CHAPTER 2

Introduction to Cloud Computing

This chapter introduces the basic concepts of cloud computing, which provides scalable storage and processing services that can be used for extracting knowledge from big data repositories. Section 2.1 defines cloud computing and discusses the main service and deployment models adopted by cloud providers. The section also describes some cloud platforms that can be used to implement applications and frameworks for distributed data analysis. Section 2.2 discusses more specifically how cloud computing technologies can be used to implement distributed data analysis systems. The section identifies the main requirements that should be satisfied by a distributed data analysis system, and then discusses how a cloud platform can be used to fulfill such requirements.

2.1 CLOUD COMPUTING: DEFINITION, MODELS, AND ARCHITECTURES

As discussed in the previous chapter, an effective solution to extract useful knowledge from big data repositories in reasonable time is exploiting parallel and distributed data mining techniques. It is also necessary and helpful to work with data analysis environments allowing the effective and efficient access, management and mining of such repositories. For example, a scientist can use a data analysis environment to run complex data mining algorithms, validate models, and compare and share results with colleagues located worldwide.

In the past few years, clouds have emerged as effective computing platforms to face the challenge of extracting knowledge from big data repositories, as well as to provide effective and efficient data analysis environments to both researchers and companies. From a client perspective, the cloud is an abstraction for remote, infinitely scalable provisioning of computation and storage resources. From an implementation point of view, cloud systems are based on large sets of computing resources, located somewhere "in the cloud", which are allocated to applications on demand (Barga et al., 2011).

Thus, cloud computing can be defined as a distributed computing paradigm in which all the resources, dynamically scalable and often virtualized, are provided as services over the Internet. Virtualization is a software-based technique that implements the separation of physical computing infrastructures and allows creating various "virtual" computing resources on the same hardware. It is a basic technology that powers cloud computing by making possible to concurrently run different operating environments and multiple applications on the same server. Differently from other distributed computing paradigms, cloud users are not required to have knowledge of, expertise in, or control over the technology infrastructure in the "cloud" that supports them. A number of features define cloud applications, services, data, and infrastructure:

- *Remotely hosted*: Services and/or data are hosted on remote infrastructure.
- *Ubiquitous*: Services or data are available from anywhere.
- *Pay-per-use*: The result is a utility computing model similar to that of traditional utilities, like gas and electricity, where you pay for what you use.

We can also use the popular National Institute of Standards and Technology (NIST) definition of cloud computing to highlight its main features (Mell and Grance, 2011): "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". From the NIST definition, we can identify five essential characteristics of cloud computing systems, which are on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service.

Cloud systems can be classified on the basis of their *service model* (Software as a Service, Platform as a Service, Infrastructure as a Service) and their *deployment model* (public cloud, private cloud, hybrid cloud).

2.1.1 Service Models

Cloud computing vendors provide their services according to three main models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

Software as a Service defines a delivery model in which software and data are provided through Internet to customers as ready-to-use services. Specifically, software and associated data are hosted by providers, and customers access them without need to use any additional hardware or software. Moreover, customers normally pay a monthly/yearly fee, with no additional purchase of infrastructure or software licenses. Examples of common SaaS applications are Webmail systems (e.g., Gmail), calendars (Yahoo Calendar), document management (Microsoft Office 365), image manipulation (Photoshop Express), customer relationship management (Salesforce), and others.

In *Platform as a Service* model, cloud vendors deliver a computing platform typically including databases, application servers, development environment for building, testing, and running custom applications. Developers can just focus on deploying of applications since cloud providers are in charge of maintenance and optimization of the environment and underlying infrastructure. Hence, customers are helped in application development as they use a set of "environment" services that are modular and can be easily integrated. Normally, the applications are developed as ready-to-use SaaS. Google App Engine, Microsoft Azure, Salesforce.com are some examples of PaaS cloud environments.

