A Workflow-oriented Language for Scalable Data Analytics

Fabrizio Marozzo, Domenico Talia, Paolo Trunfio

DIMES - University of Calabria, Italy
fmarozzo@dimes.unical.it, talia@dimes.unical.it, trunfio@dimes.unical.it

Abstract

Data in digital repositories are everyday more and more massive and distributed. Therefore analyzing them requires efficient data analysis techniques and scalable storage and computing platforms. Cloud computing infrastructures offer an effective support for addressing both the computational and data storage needs of big data mining and parallel knowledge discovery applications. In fact, complex data mining tasks involve data- and compute-intensive algorithms that require large and efficient storage facilities together with high performance processors to get results in acceptable times. In this paper we describe a Data Mining Cloud Framework (DMCF) designed for developing and executing distributed data analytics applications as workflows of services. We describe also a workflow-oriented language, called JS4Cloud, to support the design and execution of script-based data analysis workflows on DMCF. We finally present a data analysis application developed with JS4Cloud, and the scalability achieved executing it on DMCF.

Keywords Cloud computing, Data analytics, Workflows, JS4Cloud

I. Introduction

Cloud computing provides elastic services, high performance and scalable data storage to a large and everyday increasing number of users [1]. Clouds enlarged the offer of distributed computing systems by providing advanced Internet services that complement and complete functionalities of distributed computing provided by the Web, Grid computing and peer-to-peer networks. In fact, Cloud computing systems provide large-scale infrastructures for complex high-performance applications. Most of those applications use big data repositories and needs to 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 [5].

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, EMC, BigML, Splunk Storm, Kognitio, and InsightsOne. As more such platforms emerge, researchers will port increasingly powerful data mining programming tools and strategies to the cloud to exploit complex and flexible software models such as the distributed workflow paradigm.

The growing use of service-oriented computing could accelerate 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 delivery data analysis software that is called data analytics as a service (DAaaS).

Here we describe a Data Mining Cloud Framework (DMCF) that we developed according to this approach. In DMCF, data analysis workflows can be designed through visual programming, which is a very effective design approach for high-level users, e.g. domain-expert analysts having a limited understanding of programming. Recently, we extended the DMCF system to support also script-based data analysis workflows, as an additional and more flexible programming interface for skilled users. To this end, in [4] we introduced a workflow-oriented language, called JS4Cloud, to support the design and execution of script-based data analysis workflows on DMCF.

II. Data Mining Cloud Framework

The DMCF has been designed to be implemented on different Cloud systems. However, a first implementation of the framework has been carried out on the Windows Azure cloud platform and has been evaluated through a set of data analysis applications executed on a Microsoft Cloud data center. The remainder of the section describes system architecture, application execution, user interface, and visual workflow programming.

II.1 System architecture

The architecture includes different kinds of components that can be grouped into storage and compute components (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.

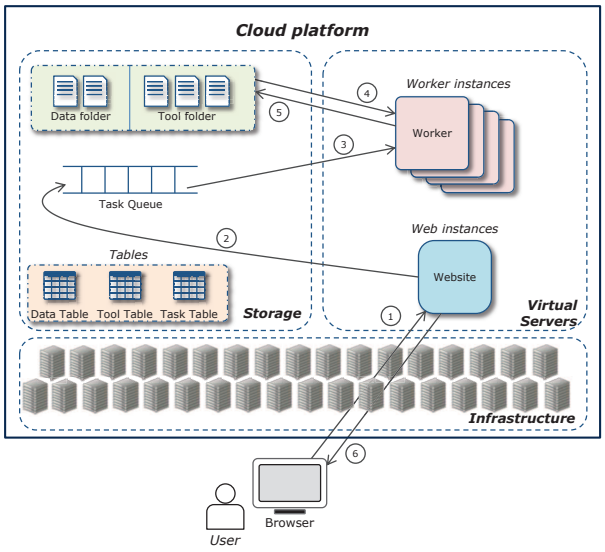


Figure 1: System architecture and application execution steps.

