Programming Visual and Script-based Big Data Analytics Workflows on Clouds

Loris Belcastro, Fabrizio Marozzo, Domenico Talia, Paolo Trunfio
DIMES, University of Calabria
Via P. Bucci 41c, 87036 Rende, Italy
{lbelcastro, fmarozzo, talia, trunfio}@dimes.unical.it

Abstract

Data analysis applications often include large datasets and complex software systems in which multiple data processing tools are executed in a coordinated way. Data analysis workflows are effective in expressing task coordination and they can be designed through visual- and script-based programming paradigms. The Data Mining Cloud Framework (DMCF) supports the design and scalable execution of data analysis applications on Cloud platforms. A workflow in DMCF can be developed using a visual- or a script-based language. The visual language, called VL4Cloud, is based on a design approach for high-level users, e.g., domain expert analysts having a limited knowledge of programming paradigms. The script-based language JS4Cloud is provided as a flexible programming paradigm for skilled users who prefer to code their workflows through scripts. Both languages implement a data-driven task parallelism that spawns ready-to-run tasks to Cloud resources. In addition, they exploit implicit parallelism that frees users from duties like workload partitioning, synchronization and communication. In this chapter, we present the DMCF framework and discuss how its workflow paradigm has been integrated with the MapReduce model. In particular, we describe how VL4Cloud/JS4Cloud workflows can include MapReduce tools, and how these workflows are executed in parallel on DMCF enabling scalable data processing on Clouds.

1. Introduction

Cloud computing systems provide elastic services, high performance and scalable data storage to a large and everyday increasing number of users [1]. Clouds enlarged the computing and storage offer of distributed computing systems by providing advanced Internet services that complement and complete functionalities of distributed computing provided by the Web, Grids, and peer-to-peer networks. In fact, Cloud computing systems provide large-scale computing infrastructures for complex high-performance applications. Most of those applications use big data repositories and often access and analyze them to extract useful information.

Big data is a new and over-used term that refers to massive, heterogeneous, and often unstructured digital content that is difficult to process using traditional data management tools and techniques. The term includes the complexity and variety of data and data types, real-time data collection and processing needs, and the value that can be obtained by smart analytics. Advanced data mining techniques and associated tools can help extract information from large, complex datasets that are useful in making

informed decisions in many business and scientific applications including advertising, market sales, social studies, bioinformatics, and high-energy physics. Combining big data analytics and knowledge discovery techniques with scalable computing systems will produce new insights in a shorter time [2].

Although a few Cloud-based analytics platforms are available today, current research work foresees that they will become common within a few years. Some current solutions are open source systems such as Apache Hadoop and SciDB, while others are proprietary solutions provided by companies such as Google, IBM, Microsoft, EMC, BigML, Splunk Storm, Kognitio, and InsightsOne. As such platforms become available, researchers are increasingly porting powerful data mining programming tools and strategies to the Cloud to exploit complex and flexible software models, such as the distributed workflow paradigm. The increasing use of service-oriented computing in many application domains is accelerating this trend. Developers and researchers can adopt the software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) models to implement big data analytics solutions in the Cloud. In such a way, data mining tasks and knowledge discovery applications can be offered as high-level services on Clouds. This approach creates a new way to deliver data analysis software that is called data analytics as a service (DAaaS).

This chapter describes a Data Mining Cloud Framework (DMCF) that we developed according to this approach. In DMCF, data analysis workflows can be designed through a visual- or a script-based formalism. The visual formalism, called VL4Cloud, is a very effective design approach for high-level users, e.g., domain expert analysts having a limited knowledge of programming languages. As an alternative, the script-based language, called JS4Cloud, offers a flexible programming approach for skilled users who prefer to program their workflows using a more technical approach. We discuss how the DMCF framework based on the workflow model has been integrated with the MapReduce paradigm. In particular, we describe how VL4Cloud/JS4Cloud workflows can include MapReduce algorithms and tools, and how these workflows can be executed in parallel on DMCF to support scalable data analysis on Clouds.

The remainder of the chapter is organized as follows. Section 2 presents the Data Mining Cloud Framework introducing its architecture, the parallel execution model and the workflow-based programming paradigm offered by VL4Cloud and JS4Cloud. Section 3 describes how the VL4Cloud and JS4Cloud languages have been extended to integrate with the MapReduce model. Section 4 discusses a data mining application implemented using the proposed approach. Finally, Section 5 concludes the chapter.