Finally, *Infrastructure as a Service* is an outsourcing model under which customers rent resources like CPUs, disks, or more complex resources like virtualized servers or operating systems to support their operations (e.g., Amazon EC2, RackSpace Cloud). Users of an IaaS have normally skills on system and network administration, as they must deal with configuration, operation, and maintenance tasks. Compared to the PaaS approach, the IaaS model has a higher system administration costs for the user; on the other hand, IaaS allows a full customization of the execution environment. Developers can scale up or down its services adding or removing virtual machines, easily instantiable from virtual machine images.

Table 2.1 describes how the three service models satisfy the requirements of developers and final users, in terms of flexibility, scalability, portability, security, maintenance, and costs.

2.1.2 Deployment Models

Cloud computing services are delivered according to three main deployment models: public, private, or hybrid.

Table 2.1 How SaaS, PaaS, and IaaS Satisfy the Requirements of Developers and Final Users

Requirements	SaaS	PaaS	IaaS
Flexibility	Users can customize the application interface and control its behavior, but cannot decide which software and hardware components are used to support its execution.	Developers write, customize, test their application using libraries and supporting tools compatible with the platform. Users can choose what kind of virtual storage and compute resources are used for executing their application.	Developers have to build the servers that will host their applications, and configure operating system and software modules on top of such servers.
Scalability	The underlying computing and storage resources normally scale automatically to match application demand, so that users do not have to allocate resources manually. The result depends only on the level of elasticity provided by the cloud system.	Like the SaaS model, the underlying computing and storage resources normally scale automatically.	Developers can use new storage and compute resources, but their applications must be scalable and allow the dynamic inclusion of new resources.
Portability	There can be problems to move applications to other providers, since some software and tools could not work on different systems. For example, application data may be in a format that cannot be read by another provider.	Applications can be moved to another provider only if the new provider shares with the old one the required platform tools and services.	If a provider allows to download a virtual machine in a standard format, it may be moved to a different provider.
Security	Users can control only some security settings of their applications (e.g., using https instead of http to access some Web pages). Additional security layers (e.g., data replication) are hidden to the user and managed directly by the system.	The security of code and additional libraries used to build application is responsibility of the developer.	Developers must take care of security issues from operating system to application layer.
Maintenance	Users have not to carry maintenance tasks.	Developers are in charge of maintaining only their application; other software components and the hardware are maintained by the provider.	Developers are in charge of all software components, including the operating system; hardware is maintained by the provider.
Cost	Users typically pay a monthly/yearly fee for using the software, with no additional fee for the infrastructure.	Developers pay for the compute and storage resources, and for the licenses of libraries and tools used by their applications.	Developers pay for all the software and hardware resources used.

A *public cloud* provider delivers services to the general public through the Internet. The users of a public cloud have little or no control over the underlying technology infrastructure. In this model, services can be offered for free, or provided according to a pay-per-use policy. The main public providers, such as Google, Microsoft, Amazon, own and manage their proprietary data centers delivering services built on top of them.

A *private cloud* provider offers operations and functionalities "as a service", which are deployed over a company intranet or hosted in a remote data center. Often, small and medium-sized IT companies prefer this deployment model as it offers advanced security and data control solutions that are not available in the public cloud model.

Finally, a *hybrid cloud* is the composition of two or more (private or public) clouds that remain different entities but are linked together. Companies can extend their private clouds using other private clouds from partner companies, or public clouds. In particular, by extending the private infrastructure with public cloud resources, it is possible to satisfy peaks of requests, better serve user requests, and implement high-availability strategies.

