A Workflow Management System for Scalable Data Mining on Clouds

Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio

Abstract—The extraction of useful information from data is often a complex process that can be conveniently modeled as a data analysis workflow. When very large data sets must be analyzed and/or complex data mining algorithms must be executed, data analysis workflows may take very long times to complete their execution. Therefore, efficient systems are required for the scalable execution of data analysis workflows, by exploiting the computing services of the Cloud platforms where data is increasingly being stored. The objective of the paper is to demonstrate how Cloud software technologies can be integrated to implement an effective environment for designing and executing scalable data analysis workflows. We describe the design and implementation of the Data Mining Cloud Framework (DMCF), a data analysis system that integrates a visual workflow language and a parallel runtime with the Software-as-a-Service (SaaS) model. DMCF was designed taking into account the needs of real data mining applications, with the goal of simplifying the development of data mining applications compared to generic workflow management systems that are not specifically designed for this domain. The result is a high-level environment that, through an integrated visual workflow language, minimizes the programming effort, making easier to domain experts the use of common patterns specifically designed for the development and the parallel execution of data mining applications. The DMCF's visual workflow language, system architecture and runtime mechanisms are presented. We also discuss several data mining workflows developed with DMCF and the scalability obtained executing such workflows on a public Cloud.

Index Terms—Workflows, Data analysis, Cloud computing, Software-as-a-Service, Scalability.

1 Introduction

The past two decades have been characterized by an exponential growth of digital data production in many fields of human activities, from science to enterprise. Very large datasets are produced daily from sensors, instruments, mobile devices and computers, and are often stored in distributed repositories. For example, astronomers analyze large image data that every day comes from telescopes and artificial satellites [1]; physicists must study the huge amount of data generated by particle accelerators to understand the laws of Universe [2]; medical doctors and biologists collect huge amount of information about patients to search and try to understand the causes of diseases [3]; sociologists analyze large social networks to find how users are influenced by others for various reasons [4]. Such few examples demonstrate how the exploration and automated analysis of large datasets powered by computing capabilities are fundamental to advance our knowledge in many fields [5].

Unfortunately, large datasets are hard to understand, and in particular models and patterns hidden in them cannot be comprehended neither by humans directly, nor by traditional analysis methodologies. To cope with big data repositories, parallel and distributed data analysis techniques must be used. It is also necessary and helpful to work with data analysis tools and frameworks allowing the effective and efficient access, management and mining of such repositories. In fact, often scientists and professionals use data analysis environments to execute complex simulations, validate models, compare and share results with colleagues located world-wide [6].

Extracting useful information from data is often a complex process that can be conveniently modeled as a data analysis workflow combining distributed datasets, preprocessing tools, data mining algorithms and knowledge models. Workflows provide a declarative way of specifying the high-level logic of an application, hiding the low-level details that are not fundamental for application design. They are also able to integrate existing software modules, datasets, and services in complex compositions implementing discovery processes in scientific and business applications. Cloud systems can be effectively used to handle data analysis workflows since they provide scalable processing and storage services, together with software platforms for developing data analysis environment on top of such services [7].

The objective of the paper is to demonstrate how Cloud software technologies can be integrated to implement an effective programming environment and an efficient runtime system for designing and executing scalable data analysis workflows. Specifically, the paper describes design and implementation of the Data Mining Cloud Framework (DMCF), a system that integrates a visual workflow language and a parallel runtime with the Software-as-a-Service (SaaS) model for enabling the scalable execution of complex data analysis workflows on Clouds. The main contri-

• *The authors are with DIMES, University of Calabria, Italy. E-mail: {fmarozzo,talia,trunfio}@dimes.unical.it*

bution of DMCF is the integration of different hardware/software solutions for high-level programming, management and execution of parallel data mining workflows.

Through its visual programming model, DMCF minimizes the programming effort, making easier to domain experts the use of common patterns specifically designed for the development and the parallel execution of data mining applications. This was done by introducing a visual workflow language, called VL4Cloud, which includes visual patterns useful in real data mining applications, in particular: *data preprocessing* (data partitioning and filtering); *parameter sweeping* (the concurrent execution of many instances of the same tool with different parameters to find the best result); *input sweeping* (the concurrent execution of many instances of the same tool with different input data); *tool sweeping* (the concurrent execution of different tools on same data); combinations of parameter, input, and tool sweeping patterns for the highest flexibility; *data and models aggregation* (e.g., models evaluations, voting operations, models aggregation). For supporting these patterns, VL4Cloud provides novel data-mining-specific visual workflow formalisms, data and tool arrays, which significantly ease the design of parallel data analysis workflows. A data array allows representing an ordered collection of input/output data sources in a single workflow node, while a tool array represents multiple instances of the same tool. Thanks to data and tool arrays, workflows are more compact compared to those designed using other visual formalisms that oblige developers to replicate node chains to obtain the same semantic.

The DMCF runtime was designed to enable the parallel execution of data analysis workflows on multiple Cloud machines, so as to improve performance and ensure scalability of applications. To this end, the runtime implements data-driven task parallelism that automatically spawns ready-to-run workflow tasks to the Cloud resources, taking into account dependencies among tasks and current availability of data to be processed. Parallelism is effectively supported by the data and tool array formalisms of VL4Cloud, because the array cardinality automatically determines the parallelism degree at runtime. In addition, data and tool arrays improve generality of workflows and therefore their reusability. In fact, once defined, a workflow can be instantiated many times not only changing the input data or the tools (as in the other workflow formalisms), but also redefining the parallelism level by specifying a different cardinality of the data/tool arrays.

Finally, DMCF is provided according with the SaaS model. This means that no installation is required on the user's machine: the DMCF visual user interface works in any modern Web browser, and so it can be run from most devices, including desktop PCs, laptops, and tablets. This is a key feature for users who need ubiquitous and seamless access to scalable data analysis services, without needing to cope with installation and system management issues. Thanks to the SaaS approach, user have online access to a large repository of ready-to-use data handling and mining algorithms. Many of such algorithms are taken from open source projects (more than 100 algorithms from the Weka and Waffles libraries) and several of them were designed by scratch (e.g., tools for data splitting, data merging, voting). In addition, it is easy for every user to add his/her own algorithm in the system using a visual configuration tool. Through a guided procedure, the configuration tool allows users to upload executable files of the new algorithm, and to specify its input and output parameters.

DMCF has been used to implement data analysis workflows in a variety of domains. Two significant examples are an association rule analysis between genome variations and clinical conditions of a group of patients [8] and a trajectory mining analysis for discovering patterns from vehicles trajectory data in a city [9]. In these two cases, the use of DMCF allowed the development of data-driven workflows in a direct and effective way, showing good scalability on large datasets.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 describes system model, general architecture and execution mechanisms. Section 4 describes how the framework has been implemented on top of the Microsoft Azure Cloud platform. Section 5 describes how the framework allows users to design and execute workflow-based data analysis applications. Section 6 presents scalability results of data analysis applications designed and executed using the DMCF framework. Finally, Section 7 concludes the paper.

2 Related work

Several systems have been proposed to design and execute workflow-based applications [10] [11] [12], but only some of them currently work on the Cloud and support visual workflow programming. In the following we discuss representative visual workflow management systems that can be used in Cloud environments.

Galaxy [13] is a web-based platform for developing genomic science applications, now used as a general bioinformatics workflow management system. A Galaxy workflow is a reusable template that a user can run repeatedly on different data. The Galaxy [13] software runs on Linux/Unix based servers, and therefore several organizations execute Galaxy on private or public Cloud IaaS. In order to improve Galaxy's capabilities with respect to interfacing with large scale computational systems and running workflows in a parallel manner, Galaxy has recently been integrated with Swift/T [14], a large-scale parallel programming framework discussed below.

Taverna [15] is a workflow management system mostly used in the life sciences community. Taverna can orchestrate Web Services and these may be running in the Cloud, but this is transparent for Taverna, as demonstrated in the BioVel project. Recently, the Tavaxy system has been developed for allowing the integration of Taverna and Galaxy [16]. In particular, Tavaxy allows users to create and execute workflows employing an extensible set of workflow patterns that enables the re-use and integration of existing workflows from Taverna and Galaxy, and allows the creation of hybrid workflows.