2. Data Mining Cloud Framework

The Data Mining Cloud Framework (DMCF) is a software system developed for allowing users to design and execute data analysis workflows on Clouds. DMCF supports a large variety of data analysis processes, including single-task applications, parameter sweeping applications, and workflow-based applications. A Web-based user interface allows users to compose their applications and to submit them for execution to a Cloud platform, according to a Software-as-a-Service approach.

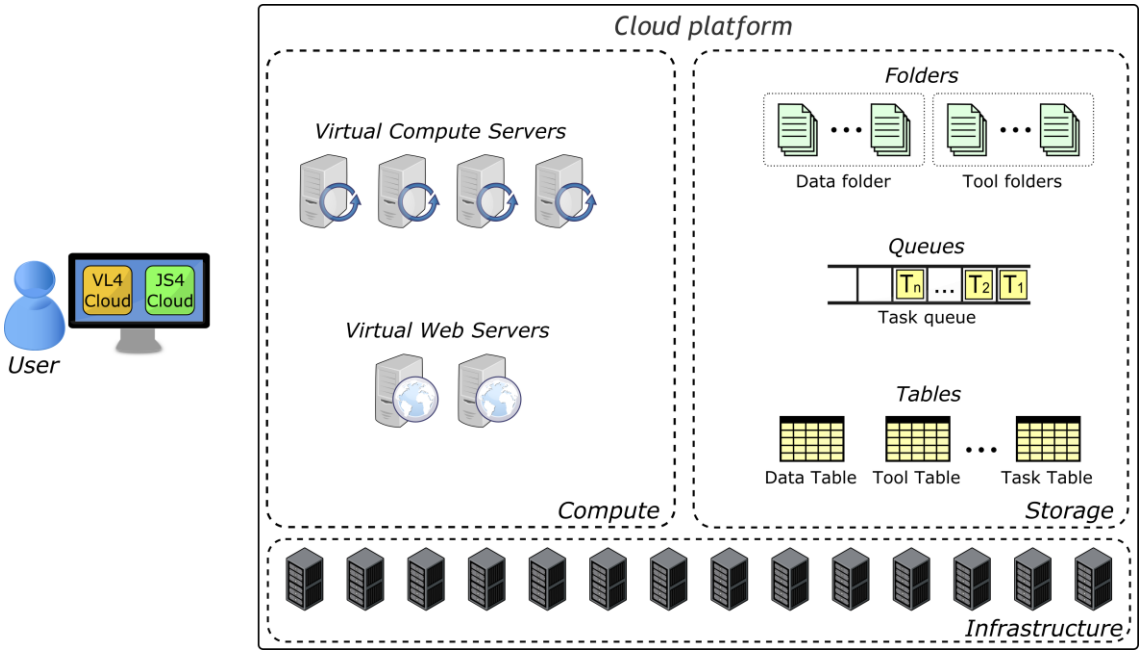

Figure 1: *Architecture of Data Mining Cloud Framework.*

The DMCF's architecture includes a set of components that can be classified as *storage* and *compute* components [3] (see Figure 1).
The *storage* components include:

- A *Data Folder* that 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.
- The *Data Table*, *Tool Table* and *Task Table* that contain metadata information associated with data, tools, and tasks.
- The *Task Queue* that manages the tasks to be executed.

The *compute* components are:

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

The user interface provides 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.

The Data Mining Cloud Framework architecture has been designed as a reference architecture to be implemented on different Cloud systems. However, a first implementation of the framework has been carried out on the Microsoft Azure Cloud platform[1] and has been evaluated through a set of data analysis applications executed on a Microsoft Cloud data center.

The DMCF framework takes advantage of cloud computing features, such as elasticity of resources provisioning. In DMCF, at least one Virtual Web Server runs continuously

[1] http://azure.microsoft.com/

in the Cloud, as it serves as user front-end. In addition, users specify the minimum and maximum number of Virtual Compute Servers. DMCF can exploit the auto-scaling features of Microsoft Azure that allows dynamic spinning up or shutting down Virtual Compute Servers, based on the number of tasks ready for execution in the DMCF's Task Queue. Since storage is managed by the Cloud platform, the number of storage servers is transparent to the user.

The remainder of the section outlines applications execution in DMCF, and describes the DMCF's visual- and script-based formalisms used to implement workflow applications.

2.1 Applications execution

For designing and executing a knowledge discovery application, users interact with the system performing the following steps:

1. The Website is used to design an application (either single-task, parameter sweeping, or workflow-based) through a Web-based interface that offers both the visual programming interface and the script.
2. When a user submits an application, the system creates a set of tasks and inserts them into the Task Queue on the basis of the application requirements.
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 the location specified by the application. To this end, 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 as soon as her/his task(s) have completed, and allows her/him to access the results.

