Cloud Services for Distributed Knowledge Discovery

Chapter · May 2016

DOI: 10.1002/9781118821930.ch51

3 authors:

Fabrizio Marozzo
Università della Calabria
60 PUBLICATIONS **483** CITATIONS
SEE PROFILE

Domenico Talia
Università della Calabria
407 PUBLICATIONS **5,113** CITATIONS
SEE PROFILE

Paolo Trunfio
Università della Calabria
139 PUBLICATIONS **1,941** CITATIONS
SEE PROFILE

51

Cloud Services for Distributed Knowledge Discovery

Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio

University of Calabria, Italy

51.1 Introduction

The information technology market has been moving from the demand and supply of products towards a service-oriented model in which all resources – processors, memories, data and applications – are provided as services to customers through the Internet. Such convergence between Internet technologies and services, combined with the use of virtualization techniques, has led to the development of the cloud computing paradigm (Mell and Grance, 2011).

In many application areas, knowledge discovery in databases (KDD) techniques are used to extract useful knowledge from large datasets. Very often, distributed KDD approaches must be used because datasets are too large to be analyzed in a single site, or because they are distributed across many locations and cannot be moved to a central site for processing. Several distributed KDD systems have been proposed so far. In most cases, those systems had to face infrastructure-level issues, such as resource allocation, execution management, fault tolerance, and so on (Talia and Trunfio, 2010).

This chapter discusses how cloud computing technologies can be exploited to implement a distributed KDD system without worrying about low-level aspects because they are already addressed by the cloud infrastructure. First, we discuss the requirements of a generic distributed KDD system, and how these requirements can be fulfilled by a cloud platform. Then, as a case study, we describe how we used a cloud platform to design and develop the *data mining cloud framework*, which supports the distributed execution of KDD applications modelled as workflows.

Encyclopedia of Cloud Computing, First Edition. Edited by San Murugesan and Irena Bojanova.

51.2 Requirements for a Distributed KDD System

In this section we identify the main requirements that should be satisfied by a generic distributed KDD system. The system requirements are divided into *functional* and *nonfunctional* requirements: the former specify which functionalities the system should provide; the latter include quality criteria mostly related to system performance.

51.2.1 Functional Requirements

The functional requirements that should be satisfied by a generic distributed KDD 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 knowledge-discovery application; the latter refers to requirements related to the design and execution of the applications themselves.

51.2.1.1 Resource Management

Resources of interests in distributed KDD applications include *data sources*, *knowledge discovery tools*, and *knowledge discovery results*. A distributed knowledge discovery system should therefore 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 of 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.

51.2.1.2 Application Management

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

- *Design management.* Distributed knowledge discovery 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 design all the classes of KDD applications mentioned above effectively.

- *Execution management.* The system has to provide a distributed execution environment that supports the efficient execution of knowledge discovery applications designed by users. As 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 user specifications, which may be expressed as a workflow; presenting the results to the user. The system should also allow users to monitor the application execution.

51.2.2 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.

51.2.2.1 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 to undertake 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.

51.2.2.2 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 words, the system services should be able to exploit the basic functionalities provided by different infrastructures.

51.2.2.3 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 datacenter, 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 when executing any given task. In the case of parallel / distributed tasks, efficient allocation of processing nodes should be guaranteed. The infrastructure should be used extensively to provide efficient services.
- *Security.* The infrastructure should provide effective security mechanisms to ensure data protection, identity management, and privacy.

51.3 Cloud for Distributed KDD

A key aspect of cloud computing is that end users do not need to have either knowledge or control over the infrastructure that supports their applications. In fact, cloud infrastructures are based on large sets of computing resources, located somewhere "in the Cloud," which are allocated to applications on demand. Cloud resources are provided in highly scalable way, i.e., they are allocated dynamically to applications depending of the current level of requests. Although similar in overall aims to grid systems, clouds are different because they hide the complexity of the underlying infrastructure, providing services ready to use where end users pay only for the resources effectively used (pay-per-use).