Figure 2.1 depicts the general architecture of a public cloud and its main components, as outlined in (Li et al., 2010). Users access

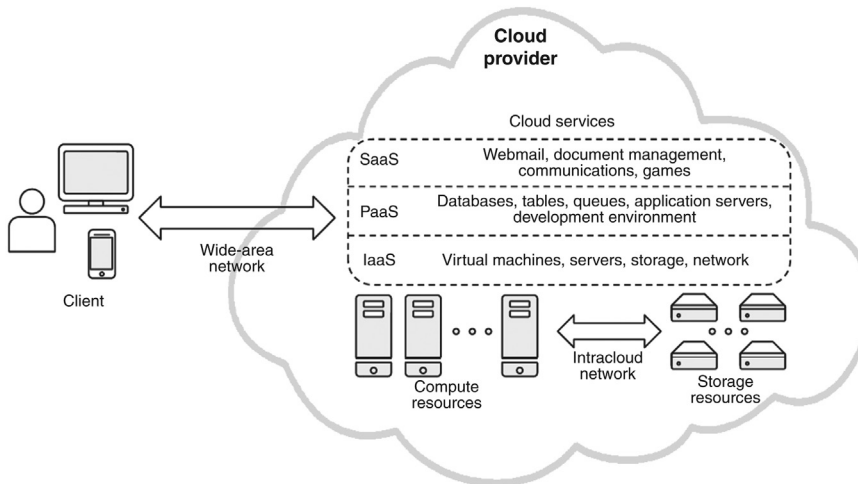

Fig. 2.1. General architecture of a public cloud.

cloud computing services using *client* devices, such as desktop computers, laptops, tablets, and smartphones. Through these devices, users access and interact with cloud-based services using a Web browser or desktop/mobile app. The business software and user's data are executed and stored on servers hosted in cloud data centers that provide *storage and compute resources*. Resources include thousands of servers and storage devices connected to each other through an *intracloud network*. The transfer of data between data center and users takes place on *wide-area network*.

Several technologies and standards are used by the different components of the architecture. For example, users can interact with cloud services through SOAP-based or RESTful Web services (Richardson and Ruby, 2007). *HTML5* and *Ajax* technologies allow Web interfaces to cloud services to have the look and interactivity equivalent to those of desktop applications. *Open Cloud Computing Interface* (OCCI)[1] specifies how cloud providers can deliver their compute, data, and network resources through a standardized interface. Another example is *Open Virtualization Format* (OVF)[2] for packaging and distributing virtual devices or software (e.g., virtual operating systems) to be run on virtual machines.

2.1.3 Cloud Environments

This section introduces four representative examples of cloud environments: *Microsoft Azure* as an example of public PaaS, *Amazon Web Services* as the most popular public IaaS, *OpenNebula* and *OpenStack* as examples of private IaaS. These environments can be used to implement applications and frameworks for data analysis in the cloud.

2.1.3.1 Microsoft Azure

Azure[3] is an environment and a set of cloud services that can be used to develop cloud-oriented applications, or to enhance existing applications with cloud-based capabilities. The platform provides on-demand compute and storage resources exploiting the computational and storage power of the Microsoft data centers. Azure is designed for supporting

[1]OCCI Working Group, http://www.occi-wg.org

[2]OVF Specification, http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_1.1.0.pdf

[3] Microsoft Azure, http://www.microsoft.com/azure

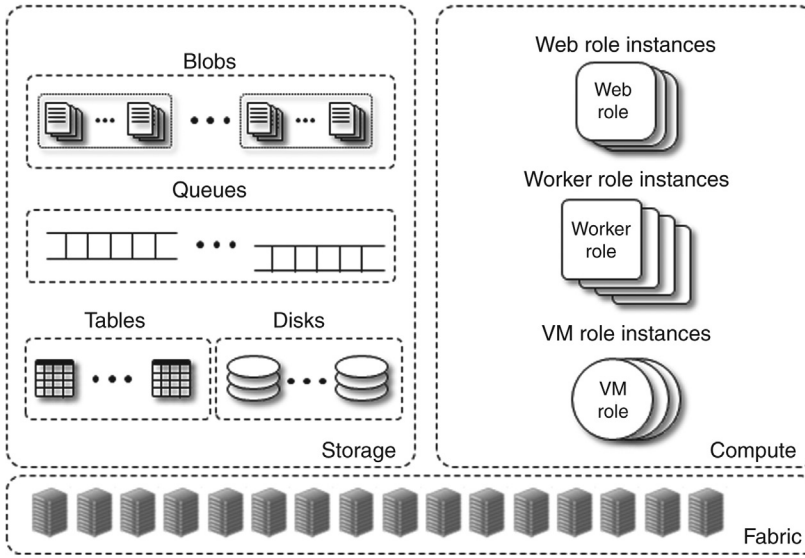

Fig. 2.2. Microsoft Azure.

high availability and dynamic scaling services that match user needs with a pay-per-use pricing model.