The set of tasks created on the second step depends on the type of application submitted by a user. In the case of a single-task application, just one data mining task is inserted into the Task Queue. If users submit a parameter sweeping application, a set of tasks corresponding to the combinations of the input parameters values are executed in parallel. If a workflow-based application has to be executed, the set of tasks created depends on how many data analysis tools are invoked within the workflow. Initially, only the workflow tasks without dependencies are inserted into the Task Queue [3].

2.2 Workflow formalisms

The DMCF allows creating data mining and knowledge discovery applications using workflow formalisms. Workflows 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. In particular, DMCF allows to program workflow applications using two languages:

- *VL4Cloud* (Visual Language for Cloud), a visual programming language that lets users develop applications by programming the workflow components graphically [4].
- *JS4Cloud* (JavaScript for Cloud), a scripting language for programming data analysis workflows based on JavaScript [5].

Both languages use two key programming abstractions:

- *Data* elements denote input files or storage elements (e.g., a dataset to be analyzed) or output files or stored elements (e.g., a data mining model).
- *Tool* elements denote algorithms, software tools or complex applications performing any kind of operation that can be applied to a data element (data mining, filtering, partitioning, etc.).

Another common element is the *task* concept, which represents the unit of parallelism in our model. A task is a Tool, invoked in the workflow, which is intended to run in parallel with other tasks on a set of Cloud resources. According to this approach, VL4Cloud and JS4Cloud implement a *data-driven task parallelism*. This means that, as soon as a task does not depend on any other task in the same workflow, the runtime asynchronously spawns it to the first available virtual machine. A task T_j does not depend on a task T_i belonging to the same workflow (with $i \neq j$), if T_j during its execution does not read any data element created by T_i.

In *VL4Cloud*, workflows are directed acyclic graphs whose nodes represent data and tools elements. 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 type 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 2 shows an example of data analysis workflow developed using the visual workflow formalism of DMCF [6].

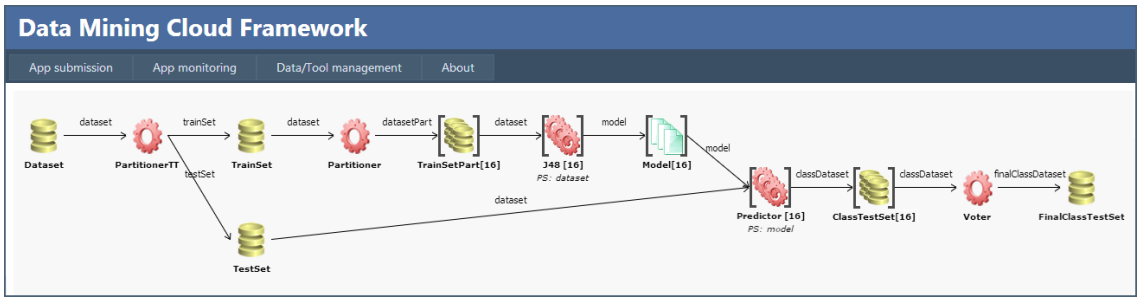

Figure 2: *Example of data analysis application designed using VL4Cloud.*

In *JS4Cloud*, workflows are defined with a JavaScript code that interacts with Data and Tool elements through three functions:

- *Data Access*, for accessing a Data element stored in the Cloud;
- *Data Definition*, to define a new Data element that will be created at runtime as a result of a Tool execution;
- *Tool Execution*: to invoke the execution of a Tool available in the Cloud.

Once the JS4Cloud workflow code has been submitted, an interpreter translates the workflow into a set of concurrent tasks by analysing the existing dependencies in the code. The main benefits of JS4Cloud are:

1) it extends the well-known JavaScript language while using only its basic functions (arrays, functions, loops);
2) it implements both a data-driven task parallelism that automatically spawns ready-to-run tasks to the Cloud resources, and data parallelism through an array-based formalism;

3) these two types of parallelism are exploited implicitly so that workflows can be programmed in a totally sequential way, which frees users from duties like work partitioning, synchronization and communication.

Figure 3 shows the script-based workflow version of the visual workflow shown in Figure 2. In this example, parallelism is exploited in the for loop at line 7, where up to 16 instances of the J48 classifier are executed in parallel on 16 different partitions of the training sets, and in the for loop at line 10, where up to 16 instances of the Predictor tool are executed in parallel to classify the test set using 16 different classification models.