Cloud computing vendors classify their services into three categories: *software as a service* (SaaS), where each software or application executed is provided through Internet to customers as ready-to-use services (e.g., Google Calendar, Microsoft Hotmail, Yahoo Maps); *platform as a service* (PaaS), also known as cloud platform services, in which cloud providers offer platform services such as databases, application servers, or environments for building, testing and running custom applications (e.g., Google Apps Engine, Microsoft Azure, Force.com); *infrastructure as a service* (IasS), also known as cloud infrastructure services, which provides computing resources like CPUs, memory, and storage for running virtualized systems over the cloud (e.g., Amazon EC2, RackSpace Cloud).

Clouds can be exploited as effective infrastructures for handling knowledge discovery applications. In particular, KDD services 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 use it directly 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. In the following we briefly discuss Windows Azure as an example of a proprietary PaaS environment that can be effectively exploited to implement KDD systems and applications.

51.3.1 An example of PaaS: Windows Azure

Windows Azure 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 datacenters. Azure is designed for supporting 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. Windows Azure includes three basic components/services:

- *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), non-relational tables (Tables), queues for asynchronous communication between components (Queues), and NTFS volume (Drives).
- *Fabric controller* whose aim is to build a network of interconnected nodes from the physical machines of a single datacenter. The Compute and Storage services are built on top of this component.

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

Based on our study summarized in Table 51.1, the Azure components and mechanisms can be effectively exploited to fulfill the functional requirements of a generic distributed KDD system that have been introduced in section 51.2. We exploited these components and mechanisms to implement the data-mining cloud framework described in the next section.

51.4 Data Mining Cloud Framework

We worked to design a framework for supporting the scalable execution of knowledge discovery applications on top of cloud platforms. The framework has been designed to be implemented on different cloud systems. However, an implementation of this framework has been carried out using Windows Azure and has been evaluated through a set of data-analysis applications executed on a Microsoft Cloud datacenter.

The framework has been designed to support three classes of knowledge discovery applications: single-task applications, in which a single data-mining task is performed on a given dataset; parameter-sweeping applications, in which a dataset is analyzed by multiple instances of the same data-mining algorithm with different parameters; and workflow-based applications, in which knowledge discovery applications are specified as workflows.

51.4.1 System Architecture

The Data Mining Cloud Framework architecture includes different kinds of components that can be grouped into storage and compute components (see Figure 51.1).

The storage components include:

- A *Data Folder,* which contains data sources and the results of knowledge-discovery processes. Similarly, a *Tool Folder* contains libraries and executable files for data selection, pre-processing, transformation, data mining, and results evaluation.

Table 51.1 *How Azure components fulfill the functional requirements of a distributed KDD system*

KDD system requirements		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.

- *Data Table, Tool Table* and *Task Table* contain metadata information associated with data sources, tools, and tasks.
- The *Task Queue* contains the tasks ready to be executed.

The compute components are:

- A pool of *Worker instances*, which are in charge of executing the data-mining tasks submitted by users.
- A pool of *Web instances* host the *web site*, by allowing users to submit, monitor the execution, and access the results of their data-mining tasks.

The *web site* is the user interface to three functionalities: (i) *app. submission*, which allows users to submit single-task, parameter-sweeping, or workflow-based applications; (ii) *app. monitoring,* which is used to monitor the status and access results of the submitted applications; (*iii*) *data / tool management*, which allows users to manage input / output data and tools.

