J Grid Computing (2019) 17:797–812
https://doi.org/10.1007/s10723-019-09487-x

VM Reservation Plan Adaptation Using Machine Learning in Cloud Computing

Bartlomiej Sniezynski · Piotr Nawrocki · Michal Wilk · Marcin Jarzab · Krzysztof Zielinski

Received: 5 September 2018 / Accepted: 30 June 2019 / Published online: 13 July 2019

Abstract In this paper we propose a novel reservation plan adaptation system based on machine learning. In the context of cloud auto-scaling, an important issue is the ability to define and use a resource reservation plan, which enables efficient resource scheduling. If necessary, the plan may allocate new resources upon reservation where a sufficient amount of resources is available. Our solution allows the updating of a reservation plan initially prepared by an administrator. It makes it possible to adapt reservation plans one or more weeks ahead. Hence, it allows time for the administrator to analyze the plan and discover potential problems with resource under-provisioning or over-provisioning, which may prevent server overload in the former case and unnecessary expenses in the latter. It also makes it possible to extract and analyze the knowledge learned, which may provide useful information about resource usage characteristics. The proposed solution is tested on OpenStack using real Wikipedia server traffic data. Experimental results demonstrate that machine learning enables an improvement in resource usage.

Keywords Automated cloud resource planning · Supervised machine learning · Online plan adaptation

B. Sniezynski · P. Nawrocki (✉) · K. Zielinski
Faculty of Computer Science, Electronics and Telecommunications, Department of Computer Science, AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: piotr.nawrocki@agh.edu.pl

B. Sniezynski
e-mail: bartlomiej.sniezynski@agh.edu.pl

K. Zielinski
e-mail: kz@agh.edu.pl

M. Wilk
Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering, Department of Computer Science, AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: michal.wilk.956@gmail.com

M. Jarzab
Samsung R&D Institute Poland, Samsung Electronics, al.Bora-Komorowskiego 25C, 31-476 Krakow, Poland
e-mail: m.jarzab@samsung.com

1 Introduction

One of the key features of cloud computing is scalability, which is achieved by appropriate resource scheduling. If requirements exceed current resources, new resources are allocated. However, in order for a request for cloud resources to be fulfilled, a sufficient amount of resources should be available. This is why resource reservation is an important feature in any virtualization-based system. There are two basic use cases where reservation plays a crucial role: (i) resource reservation for immediate use; and (ii) resource reservation for future use.

The first case refers to a situation where – even if some reserved resources are to be consumed instantly – there is latency between the issuance of a resource reservation request and the actual allocation of the resources requested. During such latency period, the resource capacity in question could change, e.g. due to a failure or allocation to a different request. Therefore, system response to a request concerning resource reservation for immediate use should have a validity period indicating the time limit until which the system can reserve the resources requested. During this time limit, the system should proceed to allocation if the user wishes to use the resources requested. If allocation does not occur within the validity period, the system response to the resource reservation request in question becomes invalid and the system is not obliged to provide these resources anymore. Reservation requests for immediate use do not have a start time but may have an end time.

The second use case addresses a scenario where cloud operators may want to reserve extra resources for future use. Such a necessity could arise from predicted congestion, e.g. due to local traffic increase in office hours during a specific day or week, natural disasters etc. In such a case, a resource reservation request sent to the system includes a start time (and an end time if necessary). The start time indicates at what time the resource reserved should be available to the consumer in question. Here, the requirement is that the resources reserved should be available when the start time arrives. After the start time has arrived, the resources reserved could be allocated to the consumer(s) in question when an explicit allocation request is issued. Resource reservation requests over a future period constitute a resource reservation plan.

This paper addresses the second scenario. The availability of a comprehensive reservation plan is especially important in the case of private clouds with limited resources, which require auto-scaling functionality to preserve QoS under changing load conditions, e.g. a problem emerges with 5G technology where Network Function Virtualization (NFV) based on cloud infrastructure is used [1, 2].

The accessibility of the reservation plan within a longer time horizon may also be very important in some applications as it makes it possible to discover potential problems with insufficient resources in advance, leaving enough time to mitigate them. For instance, new resources may be added from another (public) cloud or transferred into the cloud from another tenant or application.