Orange4WS [17] is a service-oriented workflow system that extends Orange, a data mining toolbox and a visual programming environment for the visual composition of data mining workflows. The system enables the orchestration of web-based data mining services and the collection of information in various formats, as well as design of repeatable data mining workflows used in bioinformatics and e-science applications.

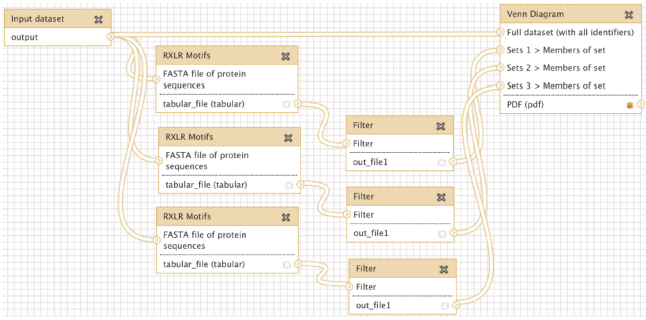
Kepler [18] is a visual workflow management system that provides a graphical user interface for designing scientific workflows. Data is encapsulated in messages or tokens, and transferred between tasks through input and output ports. Kepler provides an assortment of built-in components with a major focus on statistical analysis and supports task parallel execution of workflows using multiple threads on a single machine.

E-Science Central (e-SC) [19] allows scientists to store, analyze and share data in the Cloud. Its in-browser workflow editor allows users to design a workflow by connecting services, either uploaded by themselves or shared by other users of the system. One of the most common use cases for e-Sc is to provide a data analysis back end to a standalone desktop or Web application. In the current implementation, all the workflow services within a single invocation of a workflow execute on the same Cloud node.

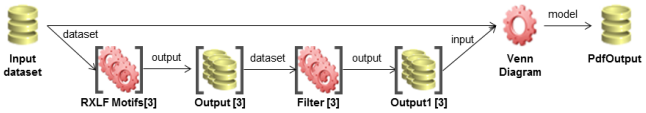
ClowdFlows [20] is a Cloud-based platform for the composition, execution, and sharing of interactive data mining workflows. Its service-oriented architecture allows using third-party services (e.g., Web services wrapping open-source or custom data mining algorithms). The server side consists of methods for the workflow editor to define workflows, and a relational database of workflows and data.

Pegasus [21] includes a set of technologies to execute workflow-based applications over clusters and Grids. The system can manage the execution of an application formalized as a visual workflow by mapping it onto available resources. Recent research activities carried out on Pegasus investigated the system implementation on Cloud platforms [22].

ASKALON [23], is a Cloud-based application development environment designed as a distributed service-oriented architecture. Users can compose



(a) Galaxy version (without arrays).



(b) DMCF version (with arrays).

Fig. 1. An example of data analysis workflow designed with and without arrays.

workflow using a UML graphical modeling tool. The abstract workflow representation is given to the middleware services for transparent execution onto the Clouds. ASKALON includes components for automatic image management, software deployments, and authenticating with multiple Cloud providers.

WS-PGRADE [24] allows users to define workflows through a graphical interface and to execute them on different distributed computing infrastructures. End-users can use the system through a simplified interface where they can download a workflow from a repository, configure its parameter, and launch and monitor its execution on different distributed computing infrastructures, including Amazon EC2.

YAWL (Yet Another Workflow Language) [25] is a modeling language for workflows based on the Petri Nets formalism, enriched with dedicated constructs to deal with multiple instance patterns. The language is supported by a framework that includes an execution engine, a graphical editor and a worklist handler. Recently, the YAWL framework has been extended with an ad-hoc load balancer for the distribution of workload over multiple YAWL engines, so as to be used in the cloud [26].

RapidMiner[1] is a powerful commercial platform through which users can exploit many analytics tools to visually create predictive analytics workflows. The system provides full integration with Hadoop technologies and MapReduce programming [27]. In fact, each tool node of a workflow can be a Hadoop program, and therefore it can exploit the Hadoop parallelism. However, differently from DMCF, the visual elements in RapidMiner do not allow developers to control and show the parallelism level of each single tool node and do not permit to configure concurrency degree at a fine grain.

1. http://rapidminer.com/

Swift/T [28] is a parallel scripting language that runs workflows across distributed systems like clusters and Clouds. It provides a script-based language that models workflows as a set of program invocations with associated command-line arguments, input and output files. The runtime comprises a set of services that implement the parallel execution of Swift/T scripts taking into account data dependencies and external resources availability. Galaxy can be used as a visual interface to Swift/T according to three integration schemes [14]: i) A Swift script is executed as a Galaxy workflow node; the overall workflow remains sequential, as only the Swift node is executed in parallel. ii) A Galaxy workflow is translated into a Swift script; the Swift engine can execute parallel paths concurrently, but the original nodes in the Galaxy workflow are not parallel. iii) The Swift "foreach" pattern is integrated in Galaxy; this is equivalent to DMCF's input sweeping, but does not support parameter and tool sweeping. To summarize, the experience reported in [14] shows that the a posteriori integration of a visual interface and a script-based language limits the exploitation of parallelism. Building a visual interface on top of an already available script-based parallel framework (e.g., Galaxy+Swift/T) can lead to the loss of important features, such as full support to sweeping patterns and parallelism exploitation at nodes and paths level.

Other systems, broadly related to DMCF, do not support visual workflow programming, as they employ script-based languages (e.g., Java in COMPSs [29]) or data representation models (e.g., XML in SciCumulus [30]), while others focus on workflow scheduling optimization [31] [32], which however is not the main goal of our work.

TABLE 1
Comparison of Cloud-based visual workflow systems.

System	Parallelism level	SaaS	Underl. Cloud	Visual Data/tool arrays
Galaxy	workflow, task	Yes	IaaS	No
Taverna	workflow, task	No	-	No
Tavaxy	workflow, task	Yes	IaaS	No
Orange4WS	workflow, task	No	-	No
Kepler	workflow, task	No	-	No
E-Science Cen.	workflow	Yes	IaaS	No
ClowdFlows	workflow	Yes	IaaS	No
Pegasus	workflow, task	Yes	IaaS	No
ASKALON	workflow, task	No	-	No
WS-PGRADE	workflow, task	Yes	IaaS	No
YAWL	workflow, task	No	IaaS	Tool arr.
RapidMiner	workflow, task	Yes	IaaS	No
Swift/T+Galaxy	workflow, task	Yes	IaaS	No
DMCF	*workflow, task*	*Yes*	*PaaS*	*Yes*

Table 1 compares related Cloud-based visual workflow management systems with DMCF (last row in the table). For each system, the table indicates: (i) the parallelism level it provides; (ii) whether or not it is provided as a Software as a Service (SaaS); (iii) the service model of the underlying Cloud; (iv) whether or not it supports visual data and tool arrays.

For comparison purposes, we distinguish two types of parallelism levels: *workflow parallelism*, which refers to the ability of executing multiple workflows concurrently; *task parallelism*, which is the ability of executing multiple tasks of the same workflow concurrently. Most systems support both workflow and task parallelisms, except for ClowdFlows and E-Science Central that focus on workflow parallelism only.

The second column shows which systems are provided according with the SaaS model. The SaaS model is implemented by Galaxy, Tavaxy, E-Science Central, ClowdFlows, Pegasus, WS-PGRADE, RapidMiner, Swift/T+Galaxy and DMCF, whereas Taverna, Orange4WS, Kepler, ASKALON and YAWL are implemented as desktop applications that can invoke Cloud software exposed as Web Services.

The third column, which applies to the system provided as SaaS, indicates the service model of the underlying Cloud platform or infrastructure. All the SaaS systems are implemented on top of Infrastructure-as-a-Service (IaaS) Clouds, except for DMCF that is designed to run on top of Platform-as-a-Service (PaaS) Clouds.

The last column, shows that DMCF is the only system supporting both visual data and tool arrays. As mentioned earlier, visual data and tool array formalisms allow users to design parallel data mining workflows in a more compact way, and are effective to fork the concurrent execution of many parallel tasks to Cloud resources, thus improving scalability.