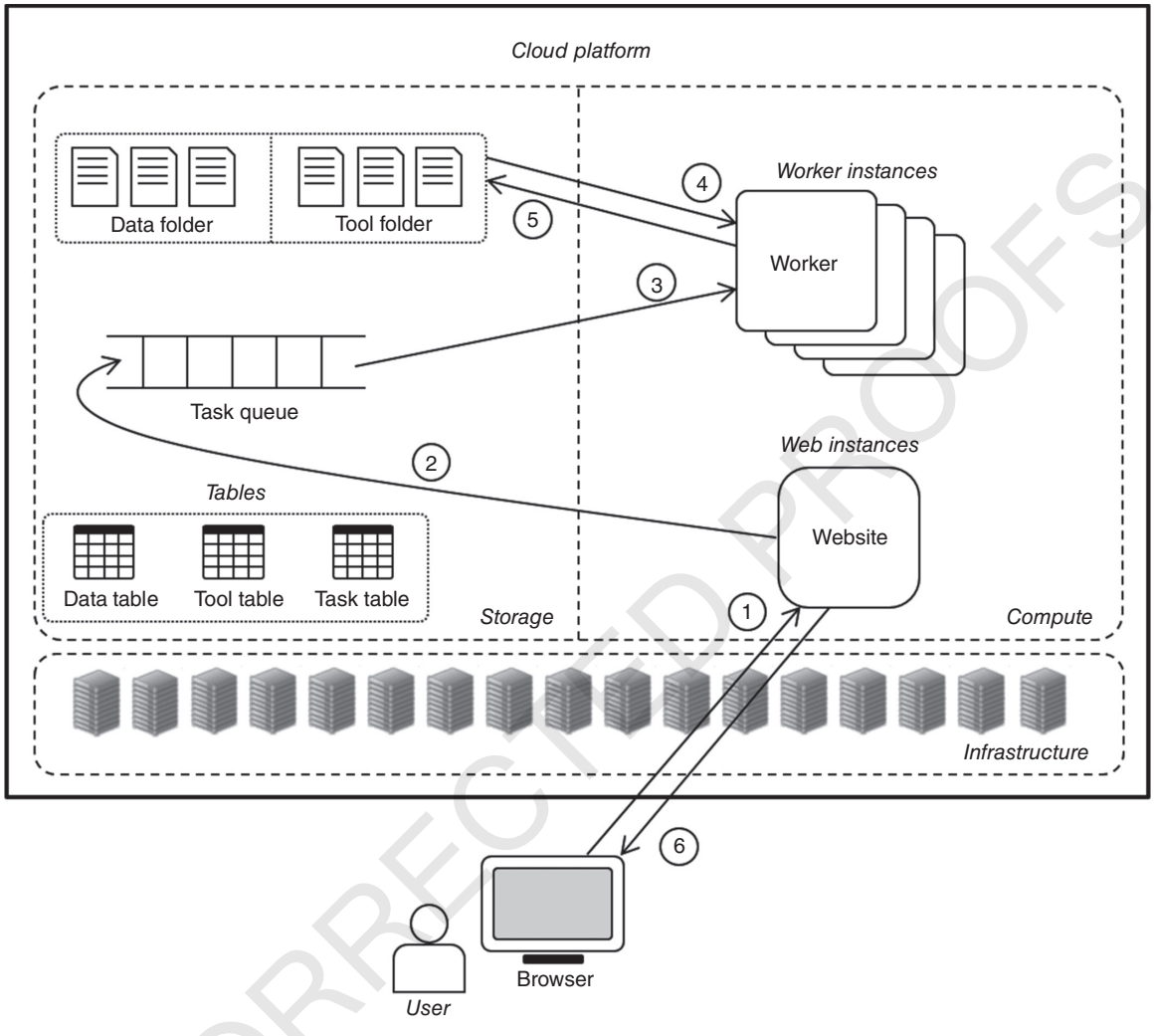

Figure 51.1 *System architecture and application execution steps*

51.4.2 Applications Execution

A user interacts with the system to perform the following steps for designing and executing a knowledge discovery application:

1. The user accesses the web site and designs the application (either single-task, parameter sweeping, or workflow-based) through a Web-based interface.
2. After application submission, the system creates a set of tasks and inserts them into the Task Queue on the basis of the application.
3. Each idle Worker picks a task from the Task Queue, and concurrently executes it.
4. Each Worker gets the input dataset from the location specified by the application. To this end, a file transfer is performed from the Data Folder where the dataset is located, to the local storage of the Worker.