A reservation plan may be drawn up manually in advance by administrators. However, such planning is complex and very imprecise because it requires knowledge about future system load and accurate predictions of future demand. Therefore, many researchers [3, 4] are working on methods which enable the automatic development of such plans and adapting them to new operating conditions.

The contribution made by this paper consists in putting forward a novel reservation plan adaptation system based on machine learning (ML) which is meant to improve plan accuracy. This system allows iterative adaptations, using the MAPE-K (*Monitor-Analyze-Plan-Execute over a shared Knowledge*) pattern [5, 6], of initial versions of long-term reservation plans (e.g. made a week or a month ahead) during system operation. Machine learning algorithms are leveraged to create resource demand models on the basis of the knowledge obtained. The manner in which this system can be used with OpenStack-related resource reservation systems such as Blazar and Promise is elaborated.

In this paper we assume that resources are allocated in units that correspond to Virtual Machine (VM) profiles. To detect insufficient (or excessive) resource allocation, VM CPU usage is measured. In the context of VM auto-scaling [7, 8], an important issue is the ability to define and use a resource reservation plan [9], which guarantees the availability of a certain number of VMs within a given period. Simultaneously, it limits the ability to use non-reserved resources.

It is possible to reserve either a vector of resources (e.g., compute, storage and network) or each resource separately. The reservation of a vector of resources is much more complicated, which is why our research started with computational resources as represented by the VM. Our goal is to automatically create a VM reservation plan based on an initial one and adapt it subsequently. The initial plan may be drawn up by an administrator or created automatically from monitoring data. Currently, we assume that all VMs have the same predefined configuration. As a result, the scheduler checks CPU usage on VMs and in the case of insufficient processing power, a new VM may be allocated to the application because the reservation guarantees sufficient processing power. The proposed ML algorithm is used to adjust (increase or decrease)

the number of VMs reserved in order to fit the predicted resource utilization. As a result, a plan is produced, which may be treated as a dynamic quota on the number of VMs that guarantees a number that is neither too high nor too low. A slightly similar approach was introduced in [10] where proactive prediction-based statistical models where tested in anticipating future resource requirements. The authors viewed the issue as a time-series analysis problem and compared the results obtained both with a sliding window and without it.

It is important to point out that if a machine learning algorithm with symbolic knowledge representation (e.g. a decision tree) is used, then it is possible to visualize the knowledge learned in a way that is suitable for human analysis. As a consequence, completely new kinds of useful information about resource usage characteristics can be derived. To test the solution in a realistic cloud environment, we decided to use the OpenStack real-life cloud framework, which is among the most popular solutions, for building and managing private and public clouds [11, 12].

This paper has the following outline. At the beginning, we analyze the literature related to cloud resources and afterwards, we review OpenStack resource reservation projects. Next, we propose a Knowledge-Based Plan Adaptation Architecture using machine learning, which provides the reservation plan adaptation functionality and presents its implementation. Finally, we present experimental results and conclusions.

2 Related Work

An important issue related to the use of cloud computing is the ability to manage resources. Management mechanisms in the context of cloud computing (and in particular with respect to federated clouds) have been extensively analyzed in [13]. The authors present resource management functions in the federated cloud environment such as resource pricing, resource discovery, resource selection, resource monitoring, resource allocation, and disaster management. Important functions from the point of view of cloud computing system management are primarily resource monitoring and allocation which also enable resource prediction and reservation. However, this study does not include a broader analysis of resource reservation options aimed at optimizing the use of resources or a discussion of the possibility of using ML for resource management, which appear to be important research directions.

Research on the use of ML for cloud resource management is presented, for example, in [14], where the authors propose an intelligent resource management mechanism which allows the optimization of system operation costs and the adaptive allocation of resources. To this end, they use deep reinforcement learning mechanisms. However, there is a shortage of research in the field of auto-scaling, resource reservation and resource use prediction. In another article [15], the authors also use ML algorithms for managing cloud resources. Using neural networks, they optimize VM migration, taking into account energy consumption and SLA parameters. However, that article only deals with some aspects of managing current resources and does not discuss resource prediction and adapting resource reservation plans.

A lot of research on cloud resource management is primarily focused on resource allocation methods. For example, in [16] the authors compare resource (such as computing, storage, communication) allocation mechanisms based on their common features such as time complexity, searching mechanisms, allocation strategies, optimality, and operational environments.

