J Grid Computing
DOI 10.1007/s10723-014-9311-x

SuMo: Analysis and Optimization of Amazon EC2 Instances

P. Kokkinos · T. A. Varvarigou · A. Kretsis · P. Soumplis · E. A. Varvarigos

Received: 29 January 2014 / Accepted: 3 September 2014

Abstract The analysis and optimization of public clouds gains momentum as an important research topic, due to their widespread exploitation by individual users, researchers and companies for their daily tasks. We identify primitive algorithmic operations that should be part of a cloud analysis and optimization tool, such as resource profiling, performance spike detection and prediction, resource resizing, and others, and we investigate ways the collected monitoring information can be processed towards these purposes. The analyzed information is valuable in driving important virtual resource management decisions. We also present an open-source tool we developed, called ***SuMo***,which contains the necessary functionalities for collecting monitoring data from Amazon Web Services (AWS), analyzing them and providing resource optimization suggestions. SuMo makes easy for anyone to analyze AWS instances behavior, incorporating a set of basic modules that provide profiling and spikef detection functionality. It can also be used as a basis for the development of new such analytic procedures for AWS. SuMo contains a Cost and Utilization Optimization (CUO) mechanism, formulated as an Integer Linear Programming (ILP) problem, for optimizing the cost and the utilization of a set of running Amazon EC2 instances. This CUO mechanism receives information on the currently used set of instances (their number, type, utilization) and proposes a new set of instances for serving the same load that minimizes cost and maximizes utilization and performance efficiency.

Keywords Public clouds · Analysis · Optimization · Amazon web services · Toolkit

P. Kokkinos (✉) · T. A. Varvarigou
Department of Electrical and Computer Engineering
National Technical University of Athens, Athens, Greece
e-mail: kokkinop@ceid.upatras.gr

A. Kretsis · P. Soumplis · E. A. Varvarigos
Department of Computer Engineering and Informatics,
University of Patras, Patra, Greece

1 Introduction

Cloud computing [1, 2] provides resources as a service over the network, enabling their efficient and flexible management. An increasing number of individual users, researchers and companies, established or startups, of any size and scope, trust their computing and storage tasks to public and private clouds, replacing fixed Information Technology (IT) costs of ownership and operation with variable use-dependent costs. Public clouds provide resizable compute capacity as a public service (e.g., Amazon Web Services – AWS [3], Rackspace [4]), while private clouds are built based on the organizations' own infrastructure, using cloud computing toolkits (e.g., OpenStack [5], OpenNebula [6], Eucalyptus [7]) and providing computing services to their employees or customers.

Published online: 03 October 2014

Monitoring, analysis and optimization are a set of important interrelated operations for cloud resource management. The sheer number of cloud resources makes it difficult for a simple user or administrator to effectively monitor and analyze their behavior or control (optimize) the parameters that determine their proper use. Estimates in [8] place the number of servers in commercial data centers, used for providing public or private cloud services, to 450.000 servers, while even larger ones (in the order of 10^6 to 10^7 machines) are envisioned for the future [9]. Also, organizations may utilize hundreds of virtual instances of a public cloud provider for their operations [10]. A good analytic and optimization entity ensures that the monitored resources run uninterrupted and with acceptable performance and utilization, keeping the associated (e.g., energy) costs low, either proactively (e.g., by informing the administration for an ill behavior) or reactively (e.g., by detecting problems as they appear).

There are a number of important differences between analyzing private and public clouds, both regarding the way monitoring is performed and regarding the parameters of interest and the requirements. In private clouds the administrator has full access to the resources and is able to monitor any kind of parameter (performance, energy, etc.), with any kind of software or hardware tools and at any granularity. This is not the case for public clouds (e.g., Amazon Web Services - AWS [3]) where the users have access to their virtual resources through web or programmable interfaces that provide only specific information (e.g., state, performance), and at specific granularities (e.g., every 5 minutes). The main parameters of interest for public clouds are the cost of the resources and their utilization. Utilization directly affects the cost, since it determines how effectively the resources of the cloud provider are being used.

The cost encountered by a user of public clouds depends mainly on the pricing policy of the cloud provider, the number and types of resources used, and the resources' utilization. Today, there are signs of merchandising the virtual/cloud resources as goods (like gold, stocks, or energy). For example, recently, it became possible for a user of AWS [3] who has bought a number of computing resources, to resell the unused part of his owned resources as the needs change, such as selling capacity for projects that end before the term expires. In general, cloud economics have emerged recently as an important field, studied in a number of works [11, 12].

In our work, we investigate ways in which the monitoring information collected by a public cloud provider in general and by AWS in particular, can be used in a "smart" way to produce valuable information and actionable data for the proper and efficient use and management of the virtual resources. Towards this end a number, of basic algorithmic operations are identified, including resource profiling, performance spike detection, performance prediction and resource resizing. We also present a tool, called SuMo, that we developed to assist in cloud analysis and optimization. SuMo contains the necessary mechanisms for collecting monitoring data from AWS and it incorporates an initial set of basic algorithms (for profiling and spike detection) for analyzing them. It also contains a Cost and Utilization Optimization (CUO) mechanism for minimizing the usage cost of Amazon EC2 instances and maximizing their utilization and the performance efficiency, based on Integer Linear Programming (ILP) formulation. The proposed mechanism collects information on the type, number, utilization and other features of the current set of AWS instances running and proposes a new set of instances that could be used for serving the same load. Our experimental results show that the proposed algorithm can increase the utilization and efficiency of the infrastructure resources, while lowering the accumulated user costs. When necessary, CUO also increases resources' capacity so as to resolve possible performance bottlenecks. Also, a number of conclusions are drawn regarding the effects that instances' characteristics (capacity granularity, region of operation) have on utilization and cost. SuMo makes easy for anyone, a researcher or an administrator, to monitor his instances and run the proposed analytic mechanisms or implement new more intelligent ones. SuMo tool is open-source and available through github [47]. Currently, the development of the core SuMo functionalities has been completed and we are investigating whether SuMo could be coupled with other tools and in particular, cloud management software for federated clouds.

The remainder of the paper is organized as follows. In Section 2 we report on previous work. In Section III we describe Amazon's computing and monitoring services. In Section 4, we discuss the algorithmic

challenges posed by the analysis of cloud monitoring data. In Section 5, we present the SuMo toolkit and its constituent modules. In Section 6 we describe an Integer Linear Program (ILP) formulation used for resource re-optimization (resizing). In Section 7 we present performance results obtained using SuMo and the CUO mechanism. Finally, in Section 8 we conclude the paper.

2 Previous Work

IT monitoring and analysis has long been around, targeting the needs of physical servers and other devices (printers, switches, ups, etc.), clusters, grids [13] and small or large data centers. Clouds bring a completely new environment and introduce new requirements for IT monitoring tools, involving a very large number of heterogeneous physical and virtual resources and producing a huge amount of raw monitoring information. Also, clouds' dynamicity and flexibility introduce the need for (re-) optimizing their characteristics.

Most works on the monitoring and analysis of cloud resources assume full access to the corresponding resources and are consequently more relevant to private clouds. In particular, a number of works attempt to aggregate (or summarize) the raw monitoring data. In [19], the authors propose a scalable distributed data collection system that utilizes technologies from the semantic web in order to generate a machine readable overview of a cloud system without the need for an additional dedicated monitoring system. In [20], the authors propose a runtime model for cloud monitoring (RMCM) that provides an intuitive representation of a running cloud. Raw monitoring data gathered by multiple monitoring techniques are organized by RMCM to present a more intuitive profile of a running cloud. CloudSense, described in [14], is a switch design that performs in-network compression of monitoring data streams via compressive sensing.

The diversity of cloud providers and applications has also raised the issue of providing a middle layer for their interaction (providers and applications). The mOSAIC project [15, 39] aims at developing an open-source platform that enables applications to negotiate Cloud services as requested by their users. Using the Cloud ontology, applications will be able to specify their service requirements and communicate them to the platform via an API. The platform will implement a multi-agent brokering mechanism that looks for services matching the applications' request, and possibly composes the requested service if no direct hit is found. Other similar projects are Optimis [16, 40] and Aeolus [41]. Aoleus is an open-source Cloud management software that allows users to choose between Private, Public or Hybrid Clouds, using δ-Cloud library. The Optimis Toolkit offers a platform for Cloud service provisioning that manages the lifecycle of the service and addresses issues like risk and trust management. In [45] authors present a federated cloud management solution that utilizes cloud-brokers for various IaaS providers. In this work, among others, a monitoring service has been designed with the capability to simultaneously monitor both private and public clouds. In [46] the execution of scientific applications in federated clouds is examined, defining a related middleware architecture.

Quite recently, a number of products/services have appeared offering monitoring and analysis tools for public cloud resources, while the functionality of other already established products has been extended appropriately to meet the particular needs of private clouds. The authors in [18] discuss the design and implementation of a private cloud monitoring system (PCMONS) and argue that it is possible to deploy a private cloud within an organization using only open-source solutions and integrating it with traditional monitoring tools, such as Nagios [29]; significant development work, however, has to be carried out in order to actually integrate these tools. In [24] the authors present an approach for configuration planning based on data refinement; correlating economic goals with sound technical data. The authors also present a proof of concept tool, called CloudXplor, which can be modularly embedded in generic resource monitoring and management frameworks. Newvem [30] cloud usage analytics collects raw usage metrics from Amazon Web Services (AWS) and performs proprietary analysis on the data gathered in order to identify cost, security, utilization and availability issues. Cloudability [31] collects daily spend updates, creates predictive alerts, and records the history of the users' cloud costs, and has been designed to support several cloud services (AWS, RackSpace, Heroku, Google Apps and other). Cloudvertical [32] helps companies manage and track the cost and usage of their cloud infrastructure efficiently.