5. After task completion, each Worker puts the result on the Data Folder.
6. The web site notifies the user as soon as her / his task(s) have completed, and allows her / him to access the results.

The set of tasks created in the second step depends on the type of application submitted by the user. In the case of a single-task application, just one data-mining task is inserted into the Task Queue. If the user submits a parameter sweeping application, the tasks corresponding to the combinations of the input parameters values are executed in parallel. In the case of a workflow-based application, the set of tasks created depends on how many data-mining tools are invoked within the workflow; initially, only the workflow tasks without dependencies are inserted into the Task Queue.

The Task Table is dynamically updated whenever the status of a task changes. The web site periodically reads and shows the content of this table, thus allowing users to monitor the status of their tasks.

Input data is temporarily staged on a server for local processing. To reduce the impact of data transfer on the overall execution time, it is important that input data are physically close to the virtual servers where the workers run on.

51.4.3 User Interface

The App submission section of the web site is composed of two main parts: one pane for composing and running both single-task and parameter-sweeping applications and another pane for programming and executing workflow-based knowledge discovery applications.

As an example, Figure 51.2 shows a screenshot of the App submission section, taken during the execution of a parameter-sweeping application. An application can be configured by selecting the algorithm to be executed, the dataset to be analyzed, and the relevant parameters for the algorithm. The system submits to the cloud a number of independent tasks that are executed concurrently on a set of virtual servers.

The user can monitor the status of each single task through the App monitoring section, as shown in Figure 51.3. For each task, the current status (submitted, running, done, or failed) and status update time are