In order to show the advantages of using data and tool arrays compared to visual workflow languages that do not support this feature, Figure 1 shows a real workflow taken from [33] designed using both Galaxy (as an example of system that does not support visual arrays) and DMCF. The workflow analyzes in parallel a dataset using three instances of the RXLR Motif tool, where each instance is run with different parameters. The results of the three RXLR Motif instances are cleaned by three filtering tools. Then, the filtered outputs are joint and presented to the user through a Venn diagram.

By comparing the Galaxy version (Figure 1(a)) with the DMCF version (Figure 1(b)), it is evident DMCF workflows are more compact compared to those designed using other visual formalisms that oblige developers to replicate node chains to obtain the same semantic. In fact, imagine that a user wants to run a higher number of RXLR Motifs tools (e.g., 20) on the same data: using Galaxy, the number of workflow's nodes and the space required to design it would increase proportionally with the number of tools; using DMCF, the workflow remains graphically identical, because only the size of data and tool arrays needs to be changed. Moreover, arrays allow implementing

task parallelism more intuitively, because the array cardinality automatically determines the parallelism degree at runtime.

Data and tool arrays also improve generality of workflows and therefore their reusability. In fact, once defined, a workflow can be instantiated many times not only changing the input data or the tools (as in the other workflow formalisms), but also the parallelism level by specifying a different cardinality of the data/tool arrays. Finally, thanks to data and tool arrays, we are able to easily implement workflow patterns that are very useful in data mining applications (parameter sweeping, data sweeping and data aggregation) as discussed in Section 5.1.

To summarize, Table 1 shows that DMCF is the only SaaS system featuring both workflow/task parallelisms and support to visual data and tool arrays. Furthermore, DMCF is the only system designed to run on top of a PaaS. A key advantage of this approach is the independence from the infrastructure layer. In fact, the DMCF's components are mapped into PaaS services, which in turn are implemented on infrastructure components. Changes to the Cloud infrastructure affect only the infrastructure/platform interface, which is managed by the Cloud provider, and therefore DMCF's implementation and functionality are not influenced. In addition, the PaaS approach facilitates the implementation of the system on a public Cloud, which free final users and organizations from any hardware and OS management duties.

3 Data Mining Cloud Framework

This section provides a conceptual description of the Data Mining Cloud Framework that is independent from specific Cloud implementations.

3.1 System model

The model introduced here provides an abstraction to describe the characteristics of applications as they are seen in our system.

A *workflow* instance is modeled as a tuple:

$$workflow = \langle workflowId, userId, workflowStatus, taskList \rangle$$

where *workflowId* is the workflow identifier, *userId* is the identifier of the user who submitted the workflow, *workflowStatus* represents the status of the workflow (new, ready, running, done, or failed), and *taskList* contains the tasks that form the workflow. For brevity, in the remainder of the paper we will use the term "workflow" in place of "workflow instance".

A *task* is modelled as a tuple:

$$task = \langle taskId, workflowId, tool, taskStatus, dependencyList \rangle$$

where *taskId* is the task identifier, *workflowId* is the identifier of the workflow the task belongs to, *tool* is a reference to the tool to be executed, *taskStatus* is the task status (new, ready, running, done, or failed), and *dependencyList* contains the identifiers of the other tasks this task depends on. A Task T_j depends on a task T_i (i.e., $T_i \rightarrow T_j$) if T_j can be executed only after that T_i has successfully completed its execution. Thus, the *dependencyList* of a task T_j contains a set of n tasks $T_1...T_n$ such that $T_i \rightarrow T_j$ for each $1 \leq i \leq n$.

A *tool* is defined as follows:

$$tool = \langle toolId, name, executable, libraryList, parameterList \rangle$$

where *toolId* is the tool identifier, *name* is a descriptive name for the tool, *executable* is a reference to the executable (program or script) that launches the tool, *libraryList* contains the references of the required libraries, and *parameterList* is a list of parameters used to specify input data, output data, and other configurations.

A *parameter* is defined as a tuple:

$$parameter = \langle name, description, parType, type, flag, mandatory, value \rangle$$

where *name* is the parameter name, *description* is a parameter description, *parType* specifies whether it is an input, output, or configuration parameter, *type* specifies the parameter type (e.g., file, string, integer, etc.), *flag* is a string that precedes the parameter value to allow its identification in a command line invocation, *mandatory* is a boolean that specifies whether the parameter is mandatory or not, *value* contains the parameter value.

For each tool element included in a workflow, an associated descriptor, expressed in JSON format, is included in the environment of the user who is developing the workflow. An example of JSON descriptor for a data classification tool is presented in Figure 2.

```
"J48": {
  "libraryList": ["java.exe","weka.jar"],
  "executable": "java.exe -cp weka.jar
    weka.classifiers.trees.J48",
  "parameterList":[{
      "name": "dataset", "flag": "-t",
      "mandatory": true, "parType": "input",
      "type": "file",
      "description": "Input dataset"
    },{
      "name": "confidence", "flag": "-C",
      "mandatory": false, "parType": "conf",
      "type": "real",
      "description": "Confidence value",
      "value": "0.25"
    },{
      "name": "model", "flag": "-d",
      "mandatory": true, "parType": "output",
      "type": "file",
      "description": "Output model"}]}
```

Fig. 2. Example of tool descriptor in JSON format.

The JSON descriptor of a new tool is created automatically through a guided procedure, which allows users to specify all the needed information for invoking the tool (executable, inputs and outputs, etc.).

3.2 General architecture

The architecture of DMCF includes different kinds of components that can be grouped into storage and compute components (see Figure 3).

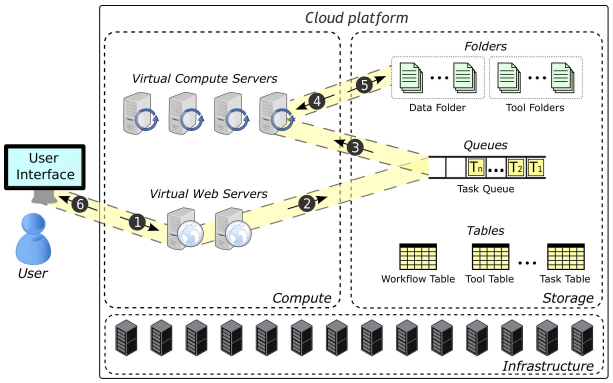


Fig. 3. Architecture of the Data Mining Cloud Framework.

The storage components include:

- A *Data Folder* that contains data sources and the results of data analysis processes. Similarly, a *Tool Folder* contains libraries and executable files for data selection, pre-processing, transformation, data mining, and results evaluation.
- *Workflow Table*, *Tool Table* and *Task Table* contain metadata information associated with workflows, tools, and tasks.
- The *Task Queue* contains the tasks ready for execution.

The compute components are:

- A pool of *Virtual Compute Servers*, which are in charge of executing the data analysis tasks.
- A pool of *Virtual Web Servers* host the Web-based user interface.

The user interface provides access to three functionalities: i) *App submission*, which allows users to define and submit data analysis 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. All the storage and compute components are executed on the infrastructure (i.e., network of physical machines) provided by the Cloud platform.

3.3 Execution mechanisms

Design and execution of a data analysis application in DMCF is a multi-step process (see numbered arrows in Figure 3):

1) The user accesses the Website and designs the workflow through a Web-based interface.
2) After workflow submission, the system creates a set of tasks and inserts them into the Task Queue.
3) Each idle Virtual Compute Server picks a task from the Task Queue, and concurrently executes it.
4) Each Virtual Compute Server gets the input dataset from its location. To this end, a file transfer is performed from the Data Folder where the dataset is located, to the local storage of the Virtual Compute Server.
5) After task completion, each Virtual Compute Server puts the result on the Data Folder.
6) The Website notifies the user whenever each task has completed, and allows her/him to access the results.

The set of tasks created on the second step depends on how many tools are invoked within the workflow; initially, only the workflow tasks without dependencies are inserted into the Task Queue. Each virtual compute server picks and executes the task from the task queue following a FIFO policy. It is important to say that in our model all the virtual compute servers have identical capabilities, because the virtual machines provided by the cloud infrastructure are homogeneous (same CPU, memory and storage).

