

1. What is AWS? What types of services does it provide?
2. Describe Amazon EC2 and its basic features.
3. What is a bucket? What type of storage does it provide?
4. What are the differences between Amazon SimpleDB and Amazon RDS?
5. What type of problems does the Amazon Virtual Private Cloud address?
6. Introduce and present the services provided by AWS to support connectivity among applications.
7. What is the Amazon CloudWatch?
8. What type of service is AppEngine?
9. Describe the core components of AppEngine.
10. What are the development technologies currently supported by AppEngine?
11. What is DataStore? What type of data can be stored in it?
12. Discuss the compute services offered by AppEngine.
13. What is Windows Azure?
14. Describe the architecture of Windows Azure.
15. What is a role? What types of roles can be used?
16. What is AppFabric, and which services does it provide?
17. Discuss the storage services provided by Windows Azure.
18. What is SQL Azure?
19. Illustrate the architecture of SQL Azure.
20. What is the Windows Azure Platform Appliance? For which kinds of scenarios was this appliance designed?