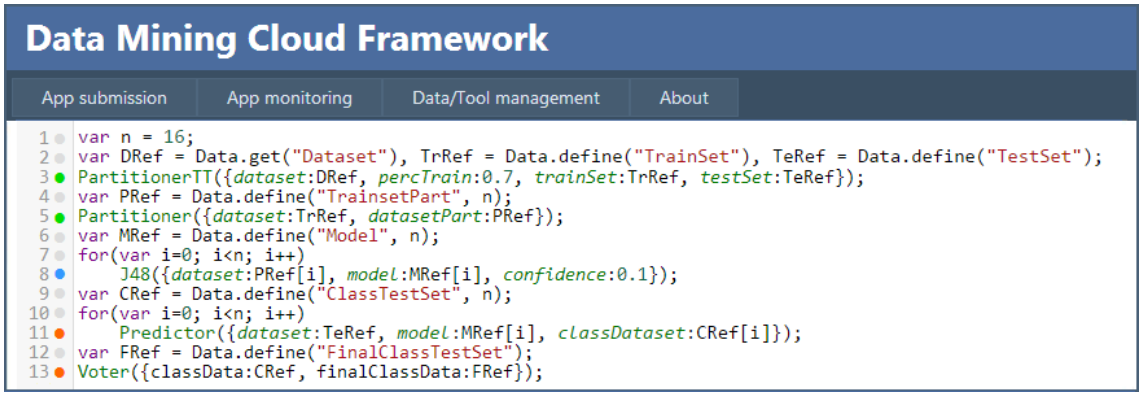

Figure 3: *Example of data analysis application designed using JS4Cloud.*

Figure 3 shows a snapshot of the parallel classification workflow taken during its execution in the DMCF's user interface. Beside each code line number, a colored circle indicates the status of execution. This feature allows user to monitor the status of the workflow execution. Green circles at lines 3 and 5 indicate that the two partitioners have completed their execution; the blue circle at line 8 indicates that J48 tasks are still running; the orange circles at lines 11 and 13 indicate that the corresponding tasks are waiting to be executed.

3. Extending VS4Cloud/JS4Cloud with MapReduce

In this section, we describe how the DMCF has been extended to include the execution of MapReduce tasks. In particular, we describe the MapReduce programming model, why it is widely used by data specialists, and how the DMCF's languages have been extended to support MapReduce applications.

3.1 Motivations

MapReduce [7] is a programming model developed by Google to process large amounts of data. It is inspired by the *map* and *reduce* primitives present in Lisp and other functional languages. A user defines a MapReduce application in terms of a map function that processes a *(key, value)* pair to generate a list of intermediate *(key, value)* pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Most MapReduce implementations are based on a master-slave architecture. A job is submitted by a user node to a master node that selects idle workers and assigns a map or reduce task to each one. When all the tasks have been completed, the master node returns the result to the user node.

MapReduce and its best-known implementation Hadoop[2] have become widely used by data specialists to develop parallel applications that analyze big amount of data. Hadoop is designed to scale up from a single server to thousands of servers, and has become the focus of several other projects, including *Spark*[3] for in-memory machine learning and data analysis, *Storm*[4] for streaming data analysis, *Hive*[5] as data warehouse software to query and manage large datasets, and *Pig*[6] as dataflow language for exploring large datasets.

Algorithms and applications written using MapReduce are automatically parallelized and executed on a large number of servers. Consequently, MapReduce has been widely used to implement data mining algorithms in parallel. Chu et al. [8] offer an overview of how several learning algorithms can be efficiently implemented using MapReduce. More in details, the authors demonstrate that MapReduce shows basically a linear speedup with an increasing number of processors on a variety of learning algorithms such as K-means, neural networks and Expectation-Maximization probabilistic clustering. Mahout[7] is an Apache project built on Hadoop that provides scalable machine learning libraries. Ricardo project [9] is a platform that integrate R[8] statistical tools and Hadoop to support parallel data analysis. The use of MapReduce for data intensive scientific analysis and bioinformatics is deeply analyzed in [10].

For the reasons discussed above and for the large number of MapReduce algorithms and applications available online, we designed an extension of the DMCF's workflow formalism to support also the execution of MapReduce tools.

3.2 Integration Model

In DMCF, a Tool represents a software tool or service performing any kind of process that can be applied to a data element (data mining, filtering, partitioning, etc.).