The actions performed by each Virtual Compute Server are described in Algorithm 1. The Virtual Compute Server cyclically checks whether there are tasks ready to be executed in the Task Queue. If so, the first task is taken from the queue (line 3) and its status is changed to 'running' (line 4). Two local folders are created to temporarily stage input data and tools, which include both executables and libraries (lines 5-6). Input and output lists are created on lines 7-13. Then, the transfer of all the needed input resources (files, executables and libraries) is performed (lines 14-18). At line 19, the Virtual Compute Server locally executes the *task* and waits for its completion.

If the *task* is 'done' (line 20), the output results are copied to a remote data folder (lines 21-22), and the *task* status is changed to 'done' also in the Task Table (line 23). Then, for each *wfTask* that belongs to the same workflow of *task* (line 24), if *wfTask* has a dependency with *task* (line 25), that dependency is deleted (line 26). If *wfTask* remains without dependencies (line 27), it becomes 'ready' and is added to the Task Queue (lines 28-29). If the *task* fails (line 30), all the tasks that directly or indirectly depend on it will be marked as 'failed' (lines 31-40). Finally, the task is removed from the Task Queue (line 41), and the local data and tools folders are deleted (lines 42-43).

4 Implementing the Data Mining Cloud Framework

We implemented a version of the Data Mining Cloud Framework using the Microsoft Azure[2] platform. The choice of Azure was guided by the need of satisfying

2. http://azure.microsoft.com

Algorithm 1: Virtual Compute Server operations.

```
1  while true do
2      if TaskQueue.isNotEmpty() then
3          task ← TaskQueue.getTask();
4          TaskTable.update(task, 'running');
5          localDataFolder = <local data folder>;
6          localToolFolder = <local tool folder>;
7          inputList = <empty list>;
8          outputList = <empty list>;
9          foreach parameter in task.tool.parameterList do
10             if parameter.parType = 'input' then
11                 inputList.add(parameter);
12             else if parameter.parType = 'output' then
13                 outputList.add(parameter);
14         foreach input in inputList do
15             transfer(input, DataFolder, localDataFolder);
16         transfer(task.tool.executable, ToolFolder,
           localToolFolder);
17         foreach library in task.tool.libraryList do
18             transfer(library, ToolFolder, localToolFolder);
19         taskStatus ← execute(task, localDataFolder,
           localToolFolder);
20         if taskStatus = 'done' then
21             foreach output in outputList do
22                 transfer(output, localDataFolder, DataFolder);
23             TaskTable.update(task, 'done');
24             foreach wfTask in
               TaskTable.getTasks(task.workflowId) do
25                 if wfTask.dependencyList.contains(task) then
26                     wfTask.dependencyList.remove(task);
27                     if wfTask.dependencyList.isEmpty() then
28                         TaskTable.update(wfTask, 'ready');
29                         TaskQueue.addTask(wfTask);
30         else
31             failedTasks = <empty set>;
32             tasksToAnalyze = <empty set>;
33             tasksToAnalyze.add(task);
34             while tasksToAnalyze.isNotEmpty() do
35                 tmpTask = tasksToAnalyze.remove();
36                 TaskTable.update(tmpTask, 'failed');
37                 failedTasks.add(tmpTask);
38                 foreach wfTask in
                   TaskTable.tasks(task.workflowId) do
39                     if failedTasks.notContains(wfTask) &&
                       wfTask.dependencyList.contains(tmpTask)
                       then
40                         tasksToAnalyze.add(wfTask);
41         TaskQueue.remove(task);
42         delete(localDataFolder);
43         delete(localToolFolder);
44     else
45         sleep(sleepTime);
```

three main requirements: i) the use of a PaaS system, since implementing the execution mechanisms of our framework does not require the low level facilities provided by a IaaS; ii) the use of a platform whose components match the needs of the components defined in our architecture; iii) the use of a public Cloud, to free final users and organizations from any hardware and OS management duties.

Microsoft Azure satisfies all the requirements above, since it is a public PaaS platform whose components fully fit with those defined by DMCF. In the following, we shortly outline the Azure platform, and describe how the generic components of DMCF's architecture are mapped to the Azure's components. Finally, we discuss how DMCF could be implemented using other Cloud systems.

4.1 Microsoft Azure

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 data centers. Azure includes three layers:

- *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*), relational databases (*SQL Databases*), queues for asynchronous communication between components (*Queues*) and virtual storage (*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.

We exploited these components and mechanisms to implement DMCF, as described in the next section.

4.2 Implementing the system on Azure

As shown in Figure 3, the architecture of our framework distinguishes its high-level components into two groups, *Storage* and *Compute*, following the same approach followed by Azure and other Cloud platforms. In this way, we were able to implement the data and computing components of DMCF by fully exploiting the Storage and Compute components and functionalities provided by Azure.

For the Storage components, we adopted the following mapping with Azure: i) *Data Folder* and *Tool Folder* are implemented as Azure's Blob containers; ii) *Workflow Table*, *Tool Table* and *Task Table*, are implemented as non-relational Tables; iii) *Task Queue* is implemented as an Azure's Queue. For the Compute components, the following mapping was adopted: i) *Virtual Compute Servers* are implemented as *Worker Role* instances; ii) *Virtual Web Servers* are implemented as *Web Role* instances.

Each Worker Role instance executes the operations described in Algorithm 1. This requires file transfers to be performed when input/output data have to be moved between storage and servers. To reduce the impact of data transfer on the overall execution time, DMCF exploits the Azure's *Affinity Group* feature, which allows Data Folder and Virtual Compute Servers to be located near to each other in the same data center for optimal performance.

At least one Virtual Web Server runs continuously in the Cloud, as it serves as user front-end for DMCF.

Moreover, the user indicates the minimum and maximum number of Virtual Compute Servers he/she wants to use. DMCF exploits the auto-scaling features of Microsoft Azure that allows spinning up or shutting down Virtual Compute Servers, based on the number of tasks ready for execution in the DMCFs Task Queue.

4.3 Implementing the system on other Clouds

Although the current implementation of DMCF is based on Azure, it has been designed to abstract from specific Cloud platforms. In fact, the DMCF architecture has been designed in an abstract way to be implemented on top of other Cloud systems, such as, for instance, the popular Amazon Web Services (AWS)[3]. AWS offers compute and storage resources in the form of Web services, which comprise compute services, storage services, database services, and queuing services. If we consider AWS as a target Cloud platform, the DMCFs components could be implemented on the AWS's components as follows: i) *Data Folder* and *Tool Folder* stored on S3 (AWS's storage service); ii) *Workflow Table*, *Tool Table*, and *Task Table* stored as non-relational tables using DynamoDB (AWS's NoSQL database); iii) *Task Queue* using the Simple Queue Service (AWS's queuing system); iv) *Virtual Compute Servers* and *Virtual Web Servers* on top of EC2 (AWS's compute service).

Other than on well-known public Cloud platforms, the DMCF could be implemented on top of a private IaaS system (e.g., OpenStack[4]) using various open source software for its components. In this scenario, the DMCF components could be implemented as follows: i) *Data Folder* and *Tool Folder* as FTP servers (e.g., using Filezilla); ii) *Workflow Table*, *Tool Table*, and *Task Table* as non-relational Tables (e.g., MongoDB); iii) *Task Queue* using a message-oriented middleware (e.g., Java Message Service); iv) *Virtual Compute Servers* as batch applications (e.g., Java applications) and *Virtual Web Servers* as Web servers (e.g., Apache/Tomcat).

5 Programming Data Analysis Workflows

In this section we focus on data analysis workflow programming by describing workflow formalism, composition and execution in DMCF.

5.1 Workflow formalism

The DMCF includes a visual programming interface and its services to support the composition and execution of data analysis workflows. Workflows provide 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, data analysis workflows can produce results that can confirm real experiments or provide insights that cannot be achieved through laboratory experiments.