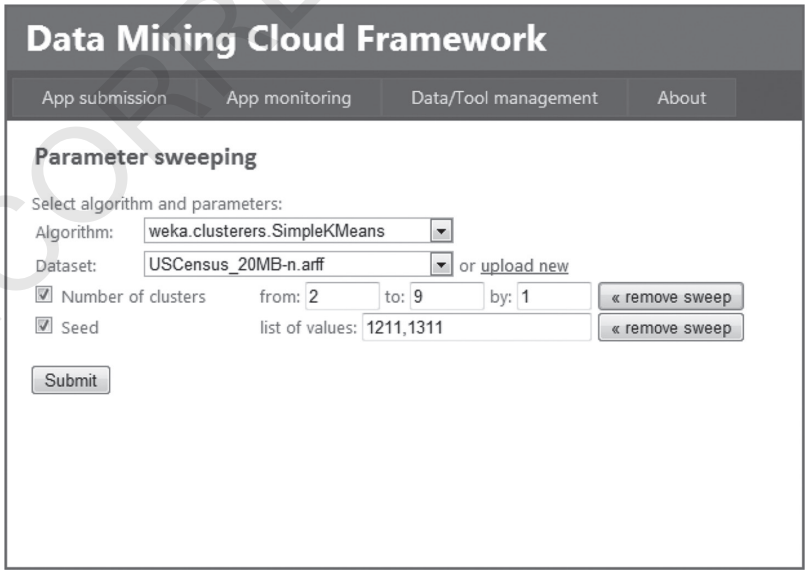

Figure 51.2 *Screenshot of the App submission section*

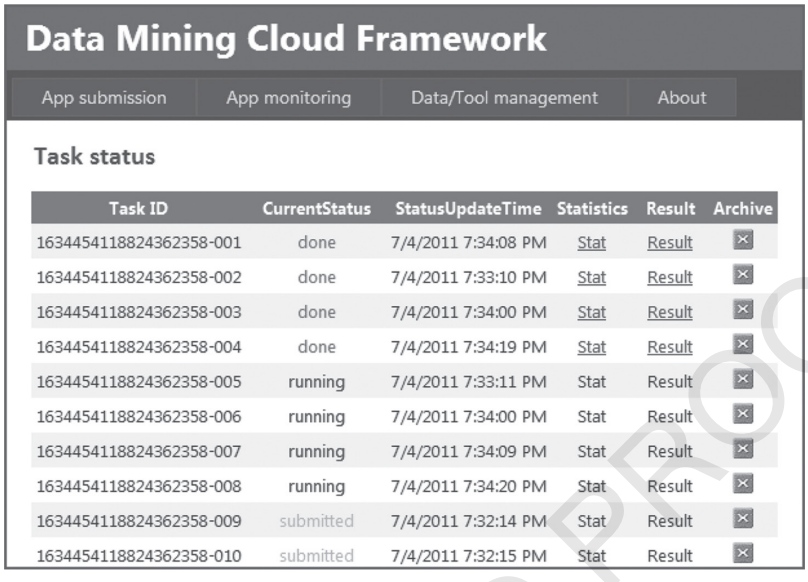

Figure 51.3 *Screenshot of the app monitoring section*

shown. Moreover, for each task that has completed its execution, two links are enabled: the first one (Stat) gives access to a file containing some statistics about the amount of resources consumed by the task; the second one (Result) visualizes the task result.

51.4.4 Workflow Programming

As mentioned above, the framework also includes the programming interface and its services to support the composition and execution of workflow-based knowledge discovery applications. Workflows support research and scientific processes by providing a paradigm that may encompass all the steps of discovery based on the execution of complex algorithms and the access and analysis of scientific data. In data-driven discovery processes, knowledge discovery workflows can produce results that can confirm real experiments or provide insights that cannot be achieved in laboratories.

Visual workflows in our framework are directed acyclic graphs whose nodes represent resources and whose edges represent the dependencies among the resources. Workflows include two types of nodes:

- *Data node*, which represents an input or output data element. Two subtypes exist: Dataset, which represents a data collection, and Model, which represents a model generated by a data analysis tool (e.g., a decision tree).
- *Tool node*, which represents a tool performing any kind of operation that can be applied to a data node (filtering, splitting, data mining, etc.).

The nodes can be connected with each other through direct edges, establishing specific dependency relationships among them. When an edge is being created between two nodes, a label is automatically attached to it representing the kind of relationship between the two nodes.

Data and Tool nodes can be added to the workflow singularly or in array form. A data array is an ordered collection of input / output data elements, while a tool array represents multiple instances of the same tool.

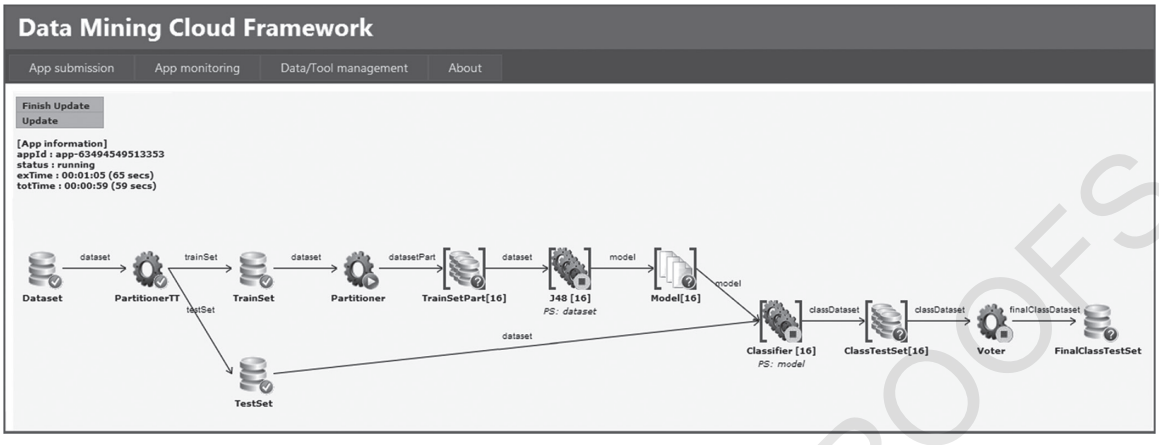

Figure 51.4 *The workflow during its execution*

Table 51.2 *Execution times and speedup of the application using up to 16 virtual machines*