An important aspect of resource management is the possibility of auto-scaling. In [7], the authors propose a system which analyzes the trend of workload changes and allows for automatic auto-scaling without user-provided metrics and threshold values. Most solutions take into account only the amount of resources used rather than their cost. However, there are studies which consider the costs of using cloud resources, such as the auto-scaling mechanism described in [17], which uses the Earliest Deadline First (EDF) algorithm to schedule tasks. This mechanism is based on a monitor-control loop and makes it possible to complete all jobs before user-specified deadlines in a cost-efficient way.

However, only the resource reservation mechanism makes it possible to set up the required resources in advance and guarantee their availability. There are few studies regarding the reservation of cloud resources. In one of them [18], the authors designed the Kraken system which enables the dynamic updating of minimum guarantees for both network bandwidth and compute resources at runtime. This system does not require prior knowledge of the resource needs of the applications but is capable of modifying reservations

at runtime. However, most studies do not take into account the reservation of cloud resources, but only forecasts of their consumption. The workload trace from the Large Hadron Collider Computing Grid was also used in [19]. This paper presents a time delay neural network (TDNN) and polynomial regression methods for predicting future workloads in the Grid or Cloud platform.

In [20], the authors predict application usage and make cloud resources available to the user in advance. The user determines initial and maximal demands. Next, VM demands are monitored in subsequent time windows (30min). The list of requested configurations is compiled. Patterns may be discovered and used for prediction. Another work about CPU, storage and memory usage prediction is [8]. It is based on models learned from historical time series with Support Vector Regression being used to predict future values. Here, resources may be allocated in advance as well.

This work is continued in [21], where two Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN) are used. They make it possible to recognize and learn long-term dependencies with up to 1,000-step time lags between relevant events. Input data consist of the CPU usage observed while output is a scaling decision. Here, resources are also measured in VMs. Therefore, each scaling decision corresponds to the number of VMs to scale up or down. This is calculated according to the difference between predicted CPU needs and current resources. The first LSTM-RNN is employed to deal with normal workloads. The second LSTM-RNN is used to deal with Slashdot situations (unpredictable increases in requests). This allows the system to detect Slashdot situations at earlier stages and perform appropriate scaling actions. Predictions are made by the network with the lower current prediction error.

The problem of resource utilization and allocation is also important in other ICT domains such as IoT where power is not taken for granted. For example, in [22] the authors are trying to predict, using weather forecasts, how much energy can be potentially harvested and on that basis make decisions which device peripherals can be used without overdrawing the power budget predicted. We have also applied an approach similar to the one described in [23, 24], where machine learning algorithms are used to create models predicting battery usage and computation time for tasks executed on mobile devices.

The need for managing and reserving resources occurs in various systems, in particular in systems that allocate virtualized resources such as VMware, Kubernetes and OpenStack [25]. Currently, the most widespread virtualization platform, used for example by telecommunications operators for NFV deployment, is OpenStack [26]. It enables the implementation of virtual network functions and ensures the appropriate level of QoS parameters. Therefore, ensuring the possibility of resource reservation on the OpenStack platform seems particularly important. Considerations regarding the OpenStack platform in the context of resource reservation are presented in the next chapter.

The analysis of existing solutions in the field of resource management in the cloud shows that there is currently no broader research that would exploit resource reservation plans while optimizing the use of cloud resources. Models that are built when developing such reservation plans require certain parameters whose values are not known in advance, and therefore it is not possible to immediately develop an optimal booking plan. Therefore, it is necessary to build such models on the basis on some experience and knowledge acquired (for instance, ML models). The research conducted by the authors is an attempt to fill the gap in the area of the use of resource reservation plans in cloud computing and their improvement with the use of ML models.

3 Resource Reservation in Open Stack Platform

As previously mentioned, one of the most popular platforms that allocate virtualized resources is OpenStack. It is an open-source solution that enables the creation of a cloud computing environment, which consists of different types of storage, servers and network devices, assuming a high level of scalability. OpenStack is an IaaS (Infrastructure as a Service) solution in which the provider offers resources to clients and enables them to create their own virtual infrastructures. This environment consists of multiple interoperable components such as Compute (Nova) or Networking (Neutron). In the context of resource reservation, OpenStack includes the Blazar service, which enables users to reserve resources of a certain type for a specific period and it leases these resources to users based on their reservations. There is also an independent solution, i.e. Promise, which makes