A visual programming language, called VL4Cloud (Visual Language for Cloud), allows users to develop applications by programming the workflow components graphically. As discussed in Section 2 there are several systems for scientific workflow programming, but we chose to define VL4Cloud as a new language to provide visual elements (data/tool arrays) and patterns (parameter/input/tool sweeping) that are specific to parallel data mining, and to ensure full integration of the programming interface with DMCF's engine. There are also business workflow languages like BPMN [34] that provide facilities for the execution of business processes including human tasks, fault handling, transactions, or quality of service features. However, as argued by Sonntag et al. [35], most of the control flow structures of business workflows, including those provided by BPMN, are not required to define data analysis applications.

VL4Cloud workflows are directed acyclic graphs whose nodes represent resources and whose edges represent dependencies among the resources. Workflows include two types of nodes. *Data* node represents an input or output data element; two subtypes exist: Dataset representing a data collection, and Model representing a model generated by a data analysis tool (e.g., a decision tree). *Tool* node represents a tool performing any kind of operation that can be applied to a data node (splitting, 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. For example, Figure 4 shows a *J48* Tool (an implementation of the C4.5 algorithm [36] provided by the Weka toolkit [37]) that takes in input a *TrainSet* and generates a *Model*. For each Tool node, input/output connections are allowed on the basis of the Tool descriptor present in ToolTable.

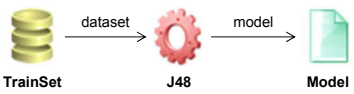
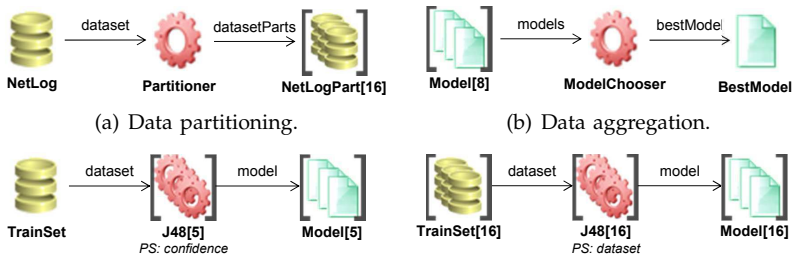


Fig. 4. Example of a tool connected to an input dataset and an output model.

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.

Several workflow patterns can be implemented with VL4Cloud. Figure 5 shows four examples of

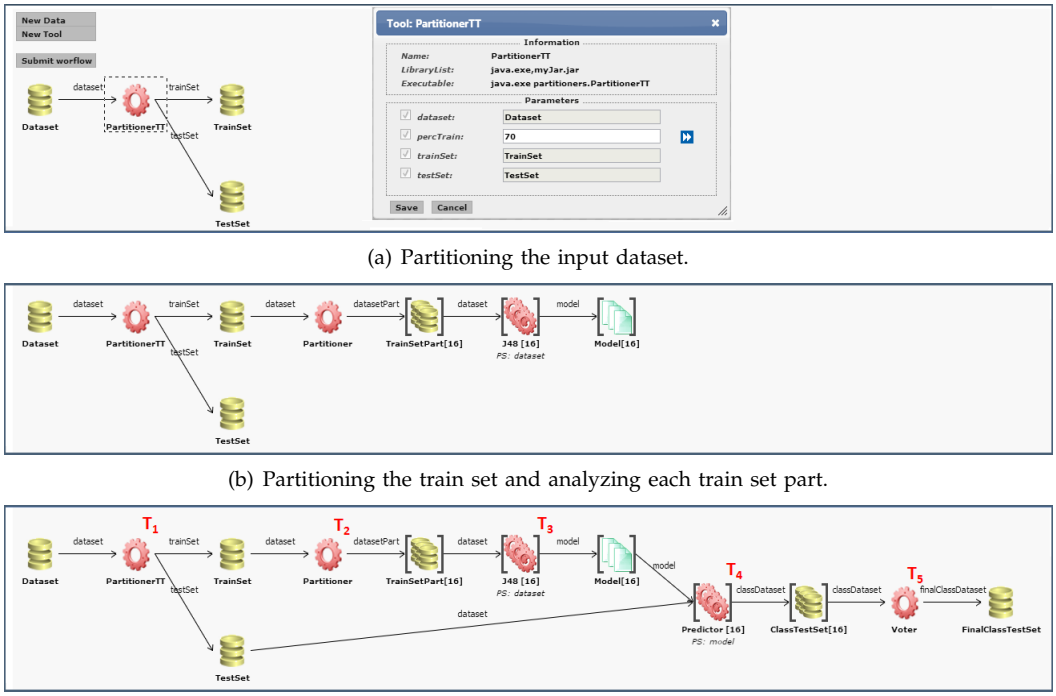
3. http://aws.amazon.com
4. http://www.openstack.org



(a) Data partitioning. (b) Data aggregation.

(c) Parameter sweeping. (d) Input sweeping.

Fig. 5. Visual workflow patterns.



(a) Partitioning the input dataset.

(b) Partitioning the train set and analyzing each train set part.

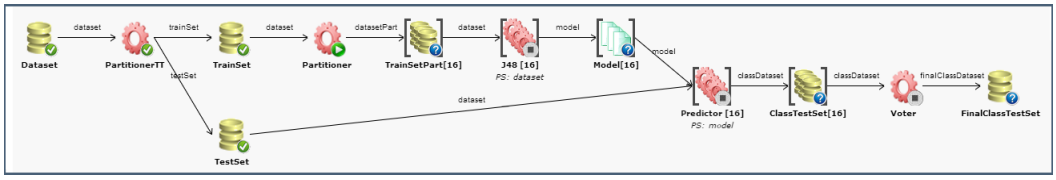
(c) Classifying the test set and voting.

Fig. 6. Example of workflow composition.

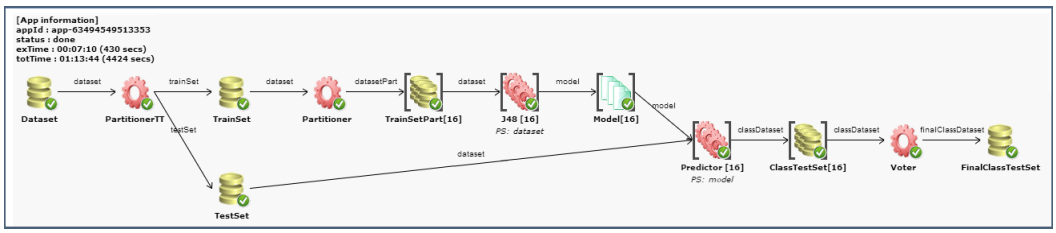
patterns that can be defined: data partitioning, data aggregation, parameter sweeping and input sweeping.

The *data partitioning* pattern produces two or more output data from an input data element, as in Figure 5(a), where a Partitioner tool divides a dataset into a number of splits. The *data aggregation* pattern generates one output data from multiple input data, as in Figure 5(b), where a ModelChooser tool takes as input eight data mining models and chooses the best one based on some evaluation criteria. *Parameter sweeping* is a data analysis pattern in which a dataset is analyzed by multiple instances of the same tool with different parameters, as in the example shown in Figure 5(c). In this example, a training set is processed in parallel by five instances of the J48 data classification tool to produce the same number of data mining models. The J48 instances differ each other by the value of a single parameter, the *confidence* factor, which has been configured (through the visual interface) to range from 0.1 to 0.5 with a step of 0.1. Finally, *input sweeping* is a pattern in which a set of input data is analyzed independently to produce the same number of output data. It is similar to the parameter sweeping pattern, with the difference that in this case the sweeping is done on the input data rather than on a tool parameter. An example of input sweeping pattern is represented in Figure 5(d), where 16 training sets are processed in parallel by 16 instances of J48, to produce the same number of data mining models.

Van Der Aalst et al. [38] proposed a formal description of several control-flow workflow patterns that can be used to give a reference for the possibilities that a workflow management systems can provide. Taking [38] as a reference, DMCFs visual language fully covers the sequence, parallel split and synchronization workflow patterns. However, compared to existing visual workflow languages, DMCF has a higher degree of flexibility on the definition of parallel patterns



(a) Workflow running.



(b) Workflow completed.

Fig. 7. Workflow execution.

(i.e., parameter, input, and tool sweeping), which can be combined together in multiple ways by exploiting the data and tool array formalisms provided by our language.