The virtual machines components are:

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

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

II.2 Applications execution

Figure 1 shows the main steps carried out for designing and executing a knowledge discovery application:

1. A user accesses the Website and designs the application (either single-task, parameter sweeping, or workflow-based) through a Web-based interface.
2. After application submission, the system creates a set of tasks and inserts them into the Task Queue on the basis of the application.
3. Each idle Worker picks a task from the Task Queue, and concurrently executes it.
4. Each Worker gets the input dataset from the location specified by the application. To this end, a file transfer is performed from the Data Folder where the dataset is located, to the local storage of the Worker.
5. After task completion, each Worker puts the result on the Data Folder.
6. The 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 the user submits a parameter sweeping application, the set of tasks corresponding to the combinations of the input parameters values are executed in parallel. If a complex workflow-based application must be executed, the set of tasks created depends on how many data mining tools are invoked within the workflow. Initially, only the workflow tasks without dependencies are inserted into the Task Queue.

II.3 User interface

The App submission section of the Website is composed of two main parts: one pane for composing and running both single-task and parameter-sweeping applications and another pane for programming and executing workflow-based knowledge discovery applications. As an example, Figure 2 shows a screenshot of the App submission section, taken during the execution of a parameter-sweeping application.

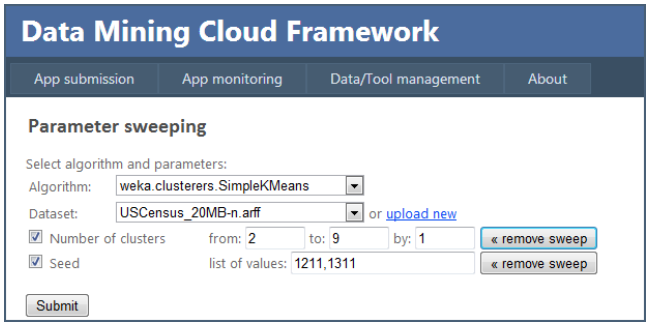


Figure 2: Screenshot of the App submission section.

Users can monitor the status of each single task through the App monitoring section, as shown in Figure 3. For each task, the current status (submitted, running, done or failed) and status update time are shown. Moreover, for each task that has completed its execution, two links are enabled: the first one (Stat) gives access to a file containing some statistics about the amount of resources consumed by the task; the second one (Result) visualizes the task result.

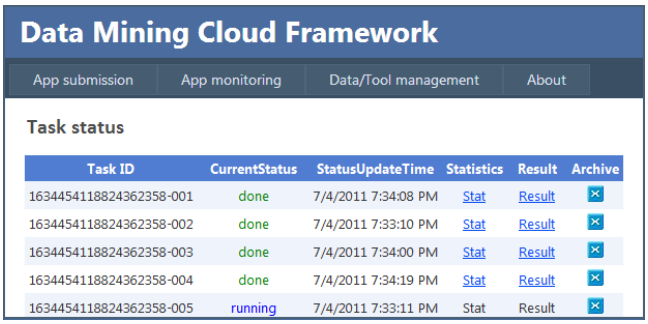


Figure 3: Screenshot of the App monitoring section.

II.4 Visual workflow programming

The DMCF includes a visual programming interface and its services to support the composition and execution of workflow-based knowledge discovery applications. 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, knowledge discovery workflows can produce results that can confirm real experiments or provide insights that cannot be achieved in laboratories.

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

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

The nodes can be connected with each other through direct edges, establishing specific dependency relationships among them. When an edge is being created between two nodes, a label is automatically attached to it representing the kind of relationship between the two nodes. Data and Tool nodes can be added to the workflow singularly or in array form. A data array is an ordered collection of input/output data elements, while a tool array represents multiple instances of the same tool.

Figure 4 shows a data mining workflow composed of several sequential and parallel steps as an example for presenting the main features of the visual programming interface of the DMCF [3]. The example workflow analyzes a dataset by using several instances of a classification algorithm that run in parallel on several cloud servers.

III. Script-based workflow programming

JS4Cloud (JavaScript for Cloud) is a JavaScript-based language for programming data analysis workflows [4]. The Web interface of DMCF allows to design and execute workflows programmed by the JS4Cloud language, by providing an environment similar to that used to develop visual workflows in the same framework.

The main benefits of JS4Cloud are: *i*) it is based on a well known scripting language, so that users do not have to learn a new programming language from scratch; *ii*) it implements a data-driven task parallelism that automatically spawns ready-to-run tasks to the available Cloud resources; *iii*) it exploits implicit parallelism so application workflows can be programmed in a totally sequential way.

Two key programming abstractions in JS4Cloud are *Data* and *Tool* elements:

- *Data* elements denote input files or storage elements, or output files or stored elements.
- *Tool* elements denote algorithms or software tools.