The Azure platform can be used to perform the storage of large datasets, execute large volumes of batch computations, and develop SaaS applications targeted towards end-users.

Microsoft Azure includes three basic components/services as shown in Figure 2.2:

- *Compute*: is the computational environment to execute cloud applications. Each application is structured into roles: *Web role*, for Web-based applications; *Worker role*, for batch applications; *VM role*, for virtual-machine images.
- *Storage*: provides scalable storage to manage binary and text data (*Blobs*), nonrelational tables (*Tables*), queues for asynchronous communication between components (*Queues*), and virtual disks (*Disks*).
- *Fabric controller*: whose aim is to build a network of interconnected nodes from the physical machines of a single data center. The Compute and Storage services are built on top of this component.

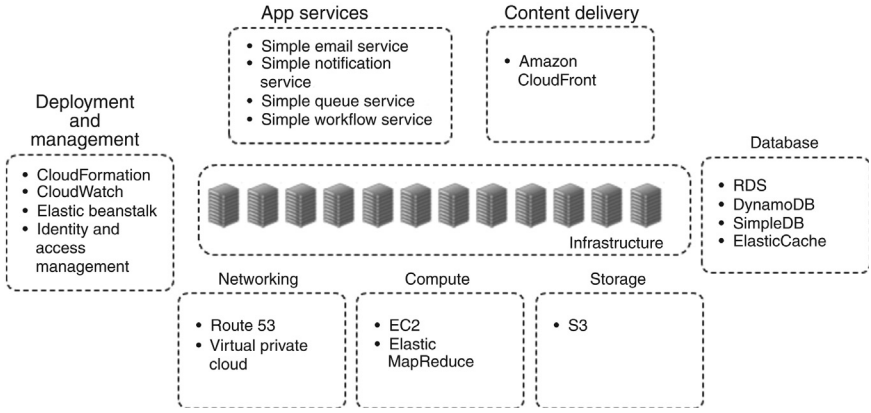

Fig. 2.3. Amazon Web Services.

Microsoft Azure provides standard interfaces that allow developers to interact with its services. Moreover, developers can use IDEs like Microsoft Visual Studio and Eclipse to easily design and publish Azure applications.

2.1.3.2 Amazon Web Services

Amazon offers compute and storage resources of its IT infrastructure to developers in the form of Web services. Amazon Web Services (AWS)[4] is a large set of cloud services that can be composed by users to build their SaaS applications or integrate traditional software with cloud capabilities (see Figure 2.3). It is simple to interact with these service since Amazon provides SDKs for the main programming languages and platforms (e.g., Java, .Net, PHP, Android).

AWS includes the following main services:

- Compute: *Elastic Compute Cloud* (EC2) allows creating and running virtual servers; *Amazon Elastic MapReduce* for building and executing MapReduce applications.
- Storage: *Simple Storage Service* (S3), which allows storing and retrieving data via the Internet.
- Database: *Relational Database Service* (RDS) for relational tables; *DynamoDB* for nonrelational tables; *SimpleDB* for managing small datasets; *ElasticCache* for caching data.

[4]Amazon Web Services, http://aws.amazon.com/

- Networking: *Route 53*, a DNS Web service; *Virtual Private Cloud* for implementing a virtual network.
- Deployment and Management: *CloudFormation* for creating a collection of ready-to-use virtual machines with preinstalled software (e.g., Web applications); *CloudWatch* for monitoring AWS resources; *Elastic Beanstalk* to deploy and execute custom applications written in Java, PHP and other languages; *Identity and Access Management* to securely control access to AWS services and resources.
- Content delivery: *Amazon CloudFront* makes easy to distribute content via a global network of edge locations.
- App services: *Simple Email Service* providing a basic email-sending service; *Simple Notification Service* to notify users; *Simple Queue Service* that implements a message queue; *Simple Workflow Service* to implement workflow-based applications.