5.2 Workflow composition

In order to present the main features of the visual programming interface of the DMCF, we use as an example a data analysis application composed of several sequential and parallel steps.

The example application analyzes a dataset by using n instances of the J48 classification algorithm that work on n partitions of the training set and generate n classification models. By using the n generated models and the test set, n predictors produce in parallel n classified datasets. In the final step of the workflow, a voter generates the final classification by assigning a class to each data item. This is done by choosing the class predicted by the majority of the models [39].

Figure 6(a) shows a snapshot of the visual interface with the first step of the workflow, where the original dataset is split in training and test set by a partitioning tool. Since a set of parameters is associated with each workflow node, the interface allows users to configure them through a pop-up panel. For example, the central part of Figure 6(a) shows the configuration panel for the partitioning tool. In this case, only one parameter can be specified, namely which percentage of the input dataset must be used as training set.

Figure 6(b) shows how the training set is partitioned into 16 parts using another partitioning tool. The 16 training sets resulting from the partitioning are represented in the workflow as a single data array node, labeled as *TrainSetPart[16]*. Then, the workflow specifies that the 16 training sets must be analyzed in parallel by 16 instances of the J48 classification algorithm, to produce the same number of classification models. A tool array node, labeled as *J48[16]*, is used to represent the 16 instances of the J48 algorithm, while another data array node, labeled as *Model[16]*, represents the models generated by the classification algorithms. In practice, this part of the workflow specifies that *J48[i]* takes in input *TrainSetPart[i]* to produce *Model[i]*, for $1 \leq i \leq 16$.

Figure 6(c) shows the complete workflow, which classifies the test set using the 16 models generated previously. The classification is performed by 16 predictors that run in parallel to produce 16 classified test sets. In more detail, *Predictor[i]* takes in input *TestSet* and *Model[i]* to produce *ClassTestSet[i]*, for $1 \leq i \leq 16$. Finally, the 16 classified test sets are passed to a voter that produces the final dataset, labeled as *FinalClassTestSet*. When the workflow design is complete, the workflow execution can start and it proceeds as detailed in the next section.

5.3 Workflow execution

The workflow defined in the previous section includes five tools (PartitionerTT, Partitioner, J48, Predictor, and Voter), which are translated into five groups of tasks, indicated as $\{T_1...T_5\}$, as shown in Figure 6(c).

The execution order of the workflow tasks depends on the dependencies specified by the workflow edges. To ensure the correct execution order, to each task is associated a list of tasks that must be completed before starting its execution. Figure 8 shows a possible order in which the tasks are generated and inserted into the Task Queue. For each task, the list of tasks to be completed before its execution is included. Note that task group T_3, which represents the execution of 16 instances of J48, is translated into 16 tasks $T_3[1]...T_3[16]$. Similarly, T_4 is translated into tasks $T_4[1]...T_4[16]$.

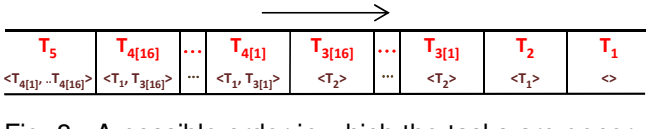


Fig. 8. A possible order in which the tasks are generated and inserted into the Task Queue.

According with the tasks dependencies specified by the workflow, the execution of T_2 will start after completion of T_1. As soon as T_2 completes, the 16 tasks that compose T_3 can be run concurrently. Each task $T_4[i]$ can be executed only after completion of both T_2 and $T_3[i]$, for $1 \leq i \leq 16$. Finally T_5 will start after completion of all tasks that compose T_4.

Figure 7(a) shows a snapshot of the workflow taken during its execution. The figure shows that PartitionerTT has completed the execution, Partitioner is running, while the other tools are still submitted. Figure 7(b) shows the workflow after completion of its execution. Some statistics about the overall application are shown on the upper left part of the window. In this example, it is shown that, using 16 virtual machines, the workflow completed 430 seconds after its submission, whereas the total execution time (i.e., the sum of the execution times of all the tasks) is 4424 seconds.

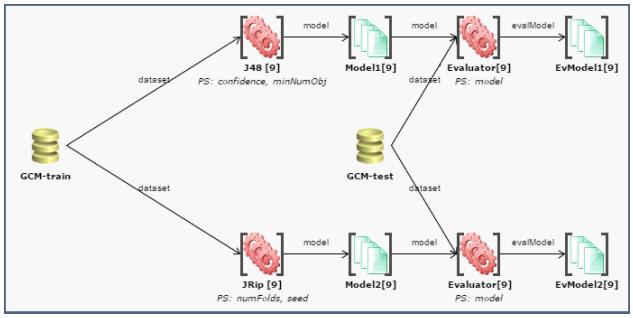
6 Experimental Evaluation

We present some experimental performance results obtained executing two VL4Cloud workflows with DMCF. The first workflow represents an ensemble learning application, while the second workflow represents a parallel classification application. The main goal of the first workflow is to illustrate the VL4Cloud capability of expressing a complex data analysis process, while the second workflow shows the good scalability that can be achieved with our approach. The Cloud environment used for the experimental evaluation was composed by 128 virtual servers, each one equipped with one dual-core 1.66 GHz CPU, 3.5 GB of memory, and 135 GB of disk space.

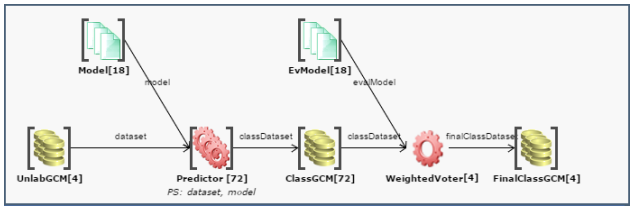
6.1 Ensemble learning workflow

This workflow is the implementation of a multi-class cancer classifier based on the analysis of genes, using an ensemble learning approach [39]. The input dataset is the Global Cancer Map (GCM)[5], which contains the gene expression profiles of 280 samples representing 14 common human cancer classes. For each sample is reported the status of 16,063 genes and the type of tumor (class label). The GCM dataset is available as a training set containing 144 instances and as a test set containing 46 instances. The goal is to classify an unlabeled dataset (*UnclassGCM*) composed by 20,000 samples, divided in four parts.

The workflow starts by analyzing the training set using l instances of the J48 classification tool and m instances of the JRip classification tool (Weka's implementation of the Ripper [40] algorithm). The l J48 instances are obtained by sweeping the *confidence* and the *minNumObj* (minimum number of instances per leaf) parameters, while the m JRip instances are obtained by sweeping the *numFolds* (number of folders) and *seed* parameters. The resulting $l+m$ classification models (classifiers) are passes as input to $l+m$ evaluators, which produce an evaluation of each model against the test set. Then, k unclassified datasets are classified using the $l+m$ models by $k(l+m)$ predictors. Finally, k voters take in input $l+m$ model evaluations and the $k(l+m)$ classified datasets, producing k classified datasets through weighted voting.



(a) First workflow for models creation and evaluation.



(b) Second workflow to classify 4 unlabeled datasets.

Fig. 9. Ensemble learning application.

The first workflow is shown in Figure 9(a). The input training set, *GCM-train*, is analyzed by: i) 9 instances of the J48 classification algorithm, obtained by sweeping the *confidence* and the *minNumObj* parameters; ii) 9 instances of the JRip classification algorithm, obtained by sweeping the *numFolds* (number of folders) and *seed* parameters. Each instance of J48, *J48[i]*, generate a model (*Model1[i]*) that is passed as input to an *Evaluator* to generate an evaluation of the model (*EvModel1[i]*). In the same way, each instance of JRip, *JRip[j]*, generates a model (*Model2[j]*) that is passed as input to an *Evaluator* to generate an evaluation of the model (*EvModel2[j]*).

At this point, we can use our ensemble classifier to classify new datasets. This is shown in the second workflow (see Figure 9(b)). We created an unlabeled dataset composed by 20,000 samples, *UnlabGCM[4]*, divided in four parts. Each part is classified by weighted majority voting using the models (*Model[18]*) and the corresponding evaluations (*EvModel[18]*) generated by the first workflow.