For each Data and Tool element included in a JS4Cloud workflow, an associated descriptor, expressed in JSON format, will be included in the environment of the user who is developing the 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. The JSON descriptor of a new tool is created automatically through a guided procedure provided by DMCF, which allows users to specify all the needed information for invoking the tool (executable, input and output parameters, etc.).

Similarly, a Data descriptor contains information to access an input or output file, including its identifier, location, and format. Differently from Tool descriptors, Data descriptors can also be created dynamically as a result of a task operation during the execution of a JS4Cloud script. For example, if a workflow W reads a dataset D_i and creates (writes) a new dataset D_j, only D_i's descriptor will be present in the environment before W's execution, whereas D_j's descriptor will be created at runtime.

Another key element in JS4Cloud is the *task* concept, which represents the unit of parallelism in our model. A task is a Tool, invoked from the script code, which is intended to run in parallel with other tasks on a set of Cloud resources.

According to this approach, JS4Cloud implements *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.

III.1 JS4Cloud functions

JS4Cloud extends JavaScript with three additional functionalities, implemented by the set of functions listed in Table 1:

- *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.

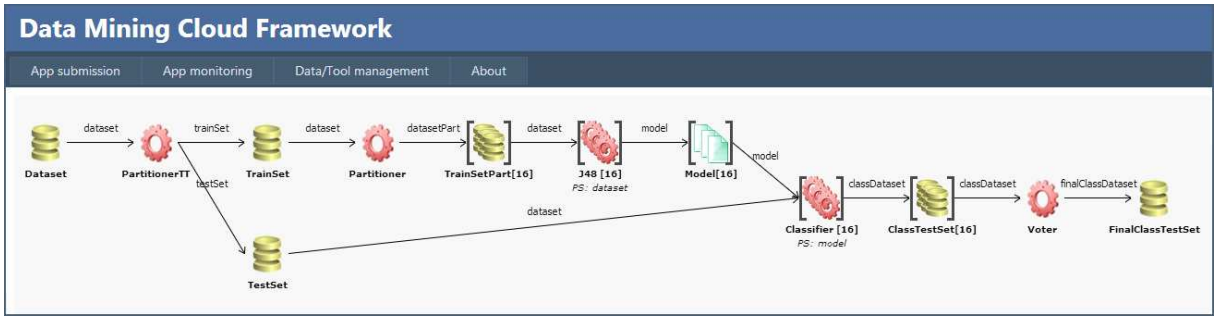


Figure 4: A visual workflow for parallel classification.

Table 1: JS4Cloud functions.

Functionality	*Function*	*Description*
Data Access	Data.get(<*dataName*>);	Returns a reference to the data element with the provided name.
	Data.get(new RegExp(<*regular expression*>));	Returns an array of references to the data elements whose name match the regular expression.
Data Definition	Data.define(<*dataName*>);	Defines a new data element that will be created at runtime.
	Data.define(<*arrayName*>,<*dim*>);	Define an array of data elements.
	Data.define(<*arrayName*>,[<dim_1>,...,<dim_n>]);	Define a multi-dimensional array of data elements.
Tool Execution	<*toolName*>(<par_1>:<val_1>,...,<par_n>:<val_n>);	Invokes an existing tool with associated parameter values.

Data Access is implemented by the `Data.get` function, which is available in two versions: the first one receives the name of a data element, and returns a reference to it; the second one returns an array of references to the data elements whose name match the provided regular expression. For example, the following statement:

```
var ref = Data.get("Census");
```

assigns to variable `ref` a reference to the dataset named `Census`, while the following statement:

```
var ref = Data.get(new RegExp("^CensusPart"));
```

assigns to `ref` an array of references (`ref[0]...ref[n-1]`) to all the datasets whose name begins with `CensusPart`.

Data Definition is done through the `Data.define` function, available in three versions: the first one defines a single data element; the second one defines a one-dimensional array of data elements; the third one defines a multi-dimensional array of data elements. For instance, the following piece of code:

```
var ref = Data.define("CensusModel");
```

defines a new data element named `CensusModel` and assigns its reference to variable `ref`, while the following statement:

```
var ref = Data.define("CensusModel", 16);
```

defines an array of data elements of size 16 (`ref[0]... ref[15]`). In both cases, the data elements will be created at runtime as result of a tool execution.