Our work and the developed toolkit (namely SuMo) cannot directly compare against EU projects like mOSAIC, Optimis and Aeolus or start-ups like Newvem, Cloudability and Cloudvertical. In a way we attempt to stand in the middle, in terms of functionality, targeted audience and ease of use. On the one hand, the aforementioned EU funded projects provide algorithms, methodologies and tools/modules covering a wide range of topics (e.g., application porting, resource management, application brokering, service provisioning, SLAs etc.), focusing less on providing a solution that any user can actually use today. On the other hand, the services offered by the aforementioned startups are very good in aggregating the collected monitoring information and presenting it in a user friendly manner to the users, leaving however significant room for adding more intelligence to the analysis of public cloud monitoring data, instead of simply forecasting virtual resource usage and associated costs [17].

SuMo utilizes the monitoring services offered by AWS (like Newvem, Cloudability and Cloudvertical do) so as to analyze and optimize the resources used (as mOSAIC also does). Multi-agent monitoring infrastructures such as those suggested by mOSAIC may be difficult to be accepted by the cloud users, since they assume that a "custom agent" is installed (by the cloud users) in each resource. On the other hand utilizing cloud providers monitoring API (like AWS CloudWatch, Section 3), as SuMo does, increases service acceptance, ease of use and ensures cloud users that their resources are well protected.

Additionally, SuMo can be used as a basis for the development of algorithms and methodologies for the analysis of collected monitoring information from AWS, while in the future it may also act as an open-source alternative to the various commercial solutions ([23–25]). In its current release SuMo includes a set of basic algorithms that provide profiling and spike detection functionality. In our work, we also present an Integer Linear Programming (ILP) formulation for optimizing the cost, the utilization and the performance of Amazon cloud resources. The corresponding mechanism (namely Cost and Utilization Optimization - CUO), included in SuMo, provides specific suggestions regarding which AWS instances' type should be modified in order to minimize the cost to the user and maximize the utilization of the resources, or suggestions on how to increase performance by resolving capacity bottlenecks. A mechanism similar to CUO was presented in [25] in the context of the mOSAIC project [39], focusing on modular applications and introducing a scheduling method for placing each application component type on every needed node, avoiding the unnecessary allocation of extra nodes and ensuring high availability. In contrast to the mOSAIC project, CUO is application-agnostic, handling a set of AWS resources (hosting one or more applications) as a whole and optimizing their usage.

Part of this paper has been presented in [26]; however, in this article we add new material, extending previous work and discussing the algorithmic challenges posed by the analysis of public cloud monitoring data. We also present in detail the SuMo toolkit and its constituent modules and elaborate more on the CUO mechanism and on its evaluation, including additional performance results.

3 Amazon Web Services – Cloud Watch

Amazon Web Services' (AWS) [1] public cloud is a collection of web services that provide access to Amazon's cloud infrastructure. AWS enables anyone to run virtually anything in the cloud: from enterprise applications and big data projects to social games and mobile apps. Amazon Elastic Computing Cloud - EC2 [27] is one of the most important such services, providing resizable compute capacity as a service. The basic unit of EC2 is the "instance", which represents a virtual resource with particular computational, storage and network characteristics, running a particular Operating System (OS) and located physically in one of Amazon's data-centers around the world. The above characteristics also determine the cost of the instance. There are two main instance types: i) On-Demand Instances (ODI) and ii) Reserved Instances (RI). With ODI a user pays for compute capacity by the hour, with no long-term commitments, while with reserved instances the user makes a one-time payment for reserving an instance and then pays again by the hour but at a significant discount. There are also Spot instances allowing one to bid on spare Amazon EC2 instances and run them whenever his bid exceeds the current spot price.

Table 1 summarizes the various instance types; there are around 196 different ODI and 1176 different RI instance types.

Table 1 Amazon EC2 instance types in numbers

14 main types of machines[a]	'm1.small', 'm1.medium', 'm1.large', 'm1.xlarge', 't1.micro', 'm2.xlarge', 'm2.2xlarge', 'm2.4xlarge', 'c1.medium', 'c1.xlarge', 'cc1.4xlarge', 'cc2.8xlarge', 'cg1.4xlarge', 'hi1.4xlarge'
7 different regions – datacenters	US East (Northern Virginia), US West (Oregon), US West (Northern California), EU (Ireland), Asia Pacific (Singapore), Asia Pacific (Tokyo), South America (Sao Paulo)
2 Operating Systems	Windows, Linux
3 RI utilization types	low, medium, high
2 RI year terms	1 or three years

a. These are actually the instance names used by AWS Application Programming Interface (API)

For example, the machine type "Extra Large Instance" (m1.xlarge) has the following characteristics:

- 15 GB memory
- 8 EC2 Compute Units – ECU: 4 virtual cores with 2 EC2 Compute Units each (according to Amazon [1], one ECU provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.)
- 1,690 GB instance storage
- 64-bit platform
- I/O Performance: High

Amazon's CloudWatch [28] provides monitoring for AWS cloud resources (such as Amazon EC2). Developers and system administrators can use it to collect and track various metrics. The basic CloudWatch metrics are the following, though customized ones can also be added:

- CPUUtilization: The percentage of allocated EC2 compute units that are currently in use by the instance
- DiskRead/WriteOps: The number of completed read/write operations from all ephemeral disks available to the instance
- DiskRead/WriteBytes: The number of bytes read/written from all ephemeral disks available to the instance
- NetworkIn/Out: The number of bytes received/sent on all network interfaces by the instance.

Monitoring data are available automatically, at 1 or 5-minute interval steps, depending on the charging policy chosen. Only 1440 points of a particular metric can be provided by CloudWatch at time; e.g., the CPU utilization of an instance per 1-minute (step size) for the last 24 hours (period). Other step sizes and periods can also be selected.

4 Algorithmic Operation for Analyzing Public Clouds

We consider three categories of algorithmic functions for analyzing monitoring data collected from (mainly public) clouds:

1. *Planning functions*: This kind of algorithmic operations help the administrator plan his future resource requirements, using as input the behavior of virtual resources till now. Planning operations are performed offline.
2. *Analytic functions*: These operations relate to the analysis of past monitoring data in order to create useful aggregated quantitative information for future use.
3. *Operational functions*: Operational functionalities make online resource management decisions or detect various kinds of "events" as they occur.

Many of these functions are signal-analysis based that is they receive as input a signal (Fig. 1) or a set of signals, representing the utilization (computational, network, storage, memory) for a particular period of time (a day, a week, a month, a year, etc.) and they analyze them using signal processing methodologies.

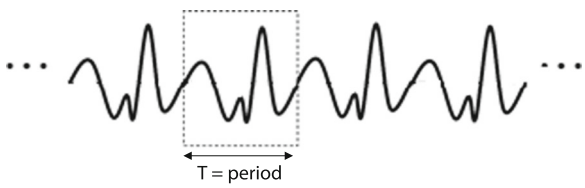

Fig. 1 Utilization of a virtual resource in a particular time period

In practice, time-continuous signals like the one shown in Fig. 1 are rather unusual – typical usage patterns are step functions with some kind of base period (e.g., 1 hour), where a single or an aggregated (e.g., the average) value is provided for each period. The granularity and the applied window mechanism (sliding, jumping) of the data retrieved is very important for the various algorithmic functions.

Next, we present a set of algorithmic operations that we judge to be important for public cloud analytics and that can be useful in various management decisions:

- Profiling (analytic): use algorithms or other mathematical techniques to discover patterns or correlations in (large) quantities of data. Profiling relates to a number of research areas, such as pattern recognition, machine learning (unsupervised learning in our case, since the pattern is not known) and other. In clouds, profiling can produce useful information regarding the way virtual resources are used (e.g., detect cyclical loads), which can be very important for IT administration and management. In our case, the signal we wish to identify patterns on is usually the utilization (cpu and network) of an instance during a time period (a day, a week, a month, etc.). The produced information will show trends of computing consumption and seasonality in the way virtual resource are used. In addition, identifying relationships among instances or groups of instances is another profiling operation that can trigger several management decisions, such as moving these instances in machines that are close to each other, or their workload in the same instance.
- Spikes Detection (online): identify abnormal changes in the resource portfolio of a user and its associated costs. Spikes detection is an important operation in many different research fields, where a variety of methods have been proposed. In the cloud computing environment, spikes relate to abnormal changes in an organizations resources portfolio, utilization and associated costs. They may also indicate erroneous operation or malicious attempts that require administrator intervention.
- Prediction (planning): estimate future resource usage and cost to be used for resource planning. Again a number of different prediction techniques can be applied. This function also relates to aging analysis, where past data is used to estimate the total or residual running time of a process in order to determine, for example, if On-Demand Instances (ODI) or Reserved Instances (RI) would be more appropriate for serving a new application, or if an existing application should be switched from one mode to the other. Predicting resource usage and cost is clearly very important for resource planning.
- Resource Resizing (planning): detect underutilized resource capacity and highlight cost reduction opportunities by recommending oversized machines that should be replaced with smaller and lower-priced ones and vice versa. Under-utilized or over-utilized resources indicate opportunities for cost reduction, or performance improvement. Resource optimization mechanisms can be used to build a recommendation system regarding which oversized machines should be replaced by smaller and lower-priced ones or by more powerful machines so as to serve the increasing needs of the applications.
- Workload consolidation & migration (planning): identify low performing instances at different time intervals (using profiling methods) will assist in consolidating groups of virtual machines in a single physical machine; maximizing resource utilization and reducing costs by shutting down the remaining unused machines.