In general, the workflow is composed of $2(l+m)+k(l+m)+k$ tasks, which are related with each other as specified by the dependency graph shown in Figure 10. In the specific example (with $l=9$, $m=9$, $k=4$) the number of tasks is 112.

5. http://eps.upo.es/bigs/datasets.html

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2589243, IEEE Transactions on Services Computing

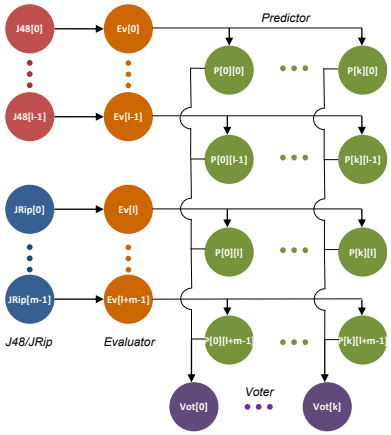


Fig. 10. Task dependency graph associated with the ensemble learning workflows in Figure 9.

The example above demonstrates the flexibility of VL4Cloud. In addition, the experimental evaluation of the workflow using DMCF, conducted using 19 virtual servers, showed a significant reduction of turnaround time compared to that achieved by the sequential execution. In particular, the turnaround time decreased from about 162 minutes using a single server, to about 11 minutes using 19 servers, which corresponds to a speedup of about 14.7.

6.2 Parallel classification workflow

We evaluate here the scalability obtained executing a larger version of the workflow presented in Section 5.1 (see Figure 6(c)), where n (i.e., number of training set partitions) is increased to 128. The workflow is composed of $3 + 2n$ tasks, whose dependency graph is shown in Figure 11. In the specific example the total number of tasks is 259.

Fig. 11. Task dependency graph associated with the parallel classification workflow.

In order to evaluate the system with increasing workloads, we generated three datasets with size of 5 GB, 10 GB and 20GB from the *KDD Cup 1999*'s dataset[6], which contains a wide variety of simulated intrusion records in a military network environment.

6. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99

Figure 12(a) shows the turnaround times of the workflow, obtained varying the number of virtual servers used to run it on the Cloud from 1 (sequential execution) to 128 (maximum parallelism) and the dataset size. For the smallest dataset (5 GB), the turnaround time decreases from about 46 hours (about 2 days) on a single server, to about 33 minutes using 128 servers. For the medium-size dataset (10 GB), the turnaround time decreases from 115 hours to 1.2 hours, while for the largest dataset (20 GB), the turnaround time ranges from 257 hours (more than 10 days) to 2.7 hours. These are evident and significant reductions of time, which prove the system scalability.

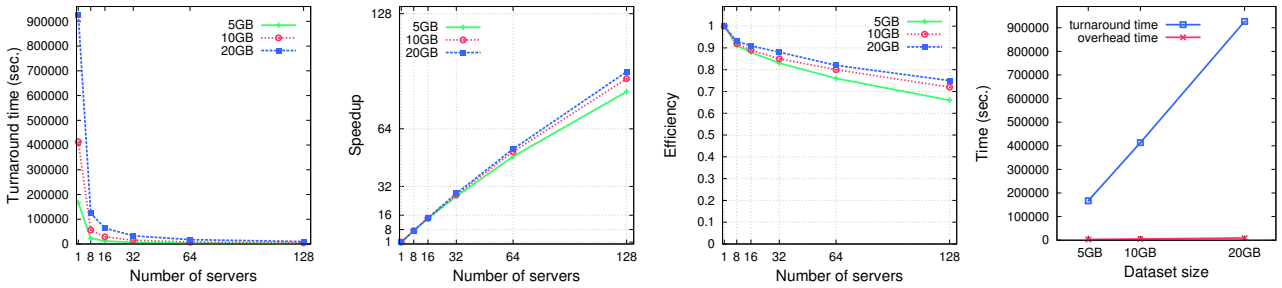
The scalability achieved using DMCF can be further evaluated through Figure 12(b), which illustrates the relative speedup obtained by using up to 128 servers. For the 5 GB dataset, the speedup passes from 7.3 using 8 servers to 84.7 using 128 servers. For the 10 GB dataset, the speedup ranges from 7.4 to 91.9. Finally, with the 20 GB dataset, we obtained a speedup ranging from 7.5 to 95.7. This is a very positive result, taking into account that some sequential parts of the implemented application (namely, partitioning and voting) cannot run in parallel.

Figure 12(c) shows the application efficiency, calculated as the speedup divided by the number of used servers. As shown in the figure, for the largest dataset the efficiency on 32 servers is equal to 0.9 whereas on 128 servers it is equal to 0.75. Thus in this case, 75% of the computing power of each used server is exploited.

We also evaluated the overhead introduced by DMCF. We define as overhead the time required by the system to perform preliminary operations (e.g., getting the task from the Task Queue, downloading libraries and input files from the Cloud storage) and final operations (e.g., updating the Task Table, uploading the output files to the Cloud storage) related to the execution of each workflow task. In Algorithm 1, the preliminary operations are those in lines 3-18, while the final operations are in lines 20-43. Figure 12(d) shows the overhead time of the workflow, compared with the turnaround time of the same workflow executed on 128 servers. We observe that the overhead is a small fraction of the turnaround time. For example, the overhead is equal to 50 minutes on a turnaround time of 46 hours considering the smallest dataset, while with the largest dataset the overhead is 2.5 hours over 257 hours. This means that the overhead lies in the range of 1%-1.7% of the total execution time.

6.3 Scalability remarks

To summarize, we present in Table 2 the speedups achieved by executing some data analysis applications with DMCF. Besides the speedups of the ensemble learning and parallel classification workflows described earlier in this section, the table reports the speedups achieved by the following applications:



(a) Turnaround time vs #servers. (b) Speedup vs #servers. (c) Efficiency vs #servers. (d) Overhead time vs data size.

Fig. 12. Parallel classification workflow.

i) *Parallel clustering* [41], where multiple instances of a clustering algorithm are executed concurrently on a large census dataset to find the most accurate data grouping; *ii*) *Association rule analysis* [8], which is a workflow for association rule analysis between genome variations and clinical conditions of a group of patients; *iii*) *Trajectory mining* [9], a data analysis workflow for discovering patterns and rules from trajectory data of vehicles in a wide urban scenario.

TABLE 2
Speedup values of some DMCF applications.

Application	#tasks	#servers	Speedup
Parallel clustering [41]	16	16	9.0
Association rules analysis [8]	53	16	15.2
Trajectory mining [9]	131	64	49.3
Ensemble learning (Sec. 6.1)	112	19	14.7
Parallel classification (Sec. 6.2)	259	128	95.7

The best speedup is achieved in applications where many tasks can be run in parallel, and the concurrent tasks are homogeneous in terms of execution times. This is the case of the association rule analysis, where, after a short sequential task, several data mining tasks of similar duration are executed in parallel.

An example in which task heterogeneity limits scalability is the parallel clustering application, where the tasks in charge of grouping data into a high number of clusters are much slower than those looking for a lower number of clusters. Therefore, in this case the speedup does not increase linearly with the number of servers used, because the turnaround time is bound to the execution time of the slowest task instances.

In larger workflows, even if tasks are not perfectly homogeneous, a good scalability can be achieved thanks to the better server utilization deriving by the higher number of tasks to be executed in parallel. This is the case of the trajectory mining workflow, where we achieved a speedup of 49.3 on 64 servers despite a significant variability in tasks execution times.

The experimental results demonstrate the good scalability achieved using DMCF to execute different types of data analysis workflows on a Cloud platform.

7 Conclusions

Cloud systems can be used as scalable infrastructures to support high-performance platforms for data analysis applications. Based on this vision, we designed DMCF for large-scale data analysis on the Cloud. The main contribution of DMCF is the integration of different hardware/software solutions for high-level programming, management and execution of parallel data mining workflows.

We evaluated the performance of DMCF through the execution of workflow-based data analysis applications on a pool of virtual servers hosted by a Microsoft Cloud data center. The experimental results demonstrated the effectiveness of the framework, as well as the scalability that can be achieved through the execution of data analysis applications on the Cloud.