Even though Amazon is best known to be the first IaaS provider (based on its EC2 and S3 services), it is now also a PaaS provider, with services like Elastic Beanstalk.

2.1.3.3 OpenNebula

OpenNebula (Sotomayor et al., 2009) is an open-source framework mainly used to build private and hybrid clouds. The main component of the OpenNebula architecture (see Figure 2.4) is the *Core*, which creates

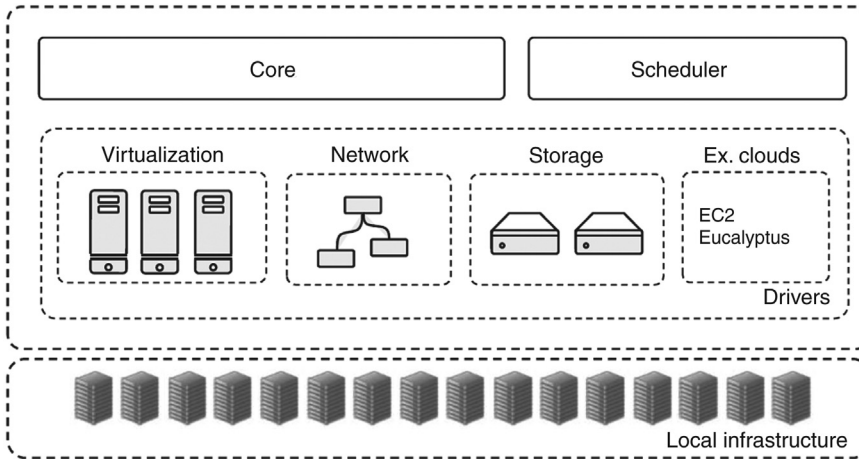

Fig. 2.4. OpenNebula.

and controls virtual machines by interconnecting them with a virtual network environment. Moreover, the Core interacts with specific storage, network, and virtualization operations through pluggable components called *Drivers*. In this way, OpenNebula is independent from the underlying infrastructure and offers a uniform management environment.

The Core also supports the deployment of *Services*, which are a set of linked components (e.g., Web server, database) executed on several virtual machines. Another component is the *Scheduler*, which is responsible for allocating the virtual machines on the physical servers. To this end, the Scheduler interacts with the Core component through appropriate deployment commands.

OpenNebula can implement a hybrid cloud using specific Drivers that allow to interact with external clouds. In this way, the local infrastructure can be supplemented with computing and storage resources from public clouds. Currently, OpenNebula includes drivers for using resources from Amazon EC2 and Eucalyptus (Nurmi et al., 2009), another open source cloud framework.

2.1.3.4 OpenStack

OpenStack[5] is a cloud operating system that allows the management of large pools of processing, storage, and networking resources in a datacenter through a Web-based interface. The system has been designed, developed and released following four open principles:

- *Open source*: OpenStack is released under the terms of the Apache License 2.0;
- *Open design*: Every six months there is a design summit to gather requirements and define new specifications for the upcoming release;
- *Open development*: A publicly available source code repository is maintained for the entire development process;
- *Open Community*: Most decisions are made by the OpenStack community using a lazy consensus model.

The modular architecture of OpenStack is composed by four main components, as shown in Figure 2.5.