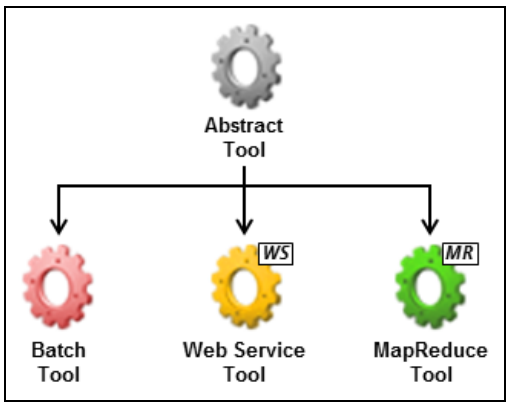

Figure 4: *Types of Tools available in DMCF.*

As shown in Figure 4, three different types of Tools can be used in a DCMF workflow:

[2] http://hadoop.apache.org
[3] http://spark.apache.org
[4] http://storm.apache.org
[5] http://hive.apache.org
[6] http://pig.apache.org
[7] http://mahout.apache.org
[8] http://www.r-project.org

- A *Batch Tool* is used to execute an algorithm or a software tool on a Virtual Compute Server without user interaction. All input parameters are passed as command-line arguments.
- A *Web Service Tool* is used to insert into a workflow a Web service invocation. It is possible to integrate both REST and SOAP-based Web services [11].
- A *MapReduce Tool* is used to insert into a workflow the execution of a MapReduce algorithm or application running on a cluster of virtual servers.

For each Tool in a workflow, a *Tool descriptor* includes a reference to its executable, the required libraries, and the list of input and output parameters. Each parameter is characterized by *name*, *description*, *type*, and can be *mandatory* or *optional*.

In more detail, a MapReduce Tool descriptor is composed by two groups of parameters: *generic parameters,* which are parameters used by the MapReduce runtime, and *applications parameters*, which are parameters associated to specific MapReduce applications. In the following, we list a few examples of generic parameters:

- *mapreduce.job.reduces*: the number of reduce tasks per job;
- *mapreduce.job.maps:* the number of map tasks per job;
- *mapreduce.input.fileinputformat.split.minsize*: the minimum size of chunk that map input should be split into;
- *mapreduce.input.fileinputformat.split.maxsize*: the maximum size of chunk that map input should be split into;
- *mapreduce.map.output.compress:* enable the compression of the intermediate mapper outputs before being sent to the reducers;

Figure 5 shows an example of MapReduce Tool descriptor for an implementation of the Random Forest algorithm. As shown by the descriptor, the algorithm can be configured with the following parameters: a set of input files (*dataInput*), the number of trees that will be generated (*nTrees*), the minimum number of elements for node split (*minSplitNum*), the column containing the class labels (*classColumn*), and the output models (*dataOutput*). DMCF uses this descriptor to allow the inclusion of a RandonForest algorithm in a workflow, and to execute it on a MapReduce cluster.

```
"MapReduceRandomForest": {
    "libraryList": ["java.exe", "MapReduceRandomForest"],
    "executable": "java.exe -jar .... ",
    "parameterList": [{
        "name": "dataset", "flag": "-in",
        "mandatory": true, "parType": "IN",
        "type": "file", "array": false,
        "description": "Input file or folder"
    },{
        "name": "nTrees", "flag": "-nTrees",
        "mandatory": false, "parType": "OP",
        "type": "int", "array": false,
        "description": "The number of trees",
        "value": "100"
    },{
        "name": "minSplitNum", "flag": "-minSplitNum",
        "mandatory": false, "parType": "OP",
        "type": "int", "array": false,
        "description": "Minimum number of elements for node split",
        "value": "2"
    },{
        "name": "classColumn", "flag": "-class",
        "mandatory": false, "parType": "OP",
        "type": "int", "array": false,
        "description": "Column index to use as the class",
        "value": ""
    }
    ...
    ,{
        "name": "output", "flag": "-out",
        "mandatory": true, "parType": "OUT",
        "type": "file", "array": false,
        "description": "Output file or folder"
    }]
}
```

Figure 5: *Example of MapReduce descriptor in JSON format.*

4. A Data Mining Application Case

In this section, we describe a DMCF data mining application whose workflow includes MapReduce computations. Through this example, we show how the MapReduce paradigm has been integrated into DMCF workflows, and how it can be used to exploit the inherent parallelism of the application. The application deals with a significant economic problem coupled with the flight delay prediction. Every year approximately 20% of airline flights are delayed or canceled mainly due to bad weather, carrier equipment or technical airport problems. These delays result in significant cost to both airlines and passengers. In fact, the cost of flight delays for US economy was estimated to be $32.9 billion in 2007 and more than half of it was charged to passengers [12].