No. of servers	125 MB dataset		250 MB dataset		500 MB dataset	
	Execution time	Speedup	Turnaround time	Speedup	Turnaround time	Speedup
1	00:29:49	1	01:18:07	1	03:16:38	1
2	00:16:03	1.86	00:39:25	1.98	01:44:03	1.89
4	00:08:25	3.54	00:19:54	3.93	00:53:04	3.92
8	00:04:57	6.02	00:12:45	6.13	00:30:47	6.76
16	00:03:05	9.67	00:07:10	10.9	00:19:02	10.9

Figure 51.4 shows a data-mining workflow composed of several sequential and parallel steps as an example for presenting the main features of the visual programming interface of the Data Mining Cloud Framework (Marozzo *et al.,* 2013a). The example workflow analyzes a dataset using 16 instances of the J48 classification algorithm provided by the Weka toolkit (Witten and Frank, 2000), which work on 16 partitions of the training set and generate the same number of knowledge models. By using the 16 generated models and the test set, 16 classifiers in parallel produce the same number of classified datasets. In the final step of the workflow, a voter generates the final dataset by assigning a class to each data item, choosing the class predicted by the majority of the models.

Table 51.2 presents execution times and speedup values achieved by using up to 16 virtual machines to execute the workflow on three datasets with size of 125, 250 and 500 MB. They are extracted from the KDD Cup 1999's dataset (Bache and Lichman, 2013).

We can observe that, in this example, the speedup achieved is satisfactory even if it does not increase linearly with the number of servers used because partitioning and voting tools run sequentially in the workflow. On the other hand, there are cases in which the inherent parallelism of applications can be fully exploited, bringing a linear speedup. For example, with a parameter-sweeping data-mining application on large data, discussed in Marozzo *et al.,* 2011), we achieved almost a linear speedup (14.6 on 16 virtual servers). Even when the speedup is not linear, the absolute amount of time saved can be significant when large datasets are analyzed. For instance, in the application whose results are presented Table 51.2, the execution time drops from more than 3 hours using 1 server, to less than 20 minutes using 16 servers. These results show the effectiveness of the proposed approach based on cloud resource exploitation for running data analysis applications.

51.4.5 Other Cloud Solutions for KDD Implementation

In the implementation mentioned before, the following mapping is used between Data Mining Cloud Framework's components and Azure's components: (i) Data Folder and Tool Folder are implemented as Azure's Blob containers; (ii) Data Table, Tool Table, Application Table, Task Table, and Users Table are implemented as Azure's nonrelational tables; (iii) the Task Queue is implemented as an Azure's Queue; (iv) Virtual Compute Servers are implemented as Azure's Worker Role instances; (v) Virtual Web Servers are implemented as Azure's Web Role instances.

Even though the current implementation of framework is based on Azure, it has been designed to abstract from specific Cloud platforms. It can therefore be implemented using other cloud systems, like, for instance, the popular Amazon Web Services (AWS) or the Google App Engine. In particular, AWS offers compute and storage resources in the form of Web services, which comprise compute services, storage services, database services, and app services. Compute services include Elastic Compute Cloud (EC2) for creating and running virtual servers and Amazon Elastic MapReduce for building and executing MapReduce applications (Dean and Ghemawat, 2008). Storage services include Simple Storage Service (S3) for storing and retrieving data via the Internet. Database services include Relational Database Service (RDS) for relational tables and DynamoDB for nonrelational tables. App services include, among others, Simple Queue Service that implements a queue for application-level messages. If AWS is considered as a target Cloud platform, the Data Mining Cloud Framework's components can be implemented on the AWS's components as follows: (i) Data Folder and Tool Folder could be stored on S3; (ii) Data Table, Tool Table, Application Table, Task Table, and Users Table could be implemented as non-relational tables using DynamoDB; (iii) the Task Queue could be implemented using the Simple Queue Service; (iv) Virtual Compute Servers and Virtual Web Servers could be created on top of EC2.