[5]OpenStack, http://www.openstack.org/

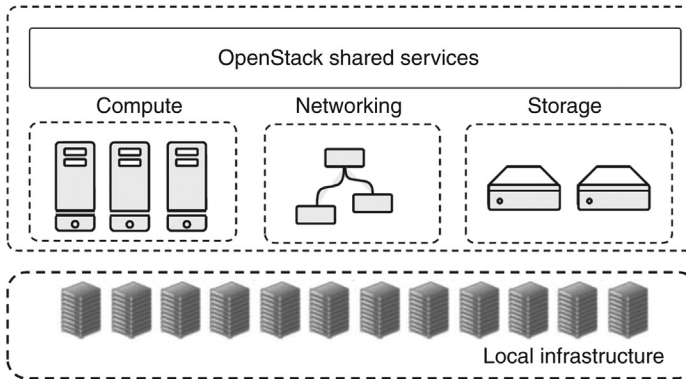

Fig. 2.5. OpenStack.

OpenStack *Compute* provides virtual servers upon demand by managing the pool of processing resources available in the datacenter. It supports different virtualization technologies (e.g., VMware, KVM) and is designed to scale horizontally. OpenStack *Storage* provides a scalable and redundant storage system. It supports Object Storage and Block Storage: the former allows storing and retrieving objects and files in the datacenter; the latter allows creating, attaching and detaching of block devices to servers. OpenStack *Networking* manages the networks and IP addresses. Finally, OpenStack *Shared Services* are additional services provided to ease the use of the datacenter. For instance, Identity Service maps users and services, Image Service manages server images, Database Service provides a relational database.

2.2 CLOUD COMPUTING SYSTEMS FOR DATA-INTENSIVE APPLICATIONS

Cloud systems can be effectively exploited to support data-intensive applications since they provide scalable storage and processing services, as well as software platforms for developing and running data analysis environments on top of such services. This section discusses how cloud computing technologies can be exploited to implement distributed data analysis systems for data-intensive KDD applications. We start identifying the main *functional* and *nonfunctional* requirements that should be satisfied by a distributed data analysis system for KDD applications.

Functional requirements specify which functionalities the system should provide; nonfunctional requirements refer to quality criteria mostly related to system performance.

2.2.1 Functional Requirements

The functional requirements that should be satisfied by a distributed data analysis system can be grouped into two main classes: *resource management* and *application management* requirements. The former refers to requirements related to the management of all the resources (data, tools, results) that may be involved in a KDD application; the latter refers to requirements related to the design and execution of the applications themselves.

2.2.1.1 Resource Management

Resources of interests in distributed KDD applications include *data sources*, *knowledge discovery tools*, and *knowledge discovery results*. Therefore, a distributed data analysis system should deal with the following resource management requirements:

- *Data management*: Data sources can be in different formats, such as relational databases, plain files, or semistructured documents (e.g., XML files). The system should provide mechanisms to store and access such data sources independently from their specific format. In addition, metadata formalisms should be defined and used to describe the relevant information associated with data sources (e.g., location, format, availability, available views), in order to enable their effective access and manipulation.
- *Tool management*: Knowledge discovery tools include algorithms and services for data selection, preprocessing, transformation, data mining, and results evaluation. The system should provide mechanisms to access and use such tools independently from their specific implementation. Metadata have to be used to describe the most important features of KDD tools (e.g., their function, location, usage).
- *Result management*: The knowledge obtained as the result of a knowledge discovery process is represented by a knowledge (or data mining) model. The system should provide mechanisms to store and access such models, independently from their structure and format. As for data and tools, data mining models need to be described by metadata to explain and interpret their content, and to enable their effective retrieval.

2.2.1.2 Application Management

A distributed data analysis system must provide effective mechanisms to design data-intensive KDD applications (*design management*) and control their execution (*execution management*):

- *Design management*: Distributed data analysis applications range from simple data mining tasks, to complex data mining patterns expressed as workflows. From a design perspective, three main classes of knowledge discovery applications can be identified: *single-task applications*, in which a single data mining task such as classification, clustering, or association rules discovery is performed on a given data source; *parameter sweeping applications*, in which a dataset is analyzed using multiple instances of the same data mining algorithm with different parameters; *workflow-based applications*, in which possibly complex knowledge discovery applications are specified as graphs that link together data sources, data mining algorithms, and visualization tools. A general system should provide environments to effectively design all the above-mentioned classes of data analysis applications.
- *Execution management*: The system has to provide a distributed execution environment that supports the efficient execution of data analysis applications designed by users. Since applications range from single tasks to complex knowledge discovery workflows, the execution environment should cope with such a variety of applications. In particular, the execution environment should provide the following functionalities, which are related to the different phases of application execution: accessing the data sources to be mined; allocating the needed compute resources; running the application based on the user specifications, which may be expressed as a workflow; presenting the results to the user. Additionally, the system should allow users to monitor the application execution.