Differently from Data Access and Data Definition, there is not a named function for Tool Execution. In fact, the invocation of a tool *T* is made by calling a function with the same name of *T*. For example, the following statement:

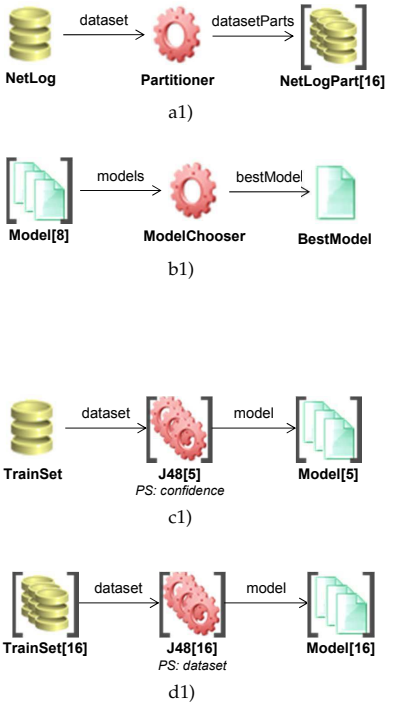
```
DTree({dataset:DRef, confidence:0.05, model:MRef});
```

invokes a tool named DTree, where `DRef` is a reference to the dataset to be analyzed, previously introduced using the `Data.get` function, `MRef` is a reference to the model to be generated, previously introduced using `Data.define`.

III.2 Basic patterns

Several workflow patterns can be implemented with JS4Cloud [4]. Figure 5 shows four examples of patterns that can be defined in JS4Cloud workflows, namely data partitioning, data aggregation, parameter sweeping and input sweeping. For each pattern, the figure shows an example as a visual DMCF workflow, and how the same example can be coded using JS4Cloud.

The *data partitioning* pattern produces two or more output data from an input data element, as in Figure 5-a1, where a Partitioner tool divides a dataset into a number of splits. With JS4Cloud, this can be written as shown in Figure 5-a2.




```
var DRef = Data.get("NetLog");
var PRef = Data.define("NetLogParts", 16);
Partitioner({dataset:DRef, datasetParts:PRef});
```

a2)

```
var BMRef = Data.define("BestModel");
ModelChooser({models:MsRef, bestModel:BMRef});
```

b2)

```
var TRef = Data.get("TrainSet");
var nMod = 5;
var MRef = Data.define("Model", nMod);
var min = 0.1;
var max = 0.5;
for(var i=0; i<nMod; i++)
   J48({dataset:TRef, model:MRef[i],
      confidence:(min+i*(max-min)/(nMod-1))});
```

c2)

```
var nMod = 16;
var MRef = Data.define("Model", nMod);
for(var i=0; i<nMod; i++)
   J48({dataset:TsRef[i], model:MRef[i],
      confidence:0.1});
```

d2)

Figure 5: Visual (left) and JS4Cloud (right) workflow patterns: a) data partitioning; b) data aggregation; c) parameter sweeping; d) input sweeping.

The *data aggregation* pattern generates one output data from multiple input data, as in Figure 5-b1, where a ModelChooser tool takes as input eight data mining models and chooses the best one based on some evaluation criteria. The same task can be coded using JS4Cloud as shown in Figure 5-b2.

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-c1. In this example, a training set is processed in parallel by 5 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. The equivalent JS4Cloud script is shown in Figure 5-c2.

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-d1. In this example, 16 training sets are processed in parallel by 16 instances of J48, to produce the same number of data mining models. The corresponding JS4Cloud script is shown in Figure 5-d2.

III.3 Example of JS4Cloud workflow

We describe a JS4Cloud workflow that analyzes a dataset using n instances of the J48 classification algorithm that work on n partitions of the training set and generate n knowledge models. By using the n generated models and the test set, n classifiers produce in parallel n classified datasets (n classifications). In the final step of the workflow, a voter generates the final classification (in the file `FinalClassTestSet`) by assigning a class to each data item. This is done by choosing the class predicted by the majority of the models [6].

The input dataset, containing about 46 million tuples and with a size of 5 GB, was generated from the *KDD Cup 1999*'s dataset, which contains a wide variety of simulated intrusion records in a military network environment.

Figure 6 shows the JS4Cloud code of the workflow. At the beginning, the input dataset is split into training set and test set by a partitioning tool (*line 3*). Then, the training set is partitioned into 64 parts using another partitioning tool (*line 5*). As third step, the training sets are analyzed in parallel by 64 instances of the J48 classification algorithm, to produce the same number of classification models (*lines 7-8*). The fourth step classifies the test set using the 64 models generated on the previous step (*lines 10-11*). The classification is performed by 64 classifiers that run in parallel to produce 64 classified test sets. As the last operation, the 64 classified test sets are passed to a voter that produces the final classified test set.