Table 1 Comparison between Blazar and Promise

	Blazar	Promise
Integration with OpenStack	Native – plugin	Separate application
Maturity of the project	Development phase	Development phase
Type of reserved resources	Bare metal and VMs	Bare metal
Ability to specify parameters of reserved resources	Yes	Yes
Machine / calculation priorities	No	No
API documentation	Basic	Complete
Ability to check available resources	Yes	Yes
Prediction of reservation	No	No
Using ML to determine the reservation plan	No	No
Possibility to extend functionality	Yes	Yes

it possible to reserve and manage resources and can be used with OpenStack. In our research, we use the OpenStack platform because of its popularity and the solutions available which allow for the reservation of resources. However, the resource reservation problem is general and exists in any system which allocates resources.

Blazar[1] (ex. Climate) is an OpenStack plugin that provides the resource reservation service in the OpenStack cloud for different resource types – both virtual (instances) and physical (hosts). A Blazar user can request cloud environment resources (virtual ones: instances, volumes, networks, and hardware ones: full hosts with specific RAM and CPU characteristics) to be leased to his or her project for a specific time, immediately or in future. At the moment, Blazar does not support resource availability checking. Therefore, it is possible to reserve more resources than a cloud can provide. According to project website, this feature is going to be added in subsequent releases, along with volume and stack reservation.

Promise[2] is a resource reservation and management project whose purpose is to identify NFV related requirements and to implement resource reservation for future usage by the capacity management of resource pools with respect to compute, network and storage. Reserved resources are guaranteed to a given user/client for a period expressed by its start and end times. The Euphrates implementation of Promise is built with the YangForge data modeling framework, using a shim-layer on top of OpenStack to provide Promise features such as capacity/reservation/allocation management.

We have analyzed the Blazar and Promise projects. Results of this comparison are presented in Table 1. As we can see, both projects are similar. Blazar has less comprehensive documentation, but it makes it possible to reserve VMs and to add additional functionality, which is crucial in our research. Therefore, we have decided to concentrate further research on the Blazar project in order to develop a plugin to the ML model which makes it possible to refine the resource reservation plan.

4 ML-Based Reservation Plan Adaptation

Based on the analysis of current research, we would like to propose a new solution: ML-Based Reservation Plan Adaptation. Resource monitoring data are used to build ML models, which are then used to adapt the resource reservation plan. This process is repeated and can be used to adapt the plan online. Details are described below.

4.1 Adaptation Process

The ML-Based plan adaptation process is presented in Fig. 1. A reservation plan is initially created by an administrator. This plan is used by Blazar or Promise to reserve resources. Resource usage is monitored and results are stored in a database together with information on reservation and fulfillment (resource utilization in relation to the limits assumed). This database is

[1] Blazar – https://docs.openstack.org/blazar

[2] OPNFV Promise Project –https://wiki.opnfv.org/display/promise

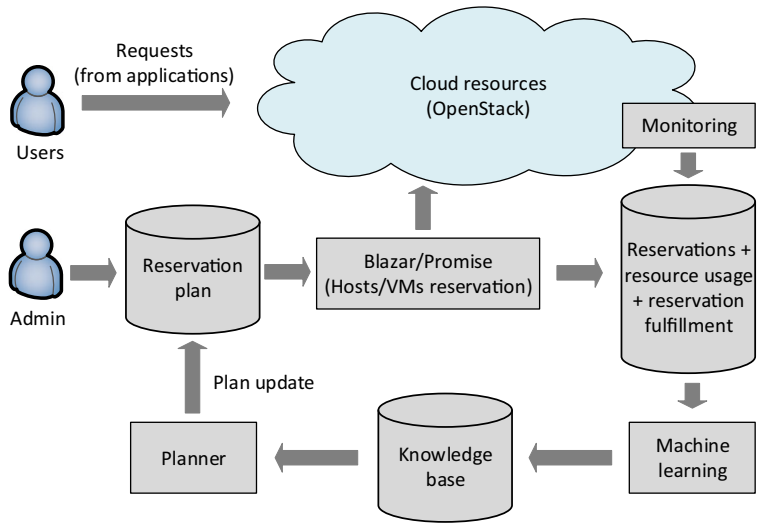

Fig. 1 ML-Based plan adaptation process

used as training data by a machine learning algorithm which creates the models stored in the knowledge base. Using this knowledge, Planner updates the reservation plan to provide sufficient resources and limit over-reservation.