5 SuMo Toolkit

In this section we present the SuMo toolkit that we developed to contain the necessary mechanisms for collecting monitoring data from Amazon Web Services (AWS) and analyzing them for optimization purposes. SuMo includes mechanisms that correspond to the algorithmic operations presented in Section 4, such as the CUO mechanism, to be presented in Section 6, for cost and utilization optimization in clouds. SuMo makes easy for anyone, a researcher or an administrator, to analyze the owned instances, run the proposed mechanisms or implement new and more intelligent ones. SuMo is open-source and is available through github [47].

SuMo is written in Python, utilizing the boto framework [33] for communicating with AWS. SuMo is

used as python library providing the required programmatically functionality to access, monitor, analyze and optimize a user's AWS instances. SuMo uses the SciPy [34] and NumPy [35] libraries for scientific computations. For the ILP solving SuMo interfaces with IBM ILOG CPLEX Optimizer [36], which is free for non-commercial purposes.

SuMo consists of three main components/modules, shown in Fig. 2: *cloud Data, cloud Keeping* and *cloud Force*. *cloud Data* is responsible for collecting monitoring data, *cloud Keeping* contains a set of Key Performance Indicators (KPI), while *cloud Force* incorporates a set of analytic and optimization algorithms.

5.1 Cloud Data Module

The *cloudData* module contains all the necessary methods for retrieving a user's current running instances, for getting the type and price of an instance, and its exact characteristics (e.g., EC2 Compute Units - ECU). Interestingly, AWS do not provide directly, through an API, instance pricing and information on the instance characteristics. Therefore, to obtain explicit pricing information one has to retrieve and analyze the JSON files used by the AWS web site that contain the relative information [37]. For the instances' characteristics, SuMo provides static JSON files with the necessary information. The *cloudData* module also contains methods that return, through the boto API (and indirectly through CloudWatch, see Section 3 and [28]) information regarding the CPU, disk and network utilization by a particular instance, for a given time period and particular time slot. The returned utilization information is in the form of a signal (Fig. 1). Some of the most important functions of *cloud Data* module are the following:

- *get_instances*: returns a list of all running instances
- *get_instance_metric*: returns statistics for the metrics listed in Section III for a particular time period
- *get_instances_workload*: computes per instance workload based on their capacity and CPU usage
- *get_instances_cost:* computes per instance cost based on their prices [37] and running times

Since it is not always possible for a researcher to have access to a large number of instances and their related data, SuMo also includes *simCloudData* module. This module provides access to a set of functions for creating "simulated" (or synthetic) instances and the associated data (e.g., utilization profile). Users can configure all the basic parameters (machine type, region, operating system, status, instance usage) for creating the simulated instances. In particular, the user specifies the number of simulated instances to be created along with arrays of values, indicating the percentage of these instances that will be of a specific machine time (e.g., m1.small), be located at a particular region (e.g., US East), etc. Each simulated instance's values for a particular metric (e.g., CPU utilization) over a period of time (e.g., over a whole day with step size of one hour) are calculated using uniform distributions whose limits (minimum and maximum values) can be selected. For example, in case of the CPU utilization, the candidate limits are the following: idle=[0,0], low=[1,10], normal=[11,35], medium=[36,69], high=[70,90], very_high=[91,98], exceedable=[99,100], uniform=[0,100]. Other distributions can easily be added.

The "simulated" data/information produced by *simCloudData* can be used by the methods available to the other modules, without any other requirements or modifications, since they are in the same format with the data produced by *cloudData* module's

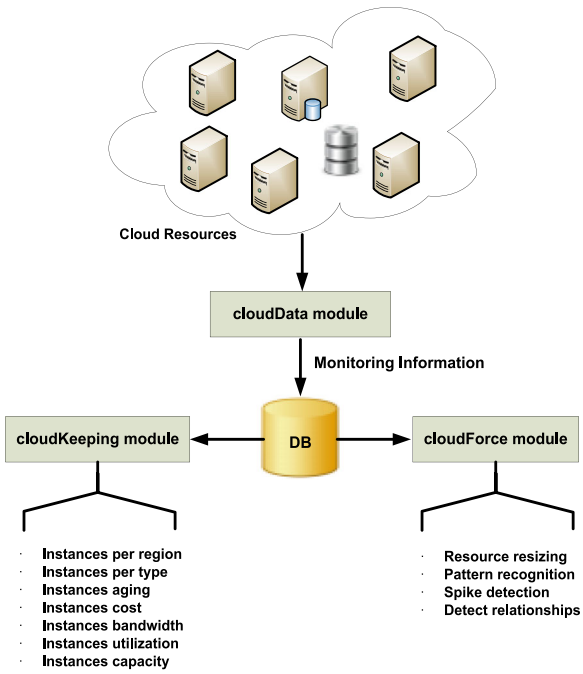

Fig. 2 SuMo – SuMo basic modules

methods. Some of the most important functions of the sim*CloudData* module are the following:

- *get_instances:* creates a number of "simulated" instances
- *get_instance_metric:* creates "simulated" statistics data, based on user preferences

In what follows we present an example of what get_instances function returns (in JSON format):

[{'type': 'm1.xlarge', 'region': 'ap-southeast-1', 'os': 'mswin', 'id': 'i-sim0', 'state': 'running'},
{'type': 'm2.4xlarge', 'region': 'ap-northeast-1', 'os': 'linux', 'id': 'i-sim1', 'state': 'running'},
{'type': 't1.micro', 'region': 'eu-west-1', 'os': 'linux', 'id': 'i-sim2', 'state': 'running'},
{'type': 'm1.xlarge', 'region': 'us-west-2', 'os': 'mswin', 'id': 'i-sim3', 'state': 'running'},
{'type': 'm1.xlarge', 'region': 'us-west-1', 'os': 'linux', 'id': 'i-sim4', 'state': 'running'},
{'type': 't1.micro', 'region': 'eu-west-1', 'os': 'mswin', 'id': 'i-sim5', 'state': 'running'},
{'type': 'm1.small', 'region': 'us-west-1', 'os': 'linux', 'id': 'i-sim6', 'state': 'running'},
{'type': 't1.micro', 'region': 'ap-northeast-1', 'os': 'linux', 'id': 'i-sim7', 'state': 'running'},
{'type': 'c1.medium', 'region': 'eu-west-1', 'os': 'mswin', 'id': 'i-sim8', 'state': 'running'},
{'type': 'c1.medium', 'region': 'ap-southeast-1', 'os': 'linux', 'id': 'i-sim9', 'state': 'running'},
{'type': 't1.micro', 'region': 'ap-southeast-1', 'os': 'linux', 'id': 'i-sim10', 'state': 'running'},
{'type': 'm1.medium', 'region': 'sa-east-1', 'os': 'linux', 'id': 'i-sim11', 'state': 'running'},
{'type': 'm1.xlarge', 'region': 'ap-northeast-1', 'os': 'mswin', 'id': 'i-sim12', 'state': 'running'},
{'type': 'm1.large', 'region': 'us-west-1', 'os': 'linux', 'id': 'i-sim13', 'state': 'running'},
{'type': 'm1.large', 'region': 'ap-northeast-1', 'os': 'linux', 'id': 'i-sim14', 'state': 'running'}]

Also, we present an example of what get_instance_metric function returns (in JSON format) for the CPUUtilization metric:

{'Timestamp': 'datetime.datetime(2014,08,04,22,20)', 'Average': 99.40455346937941, 'Maximum': 99.40455346937941, 'Minimum': 99.40455346937941, 'id': "{'type': 'c1.xlarge', 'region': 'us-west-1', 'os': 'mswin', 'id': 'i-sim9', 'state': 'running'}", 'Unit': 'Percent'}

5.2 Cloud Keeping Module

The*cloudKeeping* module contains a set of Key Performance Indicators (KPI) that perform a number of best practice checks (as the corresponding monitoring tools provided by various companies [30–32]) and statistics calculations. The current version of *cloudKeeping* module includes the following functions:

- *get_instances_per_region:* returns the percentage of running instances per region
- *get_instances_per_type:* returns the percentage of running instances per machine type
- *get_instances_aging:* returns information regarding the aging of the running instances
- *get_cost(type="instance", "os", "region", "instance type"):* returns the total cost per type
- *get_bandwidth(type="instance", "os", "region", "instance type") :* returns the total bandwidth per type
- *get_utilization(type="instance", "os", "region", "instance type") :* returns the total utilization per type
- *get_capacity(type="instance", "os", "region", "instance type") :* returns the total computational capacity per type
- *get_longest_running_instances: returns the instances that are up and running the longest time*
- *get_recently_launched_instances: returns the instances that were launched more recently*

A useful notion in public clouds is that of instance aging. As mentioned earlier, in Amazon Web Services (AWS), a user pays for the compute capacity of an ODI instance by the hour, while in the case of RI instances there is a one-time payment for reserving an instance and then the hourly cost is quite smaller than that of ODI instances. As a result for every used ODI instance there is a "breakeven" usage duration beyond which the use of an equivalent RI would be more advantageous in terms of cost. Vice versa, the use of a RI instance may be a bad choice if its usage duration is way lower than this breakeven point. In Fig. 3, we see the breakeven point for a Small Linux Instance in the US East region. In this case the ODI instance's cost is $0.065 per hour, while the corresponding cost of a RI instance (assuming light utilization and 1 year term) is $0.039 per hour and $69 upfront. We see that the break even point occurs around the ∼107th day, assuming of course that the instance is constantly used. The *get_instances_aging* function returns information for each instance indicating how close it is to the breakeven point, and suggesting the need to transform an ODI instance into a RI instance or vice versa.