2.2.2 Nonfunctional Requirements

Nonfunctional requirements can be defined at three levels: *user*, *architecture*, and *infrastructure*. User requirements specify how the user should interact with the system; architecture requirements specify which principles should inspire the design of the system architecture; finally, infrastructure requirements describe the nonfunctional features of the underlying computational infrastructure.

2.2.2.1 User Requirements

From a user point of view, the following nonfunctional requirements should be satisfied:

- *Usability*: The system should be easy to use by the end-users, without the need of undertaking any specialized training.
- *Ubiquitous access*: Users should be able to access the system from anywhere using standard network technologies (e.g., Web sites) either from a desktop PC or from a mobile device.
- *Data protection*: Data represents a key asset for users; therefore, the system should protect data to be mined and inferred knowledge from both unauthorized access and intentional/incidental losses.

2.2.2.2 Architecture Requirements

The main nonfunctional requirements at the architectural level are:

- *Service-orientation*: The architecture should be designed as a set of network-enabled software components (services) implementing the different operational capabilities of the system, to enable their effective reuse, composition, and interoperability.
- *Openness and extensibility*: The architecture should be open to the integration of new knowledge discovery tools and services. Moreover, existing services should be open for extension, but closed for modification, according to the open-closed principle.
- *Independence from infrastructure*: The architecture should be designed to be as independent as possible from the underlying infrastructure; in other terms, the system services should be able to exploit the basic functionalities provided by different infrastructures.

2.2.2.3 Infrastructure Requirements

Finally, from the infrastructure perspective, the following nonfunctional requirements should be satisfied:

- *Standardized access*: The infrastructure should expose its services using standard technologies (e.g., Web services), to make them usable as building blocks for high-level services or applications.
- *Heterogeneous/Distributed data support*: The infrastructure should be able to cope with very large and high dimensional datasets, stored in different formats in a single data center, or geographically distributed across many sites.

- *Availability*: The infrastructure should be in a functioning condition even in the presence of failures that affect a subset of the hardware/software resources. Thus, effective mechanisms (e.g., redundancy) should be implemented to ensure dependable access to sensitive resources such as user data and applications.
- *Scalability*: The infrastructure should be able to handle a growing workload (deriving from larger data to process or heavier algorithms to execute) in an efficient and effective way, by dynamically allocating the needed resources (processors, storage, network). Moreover, as soon as the workload decreases, the infrastructure should release the unneeded resources.
- *Efficiency*: The infrastructure should minimize resource consumption for a given task to execute. In the case of parallel/distributed tasks, efficient allocation of processing nodes should be guaranteed. Additionally, the infrastructure should be highly utilized so to provide efficient services.
- *Security*: The infrastructure should provide effective security mechanisms to ensure data protection, identity management, and privacy.

2.2.3 Cloud Models for Distributed Data Analysis

As discussed in Section 2.1.1, cloud providers classify their services into three main categories: Software as a Service (SaaS), where each software or application executed is provided through Internet to customers as ready-to-use services; *Platform as a Service* (PaaS), providing platform services such as databases, application servers, or environments for building, testing and running custom applications; *Infrastructure as a Service* (IaaS), providing computing resources like CPUs, memory, and storage, for running virtualized systems over the cloud.