The goal of this application is to implement a predictor of the arrival delay of scheduled flights due to weather conditions. The predicted arrival delay takes into consideration both implicit flight information (origin airport, destination airport, scheduled departure time, scheduled arrival time) and weather forecast at origin and destination airports. In particular, we consider the closest weather observation at origin and destination airports based on scheduled flight departure and arrival time. If the predicted arrival delay of a scheduled flight is less than a given threshold, it is classified as an on-time flight; otherwise, it is classified as a delayed flight.

Two open datasets of airline flights and weather observations have been collected, and exploratory data analysis has been performed to discover initial insights, evaluate the quality of data, and identify potentially interesting subsets. The first dataset is the *Airline On-Time Performance* (*AOTP*) provided by RITA - Bureau of Transportation

Statistics[9]. The AOTP dataset contains data for domestic US flights by major air carriers, providing for each flight detailed information such as origin and destination airports, scheduled and actual departure and arrival times, air time, and non-stop distance. The second is the *Quality Controlled Local Climatological Data* (*QCLCD*) dataset available from the National Climatic Data Center[10]. The dataset contains hourly weather observations from about 1,600 U.S. stations. Each weather observation includes data about temperature, humidity, wind direction and speed, barometric pressure, sky condition, visibility and weather phenomena descriptor.

For data classification, a MapReduce version of the Random Forest (RF) algorithm has been used. RF is an ensemble learning method for classification [13]. It creates a collection of different decision trees called *forest*. Once forest trees are created, the classification of an unlabeled tuple is performed by aggregating the predictions of the different trees through majority voting.

The results presented for this application have been obtained using data for a five-year period beginning on January 2009 and ending on December 2013. Data are actually too large to be analyzed on a single server. In fact, the joint table for five years data is larger than 120GB, which cannot be analyzed on a single server due to memory limits. A cloud system makes the analysis possible by providing the necessary computing resources and scalability. In addition, the cloud makes the proposed process more general: in fact, if the amount of data increases (e.g., by extending the analysis to many years of flight and weather data), the cloud provides the required resources with a high level of elasticity, reliability, and scalability. More application details are described in [14].

Using DMCF, we created a workflow for the whole data analysis process (see Figure 6).

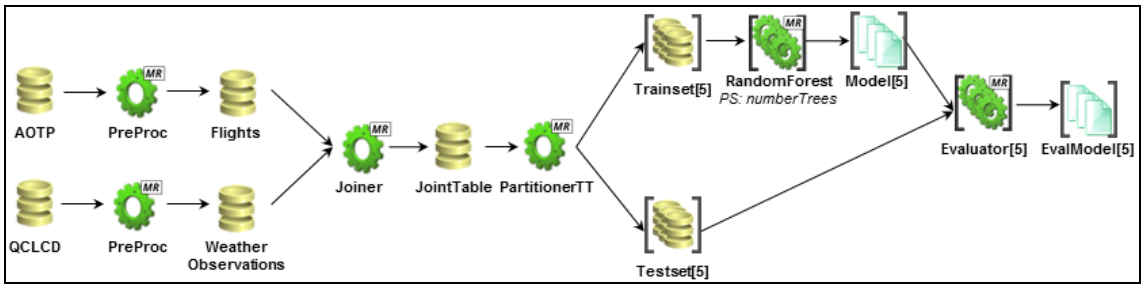

Figure 6: *Flight delay analysis workflow using DMCF with MapReduce.*

The workflow begins by pre-processing the AOTP and the QCLCD datasets using two instances of *PreProc* Tool. These steps allow looking for possible wrong data, treating missing values, and filtering out diverted and cancelled flights and weather observations not related to airport locations. Then, a *Joiner* Tool executes a relational join between *Flights* and *Weather Observations* data in parallel using a MapReduce algorithm. The result is a *JointTable*. Then, a *PartionerTT* Tool creates five pairs of <*Trainset*, *Testset*> using different delay threshold values. The five instances of training set and test set are represented in the workflow as two data array nodes, labelled as *Trainset[5]* and *Testset[5]*.

Then, five instances of the *RandomForest* Tool analyze in parallel the five instances of *Trainset* to generate five models (*Model[5]*). For each model, an instance of the *Evaluator* Tool generates the confusion matrix (*EvalModel*), which is a commonly used method to measure the quality of classification. Starting from the set of confusion

[9] http://www.transtats.bts.gov
[10] http://cdo.ncdc.noaa.gov/qclcd/QCLCD

matrices obtained, these tools calculate some metrics, e.g., accuracy, precision, recall, which can be used to select the best model.