Besides performance considerations, we point out that the main goal of DMCF is providing an easy-to-use SaaS interface to reliable data mining algorithms, thus enabling end-users to focus on their data analysis applications without worrying about low level computing and storage details, which are transparently managed by the system.

References

[1] A. Burd *et al.*, "Pi of the sky-all-sky, real-time search for fast optical transients," *New Astronomy*, vol. 10, no. 5, pp. 409 – 416, 2005.

[2] O. Rubel, C. Geddes, M. Chen, E. Cormier-Michel, and E. Bethel, "Feature-based analysis of plasma-based particle acceleration data," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 20, no. 2, pp. 196–210, 2014.

[3] T. Tucker, M. Marra, and J. Friedman, "Massively parallel sequencing: The next big thing in genetic medicine," *The American Journal of Human Genetics*, vol. 85, no. 2, pp. 142–154, 2009.

[4] *The SAGE Handbook of Social Network Analysis*, 0th ed. SAGE Publications Ltd, 2014.

[5] T. Hey, S. Tansley, and K. Tolle, Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

[6] D. Talia and P. Trunfio, "How distributed data mining tasks can thrive as knowledge services," *Commun. ACM*, vol. 53, no. 7, pp. 132–137, 2010.

[7] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the use of cloud computing for scientific workflows," in *eScience, 2008*, 2008, pp. 640–645.

[8] G. Agapito, M. Cannataro, P. H. Guzzi, F. Marozzo, D. Talia, and P. Trunfio, "Cloud4snp: Distributed analysis of snp microarray data on the cloud," in *Proc. of the ACM Conference on Bioinformatics, Computational Biology and Biomedical Informatics 2013 (ACM BCB 2013)*, Washington, DC, USA, 2013, p. 468.

[9] A. Altomare, E. Cesario, C. Comito, F. Marozzo, and D. Talia, "Trajectory pattern mining over a cloud-based framework for urban computing," in *Proc. of the 16th Int. Conference on High Performance Computing and Communications (HPCC 2014)*, Paris, France, 2014, pp. 367–374.

[10] D. Talia, "Workflow systems for science: Concepts and tools," *ISRN Software Engineering*, 2013.

[11] M. Bux and U. Leser, "Parallelization in scientific workflow management systems," *CoRR*, 2013.

[12] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *SIGMOD Rec.*, vol. 34, no. 3, pp. 44–49, 2005.

[13] J. Goecks, A. Nekrutenko, J. Taylor, and T. G. Team, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biology*, vol. 11, no. 8, p. R86, 2010.

[14] K. Maheshwari, A. Rodriguez, D. Kelly, R. Madduri, J. Wozniak, M. Wilde, and I. Foster, "Enabling multi-task computation on galaxy-based gateways using swift," in *IEEE Int. Conference on Cluster Computing*, 2013, pp. 1–3.

[15] K. Wolstencroft *et al.*, "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud," *Nucleic Acids Research*, vol. 41, pp. 557–561, 2013.

[16] M. Abouelhoda, S. Issa, and M. Ghanem, "Tavaxy: Integrating taverna and galaxy workflows with cloud computing support," *BMC Bioinformatics*, vol. 13, no. 1, 2012.

[17] V. Podpečan, M. Zemenova, and N. Lavrač, "Orange4ws environment for service-oriented data mining," *Comput. J.*, vol. 55, no. 1, pp. 82–98, 2012.

[18] B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurr. Comput.: Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, 2006.

[19] H. Hiden, S. Woodman, P. Watson, and J. Cala, "Developing cloud applications using the e-Science Central platform," *Philosophical Transactions of the Royal Society A: Mathematical,Physical and Engineering Sciences*, vol. 371, no. 1983, 2013.

[20] J. Kranjc, V. Podpečan, and N. Lavrač, "ClowdFlows: A Cloud Based Scientific Workflow Platform," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science. Springer, 2012, vol. 7524, pp. 816–819.

[21] E. Deelman *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[22] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling, "Data Sharing Options for Scientific Workflows on Amazon EC2," in *High Performance Computing, Networking, Storage and Analysis (SC), 2010 Int. Conference for*, ser. SC '10. IEEE, 2010, pp. 1–9.

[23] S. Ostermann, R. Prodan, and T. Fahringer, "Extending grids with cloud resource management for scientific computing," in *IEEE/ACM Int. on Grid Computing*, 2009, pp. 42–49.

[24] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, A. Balasko, K. Karoczkai, and I. Marton, "Ws-pgrade/guse generic dci gateway framework for a large variety of user communities," *J. Grid Comput.*, vol. 10, no. 4, pp. 601–630, 2012.

[25] W. M. P. Van Der Aalst and T. Hofstede, "Yawl: yet another workflow language," *Information Systems*, vol. 30, no. 4, pp. 245 – 275, 2005.

[26] D. M. Schunselaar, H. Verbeek, H. A. Reijers, and W. M. van der Aalst, "Yawl in the cloud: Supporting process sharing and variability," in *Business Process Management Workshops*. Springer Int. Publishing, 2014, pp. 367–379.

[27] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[28] J. Wozniak, T. Armstrong, M. Wilde, D. Katz, E. Lusk, and I. Foster, "Swift/t: Large-scale application composition via distributed-memory dataflow processing," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM Int. Symposium on*, 2013, pp. 95–102.

[29] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. lvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, and R. Badia, "ServiceSs: An interoperable programming framework for the cloud," *J. of Grid Computing*, vol. 12, no. 1, pp. 67–91, 2014.

[30] D. de Oliveira, E. Ogasawara, F. Baiao, and M. Mattoso, "Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows," in *3rd IEEE Int. Conf. on Cloud Computing*, 2010, pp. 378–385.

[31] R. Duan, R. Prodan, and X. Li, "Multi-objective game theoretic schedulingof bag-of-tasks workflows on hybrid clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 29–42, 2014.

[32] D. de Oliveira, K. Ocaa, F. Baio, and M. Mattoso, "A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds," *J. of Grid Computing*, vol. 10, no. 3, pp. 521–552, 2012.

[33] P. J. A. Cock, B. A. Grning, K. Paszkiewicz, and L. Pritchard, "Galaxy tools and workflows for sequence analysis with applications in molecular plant pathology," *PeerJ*, vol. 1, p. e167, 2013.

[34] T. Allweyer, *BPMN 2.0*. BoD, 2010.

[35] M. Sonntag, D. Karastoyanova, and E. Deelman, "Bridging the gap between business and scientific workflows: Humans in the loop of scientific workflows," in *e-Science (e-Science), 2010 IEEE Sixth Int. Conference on*, 2010, pp. 206–213.

[36] J. R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[37] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.

[38] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[39] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[40] W. W. Cohen, "Fast effective rule induction," in *In Proc. of the Twelfth Int. Conference on Machine Learning*. Morgan Kaufmann, 1995, pp. 115–123.

[41] F. Marozzo, D. Talia, and P. Trunfio, "A cloud framework for parameter sweeping data mining applications," in *Third Inter. Conference on Cloud Computing Technology and Science*, ser. Cloudcom'11, Athens, Greece, 2011, pp. 367–374.

Fabrizio Marozzo received a Ph.D. in systems and computer engineering from the University of Calabria, Italy. In 2011-2012 he visited the Barcelona SuperComputing Center (BSC) for a research internship. He co-authored many papers published in conference proceedings, edited volumes and international journals. His research interests include distributed systems, Cloud computing, data mining and peer-to-peer networks.

Domenico Talia is a professor of computer engineering at the University of Calabria. His research interests include parallel and distributed data mining algorithms, Cloud computing, distributed knowledge discovery and peer-to-peer systems. He is a member of several editorial boards, including IEEE Transactions on Cloud computing, Future Generation Computer Systems, and the Journal of Cloud Computing.

Paolo Trunfio is an associate professor of computer engineering at the University of Calabria, Italy. In 2007 he was a visiting researcher at the Swedish Institute of Computer Science (SICS). Previously, he was a research collaborator at the Institute of Systems and Computer Science of the Italian National Research Council (ISI-CNR). His research interests include Cloud computing, data mining, and peer-to-peer systems.