Data analysis services for data-intensive KDD applications may be implemented within each of the three categories listed above:

- *KDD as SaaS*: where a single well-defined data mining algorithm or a ready-to-use knowledge discovery tool is provided as an Internet service to end-users, who may directly use it through a Web browser.
- *KDD as PaaS*: where a supporting platform is provided to developers that have to build their own applications or extend existing ones. Developers can just focus on the definition of their KDD applications

without worrying about the underlying infrastructure or distributed computation issues.

- *KDD as IaaS*: where a set of virtualized resources are provided to developers as a computing infrastructure to run their data mining applications or to implement their KDD systems from scratch.

In all three scenarios listed above, the cloud plays the role of infrastructure provider, even if at the SaaS and PaaS layers the infrastructure can be transparent to the end-user.

As an example of PaaS approach, Table 2.2 summarizes how the Microsoft Azure components and mechanisms, introduced in Section 2.1.3.1, can be effectively exploited to fulfill the functional requirements of a distributed data analysis system that have been introduced in Section 2.2.1.

Table 2.2 Using Microsoft Azure to Fulfill the Functional Requirements of a Distributed Data Analysis System

Functional Requirements		Microsoft Azure Components
Resource management	Data	*Different data formats*: Binary large objects (Blobs); nonrelational tables (Tables); queues for communication data (Queues); relational databases (SQL Database). *Metadata support*: Tables/SQL Databases to store data descriptions; custom description fields can be added to Blobs containing data sources.
	Tools	*Implementation-independent access*: Tools can be exposed as Web services. *Metadata support*: Tables/SQL Databases to store tools descriptions; custom description fields can be added to Blobs containing binary tools; WSDL descriptions for Web services.
	Results	*Models storing*: Blobs to store results either in textual or visual form. *Metadata support*: Tables/SQL Databases to describe models format; custom description fields can be added to Blobs containing data mining models.
Application management	Design	*Single-task applications*: Programming the execution of a single Web service or binary tool on a single Worker role instance. *Parameter sweeping applications*: Programming the concurrent execution of a set of Web services or binary tools on a set of Worker role instances. *Workflow-based applications*: Programming the coordinated execution of a set of Web services or binary tools on a set of Worker role instances.
	Execution	*Storage resources access*: Managed by the Storage layer. *Compute resources allocation*: Managed by the Compute layer. *Application execution and monitoring*: Web services/Worker role instances to run single tasks; Tables to store tasks information; Web role instance to present monitoring information. *Results presentation*: Blobs/Tables to store/interpret the inferred models; Web role instance to present results.

2.3 SUMMARY

Clouds provide scalable storage and processing services that can be used for extracting knowledge from big data repositories, as well as software platforms for developing and running data analysis environments on top of such services. In this chapter we provided an overview of cloud technologies by describing the main service models (Software as a Service, Platform as a Service, and Infrastructure as a Service) and deployment models (public, private, or hybrid clouds) adopted by cloud providers. We also described representative examples of cloud environments (Microsoft Azure, Amazon Web Services, OpenNebula, and OpenStack) that can be used to implement applications and frameworks for data analysis in the cloud. Finally, after having identified the main requirements that should be satisfied by a distributed data analysis system, we discussed as an example how the Microsoft Azure components and mechanisms can be used to fulfill such requirements.

REFERENCES

Barga, R., Gannon, D., Reed, D., 2011. The client and the cloud: democratizing research computing. IEEE Internet Comput. 15 (1), 72–75.

Li, A., Yang, X., Kandula, S., Zhang, M., 2010. CloudCmp: comparing public cloud providers. Tenth ACM SIGCOMM Conference on Internet Measurement (IMC'10), New York, USA.

Mell, P., Grance, T., 2011. The NIST Definition of Cloud Computing. NIST Special Publication 800-145.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D., 2009. The eucalyptus open-source cloud computing system. In: Proceedings of the Ninth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09), Washington, USA.

Richardson, L., Ruby, S., 2007. RESTful Web Services. O'Reilly & Associates, California.

Sotomayor, B., Montero, R.S., Llorente, I.M., Foster, I., 2009. Virtual infrastructure management in private and hybrid clouds. IEEE Internet Comput. 13, 14–22.