Furthermore, other than on well known public cloud platforms, the framework can be implemented on the top of a private IaaS system by using open source cloud frameworks such as OpenStack. In this case, each component could be implemented using a software library/application deployed on a virtual machine executed by the IaaS system. According to this approach, the Data Mining Cloud Framework components can be implemented as follows: (i) Data Folder and Tool Folder could be implemented as FTP servers (e.g., Filezilla); (ii) Data Table, Tool Table, Application Table, Task Table, and Users Table could be implemented as nonrelational tables (e.g., MongoDB); (iii) Task Queue could be implemented using a message-oriented middleware (e.g., Java Message Service); (iv) Virtual Compute Servers could be implemented as batch applications (e.g., Java applications); (v) Virtual Web Servers could be implemented as Web servers (e.g., Apache / Tomcat).

51.5 Conclusion

In this chapter we discussed how cloud computing technologies can be exploited to implement a service-oriented distributed KDD system. Starting from the requirements of a generic distributed KDD system, we discussed how these requirements can be fulfilled by a cloud platform. As a case study, we described the Data Mining Cloud Framework, a system that supports the distributed execution of KDD applications. The user interface is very simple and hides the complexity of the Cloud infrastructure used to run applications.

The performance of the Data Mining Cloud Framework have been evaluated through the execution of data-mining applications on a pool of virtual servers hosted by a Microsoft Cloud datacenter. The experiments demonstrated the effectiveness of the Data Mining Cloud Framework, as well as the scalability that can be achieved through the parallel execution of parameter sweeping data-mining applications on a pool of virtual servers.

Current work is aimed at supporting the design and execution of script-based data analysis workflows on clouds. In Marozzo *et al.* (2013b), we introduced a workflow language, named JS4Cloud, that extends JavaScript to support the implementation of cloud-based data analysis tasks and the handling of data on the cloud. We also demonstrated how data analysis workflows programmed through JS4Cloud can be processed by the Data Mining Cloud Framework to make parallelism explicit and to enable their scalable execution on clouds.

References

Bache, K. and Lichman, M. (2013) *UCI Machine Learning Repository*, University of California, School of Information and Computer Science, Irvine, CA, http://archive.ics.uci.edu/ml (accessed January 5, 2016).

Dean, J. and Ghemawat, S. (2008) MapReduce: Simplified data processing on large clusters. *Communications of the ACM* **51**(1), 107–113.

Marozzo, F., Talia, D. and Trunfio, P. (2011) *A Cloud Framework for Parameter Sweeping Data Mining Applications*. Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CLOUDCOM'11). IEEE Computer Society, Washington, DC, pp. 367–374.

Marozzo, F., Talia, D., and Trunfio, P. (2013a) *Scalable Script-Based Data Analysis Workflows on Clouds*. Proceedings of the Eighth Workshop on Workflows in Support of Large-Scale Science (WORKS'13). ACM, New York, NY, pp. 124–133.

Marozzo, F., Talia, D., and Trunfio, P. (2013b) A cloud framework for big data analytics workflows on Azure, in *Clouds, Grids and Big Data* (ed. L. Grandinetti), IOS Press, Amsterdam.

Mell, P. M. and Grance, T. (2011) *The NIST Definition of Cloud Computing*. Special Publication 800-145. NIST, Gaithersburg, MD, http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909616 (accessed November 25, 2015).

Talia, D. and Trunfio, P. (2010) How distributed data mining tasks can thrive as knowledge services. *Communications of the ACM* **53**(7), 132–137.

Witten, I. H. and Frank, E. (2000) *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, San Francisco, CA.