For our experiments, we deployed a Hadoop cluster over the Virtual Computer Servers of DMCF. The cluster includes 1 head node having eight 2.2 GHz CPU cores and 14 GB of memory, and 8 worker nodes having four 2.2 GHz CPU cores and 7 GB of memory. Table 1 presents the workflow's turnaround times and speedup values obtained using up to 8 workers. Taking into account the whole workflow, the turnaround time decreases from about 7 hours using 2 workers, to 1.7 hours using 8 workers, with a speedup that is very close to linear values.

Tool	2 workers (1x)		4 workers (2x)		8 workers (4x)	
	Turnaround time (hh.mm.ss)	Speedup	Turnaround time (hh.mm.ss)	Speedup	Turnaround time (hh.mm.ss)	Speedup
PreProc	00.08.34	-	00.04.13	2.0	00.02.31	3.4
Filter	03.00.21	-	01.36.39	1.9	00.46.45	3.9
PartitionerTT	02.14.06	-	01.06.59	2.0	00.33.19	4.0
RandomForest	00.30.27	-	00.15.22	2.0	00.07.51	3.9
Evaluator	01.00.44	-	00.31.26	1.9	00.16.07	3.8
Total	06.54.12	-	03.34.39	1.9	01.46.33	3.8

Table 1: *Turnaround time and relative speedups (with respect to 2 workers)*

Scalability is obtained exploiting the parallelism offered both by MapReduce Tools and by the DMCF workflow languages. In the first case, each MapReduce Tool is executed in parallel exploiting the cluster resources. The level of parallelism depends on the number of map and reduce tasks and on the resources available in the cluster. In the second case, the DMCF workflow languages allow creating parallel paths and array of tools that can be executed concurrently. In this case, the level of the parallelism depends on the dependencies among tasks and on the resources available in the cluster.

5. Related work

Several systems have been proposed to design and execute workflow-based applications [15], but only some of them currently work on the Cloud and support visual or script-based workflow programming. The most known systems are Taverna [16], Orange4WS [17] [18], Kepler [19], E-Science Central (e-SC) [20], ClowdFlows [21], Pegasus [22] [23], WS-PGRADE [24] and Swift [25]. In particular, Swift is a parallel scripting language that executes workflows across several distributed systems, like clusters, Clouds, grids, and supercomputers. It provides a functional language in which workflows are modelled as a set of program invocations with their associated command-line arguments, input and output files. Swift uses a C-like syntax consisting of function definitions and expressions that provide a data-driven task parallelism. The runtime includes a set of services that implement the parallel execution of Swift scripts exploiting the maximal concurrency permitted by data dependencies within a script and by external resource availability. Swift users can use Galaxy [26] to provide a visual interface for Swift [27].

For comparison purposes, we distinguish two types of parallelism levels: workflow parallelism, which refers to the ability of executing multiple workflows concurrently; and task parallelism, which is the ability of executing multiple tasks of the same workflow concurrently. Most systems, including DMCF, support both workflow and

task parallelisms, except for ClowdFlows and E-Science Central that focus on workflow parallelism only.

Most systems are provided according with the SaaS model (e.g., E-Science Central, ClowdFlows, Pegasus, WS-PGRADE, Swift+Galaxy and DMCF), whereas Taverna, Kepler and Orange4WS are implemented as desktop applications that can invoke Cloud software exposed as Web Services. 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.

DMCF is one of the few SaaS systems featuring both workflow/task parallelism and support to data/tool arrays. However, differently from the data/tool array formalisms provided by the other systems, DMCF's arrays make explicit the parallelism level of each workflow node, i.e., the number of input/output datasets (in case of data arrays) and the number of tools to be concurrently executed (in case of 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.

6. Conclusions

Data analysis applications often involve big data and complex software systems in which multiple data processing tools are executed in a coordinated way. Big data refers to massive, heterogeneous, and often unstructured digital content that is difficult to process using traditional data management tools and technique. Cloud computing systems provide elastic services, high performance and scalable data storage, which can be used as large-scale computing infrastructures for complex high-performance data mining applications.

Data analysis workflows are effective in expressing task coordination and can be designed through visual and script-based formalisms. According to this approach, we described the Data Mining Cloud Framework (DMCF), a system supporting the scalable execution of data analysis computations on Cloud platforms. A workflow in DMCF can be defined using a visual or a script-based formalism, in both cases implementing a data-driven task parallelism that spawns ready-to-run tasks to Cloud resources.