Our solution corresponds to the MAPE-K approach, in which machine learning is responsible for Analysis, Blazar/Promise for *Execution* and other modules are strictly matched (Planner – Plan, Knowledge base – Knowledge, Monitoring – Monitor, Cloud resources – Managed resources). Depending on the machine learning algorithm, knowledge may have various forms. In the case of regression, we obtain predictions of values for time stamps. In the case of classification, we obtain information if the current plan reserves the appropriate amount of resources, too few or too many of them. In the case of rule or decision-tree induction, knowledge may be readable and the administrator is then able to verify and update it. In the case of Support Vector Machines or Neural Networks, the knowledge learned has a form which makes it difficult to analyze it. However, these algorithms usually exhibit better accuracy. Based on our research [23] in which Neural Networks were successfully applied to task allocation adaptation in Mobile Cloud, we decided to start with this machine learning model for experiments. Subsequently, we have also applied linear regression and decision trees to compare quality of results.

4.2 Reservation Plan and its Quality

We assume that resources are reserved according to a reservation plan. The plan $P : T \rightarrow \mathbb{N}$ is defined as a function over T – time interval of interest (horizon), which is discretized and consists of time stamps. As a result, $P(t)$ represents the amount of a resource reserved at time t.

In our research, VMs are reserved and the main resource that is monitored is CPU usage. However, other parameters (such as memory, network usage, etc.) can also be taken into account. The goal is to maintain the parameters monitored in a user-defined range (e.g. VM CPU usage should be between 70% and 90%). If it is too low or too high for a given time stamp, the reservation plan should be updated and less or more resources should be reserved in the future in such conditions. It is assumed that a time stamp is represented not only by a specific date and hour, but also by metadata, e.g. whether it is a workday, weekend or a holiday, season, vacations, etc. As a result, the knowledge learned is more general and makes it possible to modify reservations proactively for a longer time horizon. The learning process and P modification are performed periodically, e.g. at midnight, and the process continues.

The metric Q is defined, which allows us to evaluate the quality of the reservation plan P for a given obser-

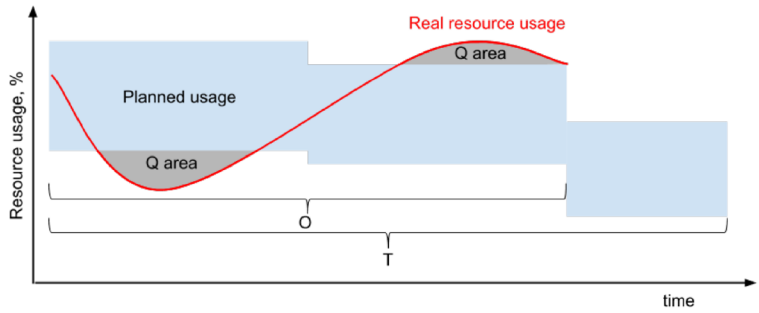

Fig. 2 Q metrics as a sum of under- and over-provisioning errors of a reservation plan

vation time period O, which is also discretized, and $O \subset T$, taking into account $R = \{r^t\}_{t \in O}$ where r^t represents the resource usage value measured at time stamp t. In our case $r^t = (r_1^t, r_2^t, \ldots, r_{P(t)}^t)$ represents CPU utilization at all VMs measured in %. The desired limits for r_i^t are represented by $L = [r_{min}, r_{max}]$. In our case L defines CPU utilization limits. Q is defined as:

$$Q(P, O, R, L) = \sum_{t \in O} \sum_{i=1}^{P(t)} d^P(r_i^t, L), \tag{1}$$

where

$$d^P(r_i^t, L) = \begin{cases} 0 & \text{if } r^t \in L \\ r_i^t - r_{max} & \text{if } r_i^t > r_{max} \\ r_{min} - r_i^t & \text{if } r_i^t < r_{min} \end{cases} \tag{2}$$

$d^P(r_i^t, L)$ is a distance (it may be a power of the distance) between the usage of resource r measured at time t and the end of the range L if r_i^t is outside the range or is equal to 0 if $r_i^t \in L$, assuming that for time stamp t the number of VMs ready to be used corresponds to $P(t)$. The Q value can be therefore seen as a sum of under- and over- provisioning errors of a reservation plan (see Fig. 2).