Beside each code line number, a colored circle indicates the status of execution. The 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 indicates that the corresponding tasks are waiting to be executed.

Figure 7 shows the turnaround times of the workflow, obtained

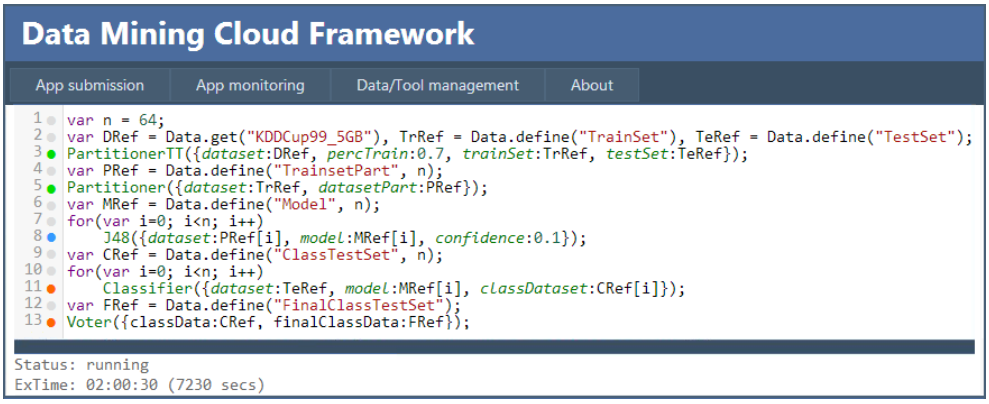


Figure 6: JS4Cloud workflow running in the DMCF's user interface.

varying the number of virtual servers used to run it on the Cloud from 1 (sequential execution) to 64 (maximum parallelism). As shown in the figure, the turnaround time decreases from more than 107 hours (4.5 days) by using a single server, to about 2 hours on 64 servers. This is an evident and significant reduction of time, with a speedup ranging from 7.64 using 8 servers to 50.78 using 64 servers. This is a very positive result, taking into account that some sequential parts of the implemented application (namely, partitioning and voting) do not run in parallel.

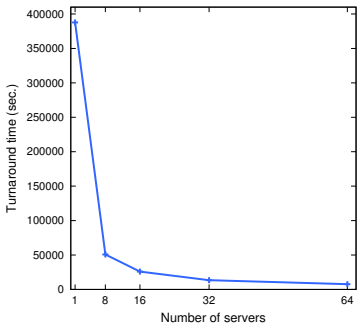


Figure 7: Turnaround time vs number of available servers.

IV. Concluding remarks

Cloud computing [2] provides scalable resources for Big data mining and parallel knowledge discovery applications. In fact, Clouds offer large and efficient storage facilities with high performance processors to get results in reduced times. In this paper we presented a Data Mining Cloud Framework (DMCF) designed for developing and running distributed data analytics applications as collections of services. In this framework, data sets, data mining algorithms and knowledge models are implemented as services that can be combined through a visual interface to produce distributed workflows executed on Clouds.

Recently, we extended the DMCF system to support also script-based data analysis workflows, as an additional and more flexible programming interface for skilled users. To this end, we introduced a workflow-oriented language, called JS4Cloud, to support the design and execution of script-based data analysis workflows on DMCF. Experimental performance results, obtained designing and executing JS4Cloud workflows in DMCF, have proven the effectiveness of the proposed language for programming data analysis workflows, as well as the scalability that can be achieved by executing such workflows on a public Cloud infrastructure.

Acknowledgment

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, 'Network for Sustainable Ultrascale Computing (NESUS)'.

References

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[2] Cloud Computing Expert Group. The future of cloud computing. Report from European Commission, January 2010.

[3] 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.

[4] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. Scalable script-based data analysis workflows on clouds. In *Proc. of the 8th Workshop on Workflows in Support of Large-Scale Science (WORKS 2013)*, pages 124–133, Denver, CO, USA, November 2013. ACM Press.

[5] Domenico Talia. Clouds for scalable big data analytics. *IEEE Computer*, 46(5):98–101, 2013.

[6] Z.-H. Zhou and M. Li. Semi-supervised learning by disagreement. *Knowl. Inf. Syst.*, 24(3):415–439, 2010.