In this chapter, we presented how the DMCF workflow paradigm has been integrated with the MapReduce model. In particular, we described how VL4Cloud/JS4Cloud workflows can include MapReduce algorithms and tools, and how these workflows are executed in parallel on DMCF to enable scalable data processing on Clouds.

We described a workflow application that exploits the support to MapReduce provided by DMCF. The goal of this workflow is implementing a predictor of the arrival delay of scheduled flights due to weather conditions, taking into consideration both implicit flight information and weather forecast at origin and destination airports. By executing the workflow on an increasing number of workers, we were able to achieve a nearly linear speedup, thanks to the combined scalability provided by the DMCF workflows languages and by the MapReduce framework.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. Commun. ACM, 53(4), pp. 50–58, April 2010.

[2] D. Talia, P. Trunfio. Service-Oriented Distributed Knowledge Discovery, Chapman & Hall/CRC, USA, 2012.

[3] F. Marozzo, D. Talia, P. Trunfio. A Cloud Framework for Parameter Sweeping Data Mining Applications. Proc. of the 3rd International Conference on Cloud Computing Technology and Science (CloudCom 2011), Athens, Greece, pp. 367-374, 2011.

[4] F. Marozzo, D. Talia, P. Trunfio. Using clouds for scalable knowledge discovery applications. In Euro-Par 2012: Parallel Processing Workshops, pp. 220-227, 2013.

[5] F. Marozzo, D. Talia, and P. Trunfio. JS4Cloud: Script-based workflow programming for scalable data analysis on cloud platforms. Concurrency and Computation: Practice and Experience, 2015.

[6] F. Marozzo, D. Talia, and P. Trunfio. A cloud framework for big data analytics workflows on Azure. In Proc. of the 2012 High Performance Computing Workshop (HPC 2012), 2012.

[7] J. Dean, S. Ghemawat. MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), pp. 107-113, 2008.

[8] C. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. Advances in neural information processing systems, 19 (2007), pp. 281, 2007.

[9] S. Das, Y. Sismanis, K. S. Beyer, R. Gemulla, P. J. Haas, and J. McPherson. Ricardo: Integrating R and Hadoop. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10). ACM, New York, NY, USA, pp. 987–998, 2010.

[10] J. Ekanayake, S. Pallickara, and G. Fox. MapReduce for Data Intensive Scientific Analyses. In Proceedings of the 2008 Fourth IEEE International Conference on eScience (ESCIENCE '08). IEEE Computer Society, Washington, DC, USA, pp. 277–284, 2008.

[11] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web* (WWW '08). ACM, New York, NY, USA, pp. 805-814, 2008 April.

[12] M. Ball, C. Barnhart, M. Dresner, M. Hansen, K. Neels, A. Odoni, E. Peterson, L. Sherry, A. A. Trani, and B. Zou. Total delay impact study: a comprehensive assessment of the costs and impacts of flight delay in the United States, 2010.

[13] L. Breiman. Random forests. Machine learning, Springer, 45(1), pp. 5–32, 2001.

[14] L. Belcastro, F. Marozzo, D. Talia, and P. Trunfio. Using Scalable Data Mining for Predicting Flight Delays. 2015. Under Review

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

[16] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble, "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud," Nucleic Acids Research, vol. 41, no. W1, pp. W557–W561, July 2013.

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

[18] J. Demšar, T. Curk, A. Erjavec, Črt Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, and B. Zupan, "Orange: Data mining toolbox in python," Journal of Machine Learning Research, vol. 14, pp. 2349–2353, 2013. [Online]. Available: http://jmlr.org/papers/v14/demsar13a.html

[19] 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," Concurrency and Computation: Practice and Experience, vol. 18, no. 10, pp. 1039–1065,2006.

[20] 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, January 2013.

[21] 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. Heidelberg, Germany: Springer, 2012, vol. 7524, pp. 816–819.

[22] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good et al., "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," Scientific Programming, vol. 13, no. 3, pp. 219–237, 2005.

[23] 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 International Conference for, ser. SC '10. IEEE, November 2010, pp. 1–9.

[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, Dec. 2012.

[25] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," Parallel Computing, vol. 37, no. 9, pp. 633–652, September 2011.

[26] B. Giardine, C. Riemer, R. C. Hardison, R. Burhans, P. Shah, Y. Zhang, D. Blankenberg, I. Albert, W. Miller, W. J. Kent, and A. Nekrutenko, "Galaxy: A platform for interactive large-scale genome analysis," Genome Res, vol. 15, pp. 1451–1455, 2005.

[27] 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 Cluster Computing (CLUSTER), 2013 IEEE International Conference on, Sept 2013, pp. 1–3.