In what follows we present an example of what get_instances_per_region function returns (in JSON format):

{'us-east-1': 33, 'ap-northeast-1': 13, 'sa-east-1': 13, 'ap-southeast-1': 20, 'us-west-2': 6, 'us-west-1': 6, 'eu-west-1': 6}

Similary, get_instances_per_type function returns (in JSON format) the following:

{'m1.medium': 20, 'm1.large': 6, 'm2.2xlarge': 6, 'm1.small': 0, 'c1.medium': 6, 'm1.xlarge': 13, 'm2.xlarge': 13, 't1.micro': 13, 'm2.4xlarge': 20, 'c1.xlarge': 0}

5.3 Cloud Force

The *cloud Force* module includes SuMo's algorithms for executing the algorithmic operations mentioned in Section 4. More advanced mechanisms/algorithms (e.g., for pattern or spark detection) will also be included in the future.

For profiling SuMo implements a pattern recognition mechanism that uses cross-correlation as a measure of the similarity of two time signals. The first signal is the pattern we wish to match. Since we do not initially have any idea about the periodicity (if any) of

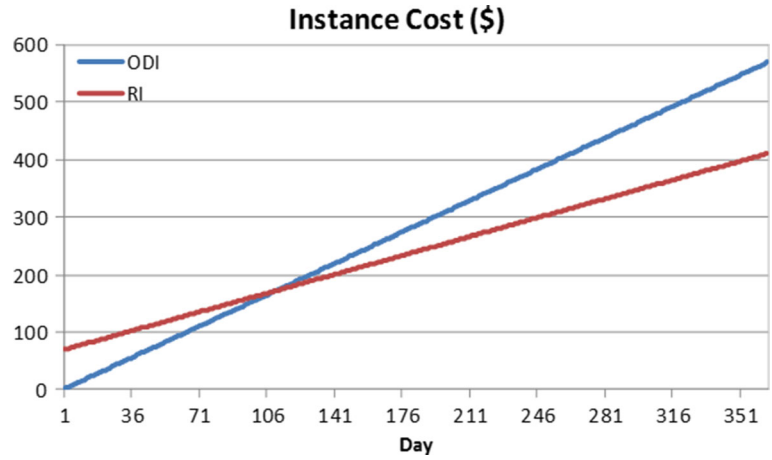

Fig. 3 ODI and RI breakeven point

the pattern, we apply the pattern recognition methodology recursively by selecting a pattern e.g., the signal in 1st hour, day, week, etc. and then matching it against the original signal. The second waveform is the original signal shifted in various periods. In the following snippet of code, the cloudForce function for pattern detection, namely *patternDetection.based_on_xcorr*, receives as input a signal (corresponding, for example, to the computational utilization of an instance), and returns an array of patterns that have been identified in that signal.

```
import sumo.cloudForce.patternDetection
[s,p] = sumo.cloudForce.patternDetection.based_on_xcorr(signal)
```

Additionally, for profiling SuMo uses the work presented in [22] and considered for the case of clouds in [23], so as to detect relationships between instances. In [23], authors analyze tweets to detect real-life events. They build a signal for individual words by measuring the number of words appearing in tweets at fixed intervals and then applying wavelet analysis on the frequency-based raw signals of the words. The authors then build a correlation matrix, which is partitioned, using a modularity-based graph partitioning technique, into groups of words each indicating a particular event. A similar event detection method is used in SuMo for analyzing communication traces and capacity utilization profiles, by creating a respective signal per VM (e.g., counting the amount of data sent (and received) by a VM in each period / time slot). The implemented function returns groups of instances that relate to each other.

For spike detection SuMo uses the return function $R(t)$, defined as:

$$R(t) = \frac{S(t) - S(t-1)}{S(t-1)},$$

where $S(t-1)$ and $S(t)$ are two consecutive observations for a signal at time instances (or time slots) t-1 and t respectively. The advantage of looking at returns of a signal is that one can see the relative changes in the signal variable. After calculating all returns, SuMo selects the ones that exceed the average by a number of standard deviations (σ):

$$R(t) > \text{avg}(R) + param \cdot \sigma(R).$$

In the following snippet of code, the cloudForce function for spike detection, namely *sparkDetection.based_on_returns*, receives as input a signal corresponding, for example, to the computational utilization of an instance and returns an array of indices in the array where spikes have been identified.

```
import sumo.cloudForce.sparkDetection
[spks] = sumo.cloudForce.sparkDetection.based_on_return(signal)
```

For resource resizing, SuMo incorporates Cost and Utilization Optimization (CUO) mechanism that solves a cost and utilization optimization problem, formulated as an ILP (Integer Linear Program), which is presented in detail in Section 6. In the

following we present a snippet of code for retrieving instance information and computing a new set of instances, which can serve the initial workload with lower cost and higher utilization (in the case of computational capacity decrease) or with better performance (in the case of resource capacity increase).

```
import sumo.cloudData
import sumo.cloudForce
instances = cloudData.get_instances()
statistics = cloudData.get_statistics(instances)
[init, SuMo] = cloudForce.cost_util_opt(instances, statistics)
```

6 CUO Mechanism

In this section, we present Cost and Utilization Optimization (CUO) mechanism that receives as input the static (e.g., type, cost) and dynamic (cpu utilization, running time) characterists of as set of running AWS instances, and computes, using a multi-objective function, a new set of instances that maximizes resource utilization for the cloud provider and minimizes the associated cost for the user. This new set of instances can serve as a suggestion to the instances' owner for modifying (at least some of) the types of the instances he uses, and realizing cost benefits assuming that the past behaviour is indicative of the owner's future computing needs.

CUO's described optimization is formulated as an Integer Linear Programming (ILP) problem and solved optimally using the IBM ILOG CPLEX Optimizer [36] ILP solver. Heuristics can also be used for solving the same problem.

6.1 Modelling

We denote by

$$S = \{I_1, I_2, \ldots, I_M\}$$

the set of running instances. Each running instance I_i is characterized by two parameters: the type of instance (IT_i) and its workload (WL_i), that is, the CPU utilization (measured in EC2 Compute Units - ECU) over time:

$$I_i = \{IT_k, WL_i\}, i \in [1, 2, \ldots, M], k \in [1, 2, \ldots, D],$$

where $D$is the number of different instance types (Section 3). In AWS, a running instance's type is defined by the machine type, the operating system (O) and the region (R) where it runs (Table 1). In the remainder of our work, we assume that all instances in S are either On-Demand Instances (ODI) or Reserved Instances (RI). Also, we can infer from the set S the number of instances N_i of each type i that are in use and the total number of running instances

$$M = \sum_{k=1}^{D} N_k.$$

It is assumed that an infinite number of instances of each type can be used. Also, the CPU utlization of an instance over a time period T, provided as input to the ILP (through CloudWatch Section 3, [10]), is a signal

$$WL_i = [wl_i(1), \mathrm{wl}_i(2), \ldots, wl_i(T)], \quad (1)$$

where $wl_i(t)$ is the CPU utilization (measured in EC2 Compute Units - ECU) of the i-th instance at time slot t (as already mentioned, the step/slot size in AWS is 1 or 5-minutes). The proposed mechanism assumes that the workload $\overline{WL_i}$ of the i-th instance over time period T (referred to as *aggregated workload* of instance I_i) is equal to the average ($avg(WL_i)$) of the CPU utilization signal WL_i (it could also be equal to its maximum value, $\max(WL_i)$). Moreover, the proposed mechanism attempts to select instances that are in position to serve the input workload even with high fluctuations, that is, variations in the CPU capacity they use overtime. To this end, the Desired/targeted Utilization DU (for the new set of instances S_{new}) in the average case scenario is computed based on the workload served by the original set of instances and on an utilization factor f provided each time by the user:

$$DU = f \cdot \sum_{i=1}^{M} \overline{WL_i}, f \geq 1 \quad (2)$$

The parameter f plays an important role. For example, setting $f$equal to 1 we state that the feature workload requirements will be similar to the present ones. This could lead to the selection of instances that serve the workload with just enough resources, reaching in this way utilization around 100 %, since CUO maximizes resources utilization. However, full (100 % percent) utilization is not necessarily something desirable, as it exhibits the workload's need for a larger

capacity instance, while possible incremental workload flactuations cannot be handled efficiently. So, setting parameter f to a value larger than 1, we (in a way) overestimate the future workload requirements, resulting in an utilization less than 100 % (since CUO will select larger capacity instances than in the former case) and enabling the better serving of instances that exhibit fluctuations in their capacity needs. Of course, using a very large value for f would result in an inefficient (in terms of the utilization and associated costs) selection of resources.

6.2 Formulation

Table 2 presents the proposed ILP formulation.

The first constraint is used to ensure that an instance can be replaced by only one new instance, while the selection of the same type of instance means actually no change. The second constraint guarantees that the new instance is able to serve the original workload. The third constraint guarantees that the new set of instances has the appropriate capacity to accommodate the instances' desired workload, based on Eq. (2). The fourth set of constraints relates to a number of choices an administrator should be able to make. Changing the operating system of an instance is usually not a desired option, since this could incur additional overhead (e.g., configuration, code rewriting) to the administrator. The administrator may also wish (or not) to keep unchanged the region an instance is operating in, or its type, for various reasons, including policy or security. In addition, it may be important for the administrator that some basic or particular memory constraints are satisfied by the new instances.