4.3 Adaptation Algorithm

Our hypothesis is that machine learning will allow $Q(P, O, R, L)$ to be minimized. We would like to verify if supervised machine learning will be appropriate for this purpose. The plan adaptation algorithm is presented in Fig. 3. Supervised machine learning algorithms are used to build a Knowledge base describing resource usage. The base consists of M_r model(s), which make it possible to predict resource usage at a given time point. The input for this model may consist of a time point description (e.g. hour, day of the week, holiday), historical resource usage values (e.g. from the last 12 hours) and other potentially relevant data (e.g. weather description). There may be a separate model for every resource. Various learning algorithms can be used (e.g. linear regression, random forest and artificial neural network).

The plan adaptation algorithm obtains some initial plan and monitoring data as inputs. It continuously optimizes the plan based on monitoring data from a certain observation period $O \subset T$ (e.g. one day). In line 3, it learns M_r predicting r at $t \in T$. In the next step, the plan P is updated based on M_r to maintain resource usage within limits L. We assume that M_r returns the number of VMs that should be

Fig. 3 Resource reservation plan adaptation algorithm using machine learning

Input: Initial resource reservation plan P
1 **begin**
2 Gather data R about the usage of resources reserved with regard to the current plan P;
3 Learn or update resource approximation model $M_r : T \to \mathbb{N}$;
4 Update plan (P, M_r, α);
5 Goto 2;
6 **end**

```
1  begin
2      Read plan for this day/week from file into memory;
3      Make a VM reservation using the Blazar plugin;
4      Create timer-tasks list containg time stamps which appear in the plan;
5      foreach task in timer-tasks do
6          Perform OpenStack server-create request;
7          Initialize the machine;
8          Send a signal to the Load Module that a new machine has been
             created (and move part of the traffic to relieve the already
             existing VM);
9      end
10 end
```

Fig. 4 Algorithm of the *Instance Module*

active. If machine learning algorithm allows incremental learning, the model may be updated using last R. Otherwise, it should be learned from scratch using all (or some number of) R sets collected to date. It is also possible to predict CPU usage and use it to calculate the appropriate number of VMs. For updating the plan (line 4), we propose to apply an approach similar to the one used in reinforcement learning:

$$P(t) := \lfloor P(t) + \alpha(M_r(t) - P(t)) \rfloor, \tag{3}$$

where $\alpha \in [0, 1]$ is learning speed. α values close to 0 mean slow changes to the plan. Values close to 1 mean rapid changes, but these may lead to oscillations. This parameter should be adjusted experimentally and its impact on plan adaptation speed is in fact tested in experiments (see Section 6).

5 Implementation

In order to perform tests of the solution designed, we have developed a system that applies machine-learning-based resource plan adaptation using OpenStack and Blazar. To simulate CPU load and to cover the functions which have not been provided in Blazar, we had to implement additional features. The system developed for performing experiments consists of the following three main components: the *Instance Module*, the *Load Module* and the *Machine Learning Module*. The *Instance Module* is responsible for loading the plan P to the system and executing Blazar's reservation and VM initialization functions. Its algorithm is presented in Fig. 4. Initially, it loads the plan and reserves resources in Blazar (lines 2–3). Next, it creates timer-tasks at time stamps when VMs should be created or destroyed (line 4). These tasks are executed in a loop (lines 5–9).

The *Load Module* is used during experiments only. It is not needed when the system is deployed and processing a real load. During experiments, the *Load Module* is responsible for simulating resource utilization. This module sets CPU consumption on the machines identified by their IP addresses to a specified level. Its operation is presented in Fig. 5.

The *Machine Learning Module (ML)* is responsible for running ML algorithms and updating the plan. This module is executed from time to time, e.g. once a day or whenever a new plan needs to be created. Its algorithm is presented in Fig. 6. In experiments, we use several machine learning algorithms to learn M_r. The plan is updated according to (3).