The presented ILP finds the appropriate instance to serve each demand. The objective is to minimize a weighted sum of two parameters: i) the cost for using the instances for a given period of time (the cost parameter), and ii) the difference between the offered processing capacity and the desired utilization (the utilization parameter), Eq. (5). The weighting coefficient W controls the relative significance given to these two parameters in the optimization function. Values of W close to 0 make the cost for using the instances the dominant optimization parameter, in which case instances with low cost (and probably lower CPU capacity) are prefered, neglecting the resource utilization ctiterion. In contrast, values of W close to 1 make

Table 2 ILP Formulation

ILP formulation

Input

D: Number of different types of instances

S: The set of running instances $S = \{I_1, I_2, ..., I_M\}$, $I_i = \{IT_k, WL_i\}$, $i \in [1, ..., M], k \in [1, ..., D]$

I_i: A running instance

$N_{k,}, k \in [1, 2, ..., D]$: Number of instances of type k that are currently in use

M: Total number of instances of all types that are in use

$\overline{WL_i}, \quad i \in [1, 2, ..., M]$: The aggregated workload of the i-th running instance of the whole time period of observat ion

$P_{k,}k \in [1, 2, ..., D]$: CPU capacity of instance of type k

$C_{k,}k \in [1, 2, ..., D]$: Cost per hour of instance of type k

$CP_{k,}k \in [1, 2, ..., D]$: Capital cost of instance of type k (for ODI, this cost is equal to zero)

DU : Desired/targeted utilization for the new set of instances

T : period of observation

W: weighting coefficient for the cost function

Variables:

$X_{i,k}i \in [1, 2, ..., M], k \in [1, 2, ..., D]$: Boolean variable equal to 1 if i-th running instance is of type i, equal 0 otherwise

P: The total computational capacity $P = \sum_{i=1}^{M} \sum_{k=1}^{D} X_{i,k} \cdot P_k$ (3)

of the set(old or new) of instances for each $i \in [1, ..., M], k \in [1, ..., D]$

C: Cost of the instances $C = \sum_{i=1}^{M} \sum_{k=1}^{D} X_{i,k} \cdot C_k \cdot T$ (4)

for the given time period for each $i \in [1, ..., M], k \in [1, ..., D]$

ILP formulation

Minimize $W \cdot C + (1 - W) \cdot (P - DU)$ (5)

Constraints

1. New instance assignment
$$\sum_{k=1}^{D} X_{i,k} = 1 \quad \text{for each } i \in [1, ..., M]$$
2. Instance capacity constraint
$$X_{i,k} \cdot \overline{WL_i} \leq P_k \quad \text{for all } i \in [1, ..., M], k \in [1, ..., D]$$
3. Desired Utilization constraint
$$P \geq DU$$
4. Additional Constraints:
   - Operating System (O)
   - Region of Operation (R)
   - Instance Type (IT)
   - Memory Size

resource utilization the dominant optimization parameter, in which case the selected instances processing capacity is close to the desired utilization without taking their cost into consideration. However, in practice, as the results in Section 7 will exhibit, the total cost is also reduced.

The number of variables and constraints in the above ILP formulation depend on the number D of instance types that can be used and the number M of running instances in S. In the worst case, all the D available instance types are candidates to replace a running instance's type. Additionally, the number of candidate types can be reduced if the user sets region and operating constraints. For example, the user can set constrains to ensure that the new running instance is of the same machine type, of the same operation system, or running in the same region (Table 1) with the current one. Using these constraints the number of available choices for the algorithm is reduced significantly, along with the time required to acquire the optimal solution. For that reason the proposed mechanisms include a pre-processing phase, in order to select the type of instances that can be used by the ILP. In case there are no such constraints, the pre-processing phase can be omitted.

6.3 Extensions

The presented algorithm can also be used to support and optimize other resource characteristics, such as the the memory (instead of the CPU) of the virtual instances. Similarly, the presented mechanism can maximize the utilization of the memory usage for all instances by suggesting the replacement of underutilized (in terms of memory) instances with other instances of lower memory capacity and of lower cost. Again, by configuring properly parameter f, the CUO algorithm can also lead to the selection of high memory instances in order to replace instances with utilization near 100 %, indicating memory bottleneck. In our implementation of the CUO algorithm in SuMo, we did not include this extension because memory utilization is not among the metrics provided by default by CloudWatch (Section 3).

In the CUO algorithm presented, we do not consider instance migration costs, which are in any case difficult to calculate. Migration costs depend on a number of measurable and non-measurable (at least, not easily) parameters. For example the instance data size need to be moved can be relatively easily calculated, while the cost of disrupting the service offered by the particular instance, is quite more difficult to calculate. In any case, if instance migration cost is known, the CUO algorithm can easily be extended appropriately to include it in its optimization.

A limitation of the ILP formulation presented is that the number of instances in the new set of instances is the same as in the original one. Nevertheless, CUO can be coupled with AWS autoscaling functionality. AWS Auto scaling ensures that the number of instances one is using increases during demand spikes, and decreases automatically when demand is reduced. CUO algorithm's suggestions can be applied even when this auto scaling functionality is enabled, providing suggestions for the resizing of the resources independently of whether they have been enabled for the whole period of observation or not.

Moreover, many real world services (like Pinterest content sharing service [38]) utilizing AWS tend to use a mix of reserved - RI, on demand - ODI and spots EC2 instances to serve the fluctuating daily traffic (Fig. 4), utilizing also the AWS auto-scaling feature. Using RI instances all the time they provide a baseline capacity, paying an initial fixed cost and a very small hourly cost. Normal ODI instances are used for serving the expected capacity for the day workload, with no upfront cost, only an hourly one. Additionally, peaks can be handled utilizing spot EC2 instances, which generally cost less than the ODI instances. AWS Auto scaling ensures that the number of instances used increases during demand spikes, and decreases automatically when demand is reduced. (Other services, utilizing AWS will probably use similar tactics, however we bring up Pinterest use case, since some internal information of the service's operation have become public [34]).

The CUO algorithm can also handle both ODI and RI instances, providing instance resource resizing suggestions for ODI instances and workload migration suggestions from ODI to RI instances, so as to increase their utilization. Moreover, the proposed mechanism focuses on EC2 recourses only and does include propositions for migrating to other kind of resources (e.g., to Amazon DynamoDB if an instance is I/O bounded). Another, possible extension is that different aggregation operators (maximum, minimum, average) can be used to calculate the aggregated workload $\overline{WL_i}$ of the i–th instance over time period T. In

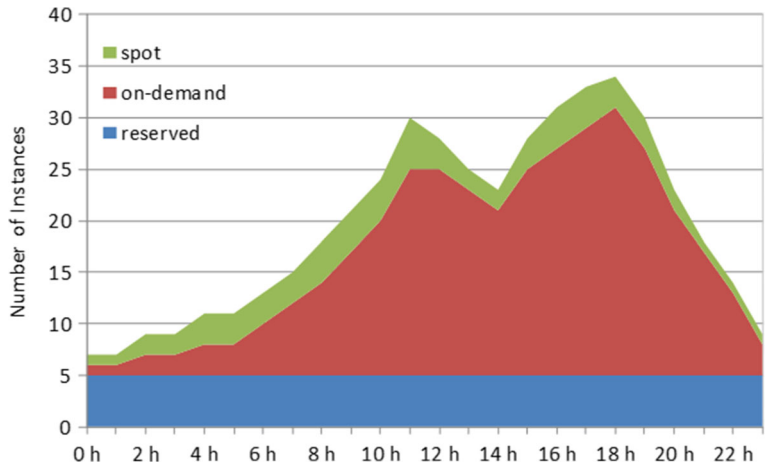

Fig. 4 Pinterest use case, where a mixed of EC2 instances are used to serve the fluctuating daily traffic

this way, for example, CUO can handle both instances with cyclical loads (e.g., web servers), where the peak workload at "rush hours" is important as well as instances with more uniform load.

7 Results

7.1 Configuration Scenarios and Metrics

We evaluated the Cost and Utilization Optimization (CUO) algorithm using synthetic data produced by *simCloudData* and feeding them in the CUO mechanism included in *cloudForce*. The *simCloudData* module produces an initial set $S$of M instances, and their workload signal WL (Eq. (1)) for a period of $T = 24$ hours (step size equal to 1 hour), while *cloudForce* calculates a new set S_{new} of instances for serving the same load. We do not evaluate the other mechanisms (profiling, spark detection) incorporated in SuMo, since their operation is straightforward.

Also, even though using real data would be more useful in evaluating the proposed mechanism, monitoring data from public clouds are not publically available. On the other hand, one can find a small set of raw traces originating from data centers [42, 43] (some of which we also used in a previous work of ours [23]) and several works profiling these traces [44]. These cannot be used in our work since they do not match the model of operation of AWS or of any other public cloud provider. Instead in order to perform our evaluation, we identify a number of configuration scenarios that correspond (most of them) to real use cases, such as the Pinterest's [38]:

- *Flat*, where every instance's OS and region of operation can be changed by CUO. This is not a very realistic scenario, and is merely used in order to obtain some baseline results to compare against. Additionally, the workload of each instance is selected uniformly in the range of [0,100] percent of the instance's machine type capacity. Other workload ranges ([0,20] and [80,100] percent) are also evaluated.
- *Region-constrained*, where every instance's OS and region of operation cannot change. The workload scenarios of the Flat configuration are used.
- *Cyclical-load,* where the total workload changes over time causing the initiation of more instances, following a pattern similar to the one presented in Table 2 for the Pinterest use case. Also, we assume that the OS and region of operation of all the instances cannot change.

Additionally, in our experiments, we consider ODI whose exact type is selected uniformly among the 196 different ODI instance types (see Table 1). The weighting coefficient W in the optimization function of Eq. (5) was equal to 0.5, while parameter f was set to 1. Other parameter values were also considered in the results that follow. A number of experiments with different values for the parameters (WL, T, f, W) were also performed with some of the results obtained presented in the following.

The main metrics of interest are the capacity, the cost and the utilization of the instances selected by SuMo, which are compared to those of the initial set of instances. Furthermore, we are interested in tracking the changes of machine types (Table 1) in the new set of instances S_{new} selected by SuMo.

7.2 Flat Configuration: Performance

We initially performed a number of experiments to study the role played by the number M of running instances. This was the basic set of experiments performed to evaluate the improvements obtained by the CUO mechanism of SuMo.

Figure 5 shows the total cost (Eq. (4)) of the instances in the original set S and in the new set S_{new}selected by SuMo, assuming an observation period equal to T. We see that the instances selected by SuMo result in a 40-50 % reduction in the cost paid by the user. In both instance sets, S and S_{new}, the total cost increases close to linearly with the number of running instances M, as expected.

Figure 6 shows the utilization of the cloud resources used and the total capacity (measured in EC2 Compute Units - ECU) of the instances in S and in S_{new}. The instances' utilization is defined as the ratio of the instances' average workload $\overline{WL}$ during the observation period T, over the total capacity P of instances, defined in Eq. (3). We observe that the instances selected by SuMo result in a higher utilization ratio (Fig. 6a), around 10 % higher, due to the smaller total capacity instances chosen (Fig. 6b). The exact utilization ratio is affected by the chosen parameters (see also Section 7.4). However, the improvement in utilization shown in Fig. 6a ($\sim$ 10 %) of S_{new} in comparison to S is higher than the percentage of capacity reduction, shown in Fig. 6b, leading to the conclusion that the CUO mechanism also achieves a better matching between the requested workload and the offered capacity than the one for the initial set S. The efficiency of the matching provided by the CUO mechanism is also exhibited by the fact that the utilization achieved by the new set of instances S_{new}, remains unaffected by the number of running instances M. One would expect that a large number of running instances would leave room for more "errors" in the assignment of instances to resources and inefficient utilization of the available capacity, but this does not appear to be the case. Also, we should note that the utilization of the instances in the initial set S remains unaffected by M, since the values of the related parameters (e.g., workload) were chosen, as already mentioned, from a uniform distribution.

Additionally, one may expect that CUO (or other similar mechanisms) should achieve utilization as close to a 100 % as possible. However, the resulting utilization is affected by two parameters: First, it is affected by the number of upgrades (see also next section) performed by CUO (so as to alleviate performance bottlenecks) that result in selecting instances of higher capacity, leading to smaller utilization for particular workload/instance pairs (e.g., from 100 % to 80 %). Second, there is always an upper bound on the utilization that can be achieved, considering that in the new set S_{new}, all the workloads have to be served by the selected instances (Eq. (6)) and that the machines' capacity offered by Amazon [27] is not linear. For example, there are machine types of (this list is not

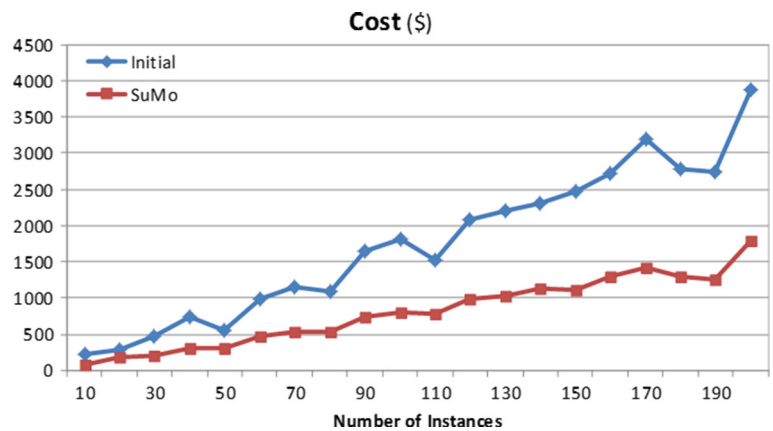

Fig. 5 The total cost *(measured in $)* of the instances of the initial set S, and the new set S_{new}, selected by SuMo

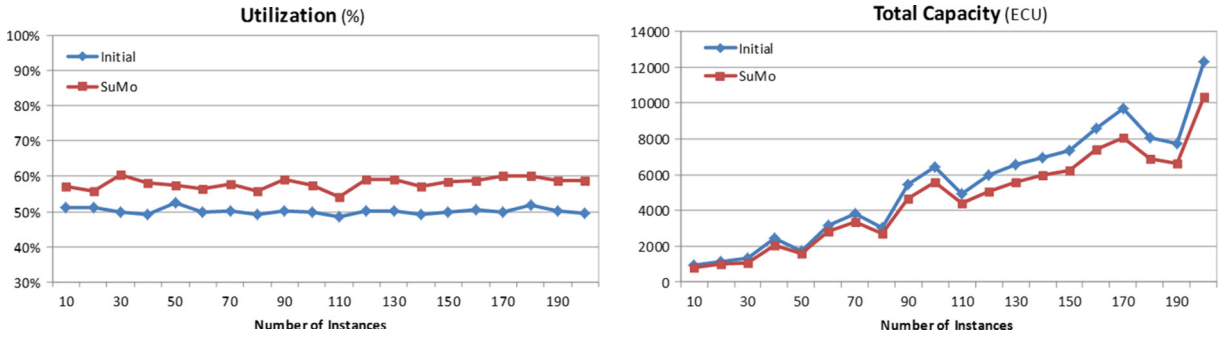

Fig. 6 The percentage utilization and the total capacity *(measured in Elastic Compute Units – ECU)* of the instances of the initial set S, and of the new set S_{new}, selected by SuMo

complete) 1, 2, 4, 5, 7, 8,13, 26 ECU capacity and so for an instance of 4 ECU capacity and workload equal to 2.5 ECU, SuMo cannot find an instance that results in a higher utilization, only a cheaper instance by choosing another OS or region of operation. This also leads to the remark that had all possible ECU capacity granularities (e.g., 1,2,3...,25,26) been provided by Amazon, the utilization achieved by CUO would be higher, with the "cost" however landing on the public cloud provider's side, regarding the efficient management of these different instance types.

7.3 Flat Configuration: Instance'S Changes

In Fig. 7, we illustrate the changes in the machine types, operating systems and regions (Table 1) in the new set of instances S_{new} selected by SuMo, in relation to the original set S. We observe that the CUO mechanism usually results in regional changes, while the changes in the machine type and the operating systems are less. Machine type changes relate both to utilization and cost factors, while regional and operating system changes relate only to cost parameters.

In particular, while initially the percentage of Linux versus Windows instances is on average (for various number of instances) equal, the application of CUO leads to several instances with Windows OS to be replaced by the cheaper Linux running instances (Fig. 8). Also, while the instances of the initial set S run uniformly in all available regions, in the set Snew, selected by SuMo, a smaller number of regions are preferred (Fig. 9). This relates to the pricing policy used in each region that makes SuMo choose the "cheaper" regions for running the instances. We should note that in practice, an administrator would probably request that the instances'

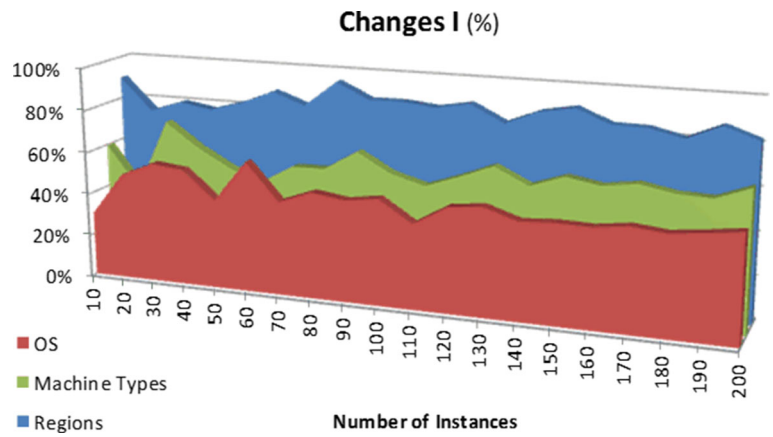

Fig. 7 Changes in the machine types, operating systems and regions (Table 1) in the new set of instances S_{new} selected by SuMo, in relation to the original set S

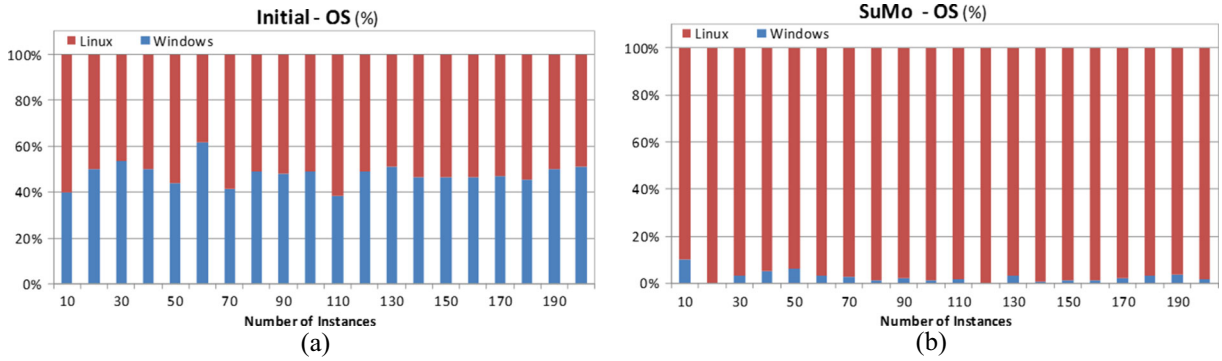

Fig. 8 Operating Systems of the instances in **(a)** the initial set S, and **(b)** the new set S_{new} selected by SuMo

OS remain unchanged, since not all Linux applications run in Windows, and vice versa, or simply because the administrator is not interested in changing the running environment of the applications. Moreover, regional constraints could also be set by an administrator.