The modules described above operate in a real-life OpenStack environment. To simulate a time flow faster than in reality (time compression), the *Fast Training Module* was created. Its role is to deliver a

```
1  begin
2      Download IPs of all active machines from the Nova module;
3      Compute the CPU usage for current hour according to Wikipedia
         traffic data;
4      foreach IP in IPs do
5          Log in to the VM via SSH;
6          Restart the script that will load the CPU to the new value of the
             utilization parameter;
7      end
8  end
```

Fig. 5 Algorithm of the *Load Module*

Fig. 6 Algorithm of the *Machine Learning Module*

```
1 begin
2 |  Download the plan P used and monitoring data R;
3 |  Train the model M_r on R;
4 |  Update the plan P – correct the number of machines in the plan
  |    according to the knowledge M_r;
5 end
```

pre-trained network and therefore not only speed up the experiment time but also allow the simulation of long-term operation (e.g. for a year) according to predefined load data during this period. It trains the M_r model as the *Machine Learning Module*, assuming that the rest of the system (i.e. lease creation, instance reservation, CPU loading and telemetry data collecting mechanisms) is working as expected without any deficiencies. In particular, it assumes that the instances reserved are spawned rapidly and without any failures, CPU usage is distributed perfectly equally among all active VMs and it is constant within any given hour. Therefore, it is possible to omit OpenStack and Blazar integration and to test the concept of reservation system operation (particularly, ANN performance) during a long period in compressed time.

From the different VM images available for OpenStack, Ubuntu 16.04 LTS Xenial was selected to perform tests. The CPU of each VM was loaded to a specified percentage by using the *stress-ng* library.[3] With respect to monitoring, the Ceilometer[4] and Gnocchi 4.0[5] plugins were employed. The former tracks VM parameters within user-defined time intervals to measure r^t at O time stamps. The latter is recommended for the aggregation of measurements and as a metric data storage backend. As a result, the R set was created by querying CPU data usage for each VM via a REST service. OpenStack and plugins were in the Pike release version.

6 Evaluation

Evaluation is divided into two parts: learning effectiveness and OpenStack integration. The first part was performed using the *Fast Training Module*. This made it possible to speed up the experiments (e.g. there was no need to wait for VM start) and requests from the one-year period being used for experiments were processed in around 15 minutes. This part of evaluation is discussed below. In the integration experiment, we tested the entire system consisting of three modules with the OpenStack framework to check if it performs well. Test results allow us to claim that the integration was successful and all modules cooperated well. Such tests are time-consuming as VM startup takes about 10 minutes and the monitoring system has limited resolution, so it is impossible to apply simulated time compression to speed it up.

During the experiments, a virtual machine with the following parameters was used: 4 cores (Intel Xeon CPU E5-2680, 2.70 GHz), 16 GB of operating memory and 100 GB of disk space. On the virtual machine, OpenStack and Blazar software (Pike release), Gnocchi 4.0, deeplearning4j (version 0.8.0) and Weka (version 3.8.2) were installed.

The data used in the experiments to generate load originate from the Wikipedia monitoring system[6] and represent requests submitted to Wikipedia servers (page views of the Polish Wikipedia Project to be more specific) in 2015. For every hour of 2015, there is a file in which the number of page requests during that hour is stated. We assume that 1K requests generate a 1% load on a single VM. This served as an input for the Load Module described in Section 5.

Measurements were aggregated every hour. The reservation plan P stores the number of VMs for every hour for T representing one day. For the first day, a constant initial value was selected manually. Subsequently, it was modified using a machine learning algorithm.

Initially, we applied a Multilayer Perceptron Network from the DL4J Java library to create the M_n model. Training data were prepared by processing each hour of load data using the algorithm presented in Fig. 7. As a result, training data consist of the pairs (t, n_t), where t is described by the following

[3] Stress-ng tool manual – http://manpages.ubuntu.com/manpages/xenial/man1/stress-ng.1.html

[4] Ceilometer project documentation – https://docs.openstack.org/ceilometer/pike/

[5] Gnocchi project site – https://gnocchi.xyz/

[6] Wikipedia server data – http://dumps.wikimedia.org/other/pagecounts-raw/ and http://grafana.wikimedia.org