The percentage of regional changes (that result in cost reduction) would have been smaller if the migration costs were also accounted for. However, having said that, we should highlight the importance for the cloud community to extend research and development efforts towards Wide Area Network (WAN) Virtual Machine (or instance, in Amazon terms) migration. Considering a future of many small and large cloud providers, operating in different regions of the world, efficient (in terms of time, resources utilized etc.) instance migration of WAN is a very important area.

Figure 10 shows the percentage of upgrades, downgrades and the cases without changes of instances in the new set S_{new}, selected by SuMo. We have an upgrade (or downgrade or no change) when the workload of an instance in set S is executed in a higher (or lower or equal, respectively) capacity instance in the new set S_{new}. We observe that the CUO mechanism results mostly (40 % -50 %) in downgrades, while we have fewer upgrades (30 %-40 %) and even fewer cases without changes (10 %-30 %). These percentages are highly affected by the chosen parameters, nevertheless they demonstrate, generally the efficient operation of the CUO mechanism used in SuMo, since not only downgrades are performed, as one would normally expect so to increase resource utilization and decrease total cost, but more intelligent and less obvious choices, often resulting in upgrades, are also considered.

7.4 Flat Configuration: Role of the Weighting Coefficient W

The weighting coefficient W determines the relative importance given to the cost of the user and to the

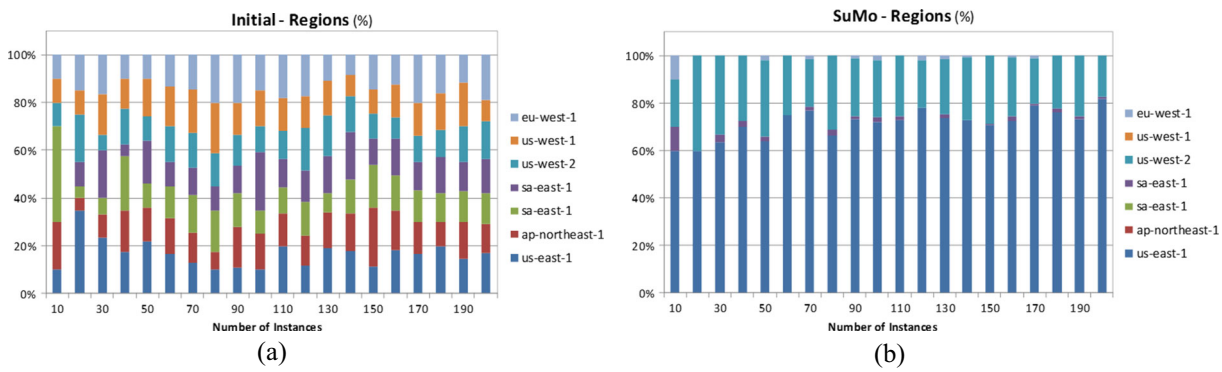

Fig. 9 Regions where the instances are running for **(a)** the initial set S, and **(b)** the new set S_{new} selected by SuMo

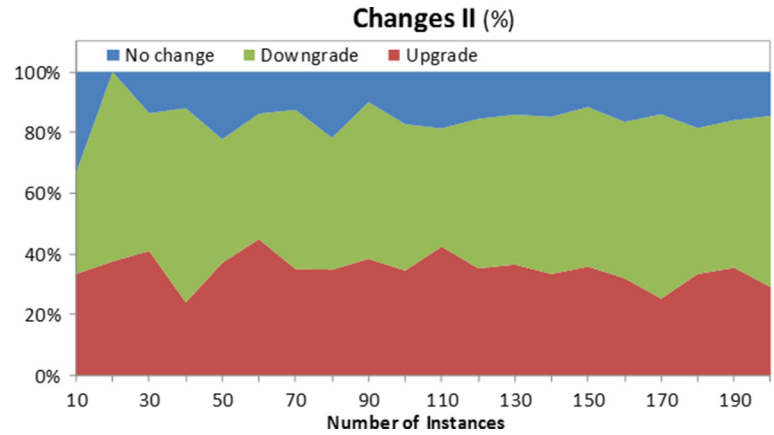

Fig. 10 Percentage of instance type upgrades, downgrades and cases without changes, in the new set of instances S_{new} selected by SuMo, in relation to the original set S

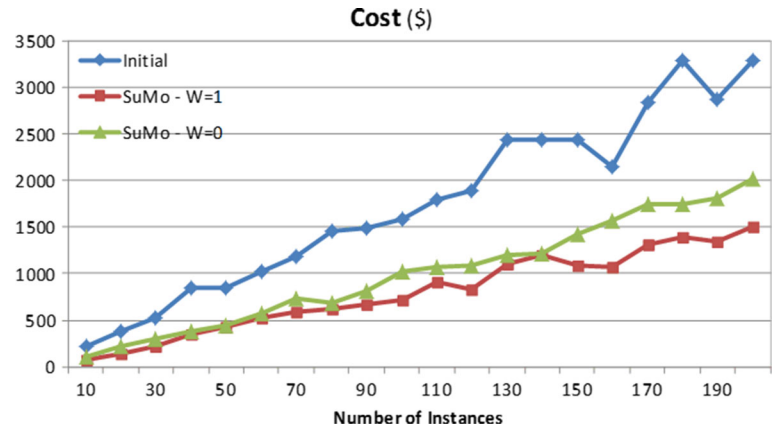

Fig. 11 The total cost *(measured in $)* of the instances of the initial set S, and the new set S_{new}, selected by SuMo, for W=0 and W=1

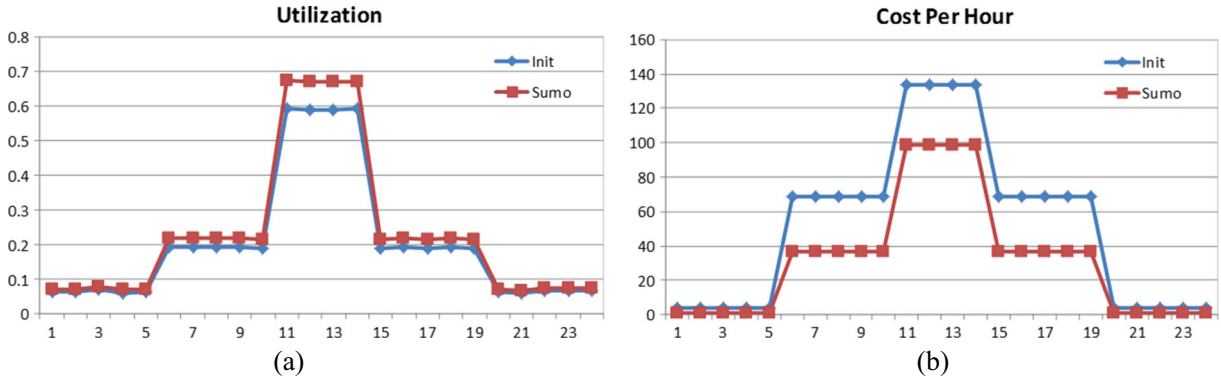

Fig. 12 The % utilization and the cost per hour of the instances of the initial set S, and of the new set S_{new}, selected by CUO, for the Cyclical-load configuration

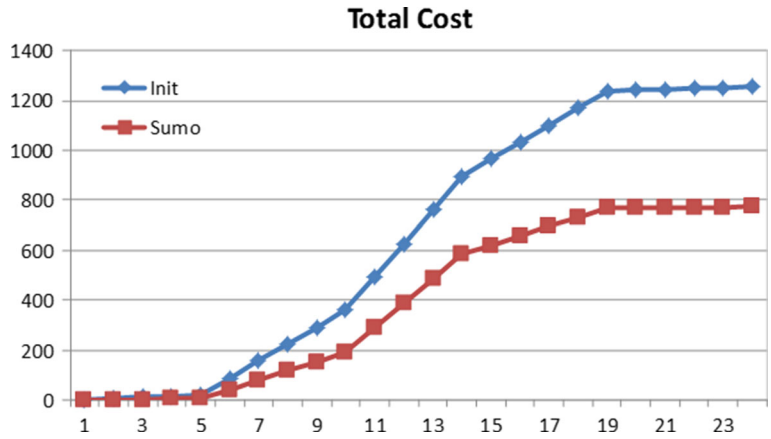

Fig. 13 The total cost (cumulative sum) of the instances of the initial set S, and of the new set S_{new}, selected by CUO, for the Cyclical-load configuration

cloud resource utilization criteria in Eq. (5). We performed experiments for $W = 0$ (maximizing only resource utilization) and $W = 1$ (minimizing only cost to the user). Figure 11 shows, as expected, that when only cost is optimized ($W = 1$), the total cost of the new set of instances S_{new}, is smaller than when only utilization is maximized ($W = 0$). However, even in the latter case, significant cost reduction is achieved, since maximizing utilization leads in many cases to the selection of lower capacity and therefore cheaper instances (Fig. 10). Also, the utilization achieved in both cases ($W = 0$ and $W = 1$), even though larger than the initial one (for set S), is independent of the number of instances M. As mentioned earlier, this is also because there is always an upper bound on the utilization that can be achieved, considering that in the new set S_{new}, all the workloads have to be served by the selected instances (Eq. (6)) and the machines' capacity offered by Amazon [27] is not linear.

7.5 Cyclical-load configuration

In the experiments performed for the Cyclical-load configuration, we assumed a setting similar to the one presented in Section 6.3 and in Fig. 4, for the Pinterest's use case.

In particular, during a 24-hour period there is a variable workload that causes the initiation of new instances to serve the increasing load. In Fig. 12, we observe that the utilization and the cost per hour increase during peak hours. Even in this configuration, with time-varying characteristics, CUO achieves improved utilization and smaller cost. Figure 13 shows the cumulative sum of the cost.

7.6 Region-constrained Configuration

The results obtained for the region-constrained configuration were similar to the Flat configuration, except for the fact that only machine type changes occurred.

8 Conclusions

The widespread use of public cloud resources makes analyzing and optimizing clouds increasingly important, but also makes it difficult for a user or administrator to effectively control their proper use. We have developed a toolkit named SuMo that implements important functionalities for collecting monitoring data from Amazon Web Services (AWS), analyzing them and suggesting changes that optimize the use of resources and the associated costs. SuMo consists of three main components/modules: *cloudData* is responsible for collecting monitoring data, *cloudKeeping* contains a set of Key Performance Indicators (KPI), while *cloudForce* incorporates a set of analytic and optimization algorithms. Optimization in SuMo is performed using an ILP-based Cost and Utilization Optimization (CUO) mechanism that maximizes the utilization of the resources/instances and minimizes the costs of their use. When necessary, CUO also recommends increasing resource capacities, so as to resolve possible performance bottlenecks. In addition, SuMo incorporates an initial set of basic algorithms (for profiling and spike detection) for analyzing the collected monitoring data and detecting trends in the way virtual resource are used or possible malicious or erroneous instance behavior. SuMo is

open-source [47] and can be used as a basis for the development of new mechanisms for the analysis of collected monitoring information from AWS.

We performed a number of experiments using synthetic monitoring data and illustrated the benefits of the proposed CUO mechanism. In summary, the instances selected by the CUO mechanism, used in SuMo, result in a large reduction in the cloud resources cost. Also, it achieves a good matching between the requested workload and the offered capacity, leading also to high utilization ratio. We also observed that CUO mechanism usually results in regional changes for the new instances, while the changes in the machine type and the operating system are less. Machine type changes relate both to utilization and cost factors, while regional and operating system changes relate only to cost parameters, tending to select a limited number of low cost regions for hosting the instances or of cheaper Linux instances. Nevertheless, in practice both regional and operating system changes may not be always possible or without cost. Also, the efficient operation of the CUO mechanism was exhibited by the fact that not only downgrades (selecting lower capacity instances) are performed, as one would normally expect so to increase resource utilization and decrease total cost, but more intelligent and less obvious choices, often resulting in upgrades (selecting higher capacity instances), are also considered. CUO parameterized operation and efficiency under instances with time-varying characteristics is also exhibited.

Acknowledgments This work is implemented within the framework of the Action Supporting Postdoctoral Researchers of the Operational Program Education and Lifelong Learning (Action's Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State.

References

1. Buyya, R., et al.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Futur. Gener. Comput. Syst. **25**, 599–616 (2009)
2. Rimal, B.P., Jukan, A., Katsaros, D., Goeleven, I.: Architecture requirementes for cloud computing systems: An enterprise cloud approach. J. Grid Comput. **9**(1), 3–26 (2011)
3. Amazon Web Services – AWS: aws.amazon.com, last seen January 2014
4. RackSpace: www.rackspace.com, last seen January 2014
5. Openstack: www.openstack.org, last seen January 2014
6. OpenNebula: opennebula.org, last seen January 2014
7. Eucaluptus: www.eucalyptus.com, last seen January 2014
8. Amazon data center size: huanliu.wordpress.com/2012/03/13/amazon-data-center-size, last seen January 2014
9. Designs, Lessons and Advice from Building Large Distributed Systems. www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf
10. Amazon Case Studies: aws.amazon.com/solutions/case-studies/
11. Wang, H. et al.: Distributed systems meet economics: pricing in the cloud. USENIX Hot Topics in Cloud Computing (HotCloud) (2010)
12. Chen, J., et al.: Tradeoffs Between Profit and Customer Satisfaction for Service Provisioning in the Cloud. International Symposium on High performance Distributed Computing (HPDC) (2011)
13. Zanikolas, S., Sakellariou, R.: A Taxonomy of grid monitoring systems. FGCS **21**(1), 163–188 (2005)
14. Kung, H.T., Lin, C.-K., Vlah, D.: CloudSense: Continuous fine-grain cloud monitoring with compressive sensing. USENIX HotCloud (2011)
15. Petcu, D., et al.: Experiences in building a mOSAIC of clouds. J. Cloud Comput. **2**(1) (2013)
16. Ferrer, A.J., et al.: OPTIMIS: A holistic approach to cloud service provisioning. Futur. Gener. Comput. Syst. **28**(1), 66–77 (2012)
17. Mallick, S.: Virtualization based cloud capacity prediction. HPCS, pp. 849–852 (2011)
18. De Chaves, S., et al.: Toward an architecture for monitoring private clouds. IEEE Comm. Mag. **49**(12), 130–137 (2011)
19. Ward, J.S., Barker, A.: Semantic Based Data Collection for Large Scale Cloud Systems. DIDC, pp. 13–22 (2012)
20. Shao, J., Wei, H., Wang, Q., Mei, H.: A Runtime Model Based Monitoring Approach for Cloud. IEEE CLOUD, pp. 313–320 (2010)
21. Meng, S., et al.: Reliable State Monitoring in Cloud Datacenters. IEEE CLOUD, pp. 951–958 (2012)
22. Weng, J., et al.: Event Detection in Twitter. HP Laboratories (2011)
23. Kokkinos, P., Kretsis, A., Varvarigou, T., Varvarigos, E.: Social-like Analysis on Virtual Machine Communication Traces. IEEE Cloudnet (2012)
24. Malkowski, S., Hedwig, M., Jayasinghe, D., Pu, C., Neumann, D.: CloudXplor: A tool for configuration planning in clouds based on empirical data. ACM Symposium on Applied Computing (SAC) (2010)
25. M. Frîncu: Scheduling highly available applications on cloud environments. Futur. Gener. Comput. Syst. **32**, 138–153 (2014)
26. Kokkinos, P., et al.: Cost and Utilization Optimization of Amazon EC2 instances. IEEE Sixth International Conference on Cloud Computing, pp. 518–5255 (2013)
27. Amazon Elastic Compute Cloud: aws.amazon.com/ec2, last seen January 2014
28. Amazon CloudWatch: aws.amazon.com/cloudwatch, last seen January 2014
29. Nagios: www.nagios.org, last seen January 2014
30. Newvem: www.newvem.com, last seen January 2014
31. Cloudability: cloudability.com, last seen January 2014

32. Cloudvertical. www.cloudvertical.com, last seen January 2014
33. boto - A Python interface to Amazon Web Services: docs.pythonboto.org, last seen January 2014
34. SciPy: www.scipy.org, last seen January 2014
35. NumPy: numpy.scipy.org, last seen January 2014
36. IBM ILOG CPLEX Optimizer: www-01.ibm.com/software/integration/optimization/cplex-optimizer, last seen January 2014
37. On demand instances pricing: aws.amazon.com/ec2/pricing/pricing-on-demand-instances.json, last seen January 2014
38. Pinterest use case: www.theregister.co.uk/2012/04/30/inside_pinterest_virtual_data_center, last seen January 2014
39. mOSAIC project: http://www.mosaic-cloud.eu, last seen January 2014
40. Optimis project: http://www.optimis-project.eu, last seen January 2014
41. Aeolus project: www.aeolusproject.org, last seen January 2014
42. Data Set for IMC 2010 Data Center Measurement: pages.cs.wisc.edu/~tbenson/IMC10_Data.html, last seen January 2014
43. Google. Google Cluster Data V1. Available: http://code.google.com/p/googleclusterdata/wiki/TraceVersion1, last seen January 2014
44. Kavulya, S., et al.: An analysis of traces from a production mapreduce cluster. Cluster, Cloud and Grid Comput. (CCGrid), pp. 94–103 (2010)
45. Kertesz, A., Kecskemeti, G., Oriol, M., Kotcauer, P., Acs, S., Rodrıguez, M., Merce, O., Marosi, A.C., Marco, J., Franch, X.: Enhancing federated cloud management with an integrated service monitoring approach. J. Grid Comput. **11**(4), 699–720 (2013)
46. Mendez, V., Casajus, A., Fernandez, V., Graciani, R., Merino, G.: Rafhyc: An architecture for constructing resilient services on federated hybrid clouds. J. Grid Comput. **11**(4), 753–770 (2013)
47. SuMo-tool: https://github.com/SuMo-tool, last seen January 2014