Fig. 7 Algorithm for transforming load data ($nof\_requests$) from hour t_n into a training example

```
1  begin
       input  : t_n, nof_requests, L = [r_min, r_max]
       output: Training example
2      hour = extract hour from t_n; hour = hour/23;
3      date = extract date from t_n;
4      day_of_the_week= extract day of the week from date;
5      if date ∈{2015/01/01, 2015/01/06, 2015/04/05, 2015/04/06,
         2015/05/01, 2015/05/03, 2015/05/24, 2015/06/04, 2015/08/15,
         2015/11/01, 2015/11/11, 2015/12/25, 2015/12/26} then holiday =
         true;
6      else holiday = false;
7      if day_of_the_week is between 'Monday' and 'Friday' then workday
         = true;
8      else workday = false;
9      if day_of_the_week = 'Saturday' then Saturday = true;
10     else Saturday = false;
11     if day_of_the_week = 'Sunday' then Sunday = true;
12     else Sunday = false;
13     if date is between 2015/06/27 and 2015/08/31 then
         summer_vacations = true;
14     else summer_vacations = false;
15     if date is between 2015/01/01 and 2015/06/30 or date is between
         2015/10/01 and 2015/12/31 then academic_year= true;
16     else academic_year = false;
17     cpu_usage = nof_requests/1000;
18     mid = (r_min + r_max)/2;
19     n_t = max(round(cpu_usage/mid), 1);
20     return ((hour, day_of_the_week, workday, Saturday, Sunday,
         summer_vacations, academic_year), n_t);
21 end
```

attributes: hour, holiday, workday, Saturday, Sunday, summer vacation, academic year. The first attribute is a natural number from 0 to 23, which is normalized to the range [0, 1]. The rest of attributes are binary (0 or 1) and represent holidays, summer vacations, and the academic year. These calendar dates are real[7] and their impact on Wikipedia logs is noticeable. Poland lies within a single time zone, which simplifies the relationship between the time of the day and the resource load level considerably. n_t is the number of VMs which keeps VM loads closest to the middle of the desired range L during t (lines 18–19), $L = [50\%, 70\%]$. t is an input vector for the model. n_t is its desired output. It is normalized to the range [0, 1].

Four neural network architectures have been tested. These are shown in Table 2. Two of them (architecture 1 and 2) included one hidden layer each and the rest included two hidden layers each. All layers are fully connected. We have tested the following learning rates: 10^{-2}, 10^{-3}, 10^{-5}, 10^{-6} and 10^{-7}. Weights are updated at the end of each day via the stochastic gradient descent algorithm using data from the day

Table 2 Network architectures tested – numbers of neurons in layers

Network architecture	1	2	3	4
Input	7	7	7	7
Hidden 1	7	14	14	21
Hidden 2	N/A	N/A	14	14
Output	1	1	1	1

[7]Holidays fall on the following days: 2015/01/01, 2015/01/06, 2015/04/05, 2015/04/06, 2015/05/01, 2015/05/03, 2015/05/24, 2015/06/04, 2015/08/15, 2015/11/01, 2015/11/11, 2015/12/25, 2015/12/26. Summer vacations are from 2015/06/27 to 2015/08/31. Academic year is from 2015/01/01 to 2015/06/30 and from 2015/10/01 to 2015/12/31.

in question. The number of epochs is equal to 1,000, and the number of iterations is set to 1. The alpha parameter is disabled. Network 4 with a learning rate of 10^{-6} exhibited the best performance in minimizing the Q value, hence it was used exclusively in further experiments.

In our case, according to (1), Q represents a sum of loads that are outside of the L range over all VMs and hours of a day (O represents 24 hours). The figures present average $Q(t)/h$ values, i.e. Q value per hour (Q is divided by 24).

The chart in Fig. 8 presents the 14-day moving average of the $Q(t)/h$ value for the entire year and four learning rates: $\alpha = 0.1$, $\alpha = 0.2$, $\alpha = 0.5$ and $\alpha = 1.0$. $Q(t)$ for the day is computed as defined in (1), where $t \in O = (0, 1, 2, \ldots, 23)$.

Plans were generated one day ahead. As we can see, machine learning makes it possible to improve the Q value. For $\alpha = 1.0$, the starting value is ca. 140. At the end of the year, this drops to around 10 which means that the average reservation error (summed over all VMs) per hour is equal to 10%. The improvement is the most pronounced at the beginning, because the learning algorithm obtains new data, which has a large impact on system performance. Subsequently, the Q value stabilizes. Reducing the learning rate to $\alpha = 0.5$ results in a slight deterioration of results. For $\alpha = 0.2$, the performance is much worse. Moreover, there are two periods when Q increases by up to approximately 40 (around days 190 and 320), while for larger α values there are no such increases. Results

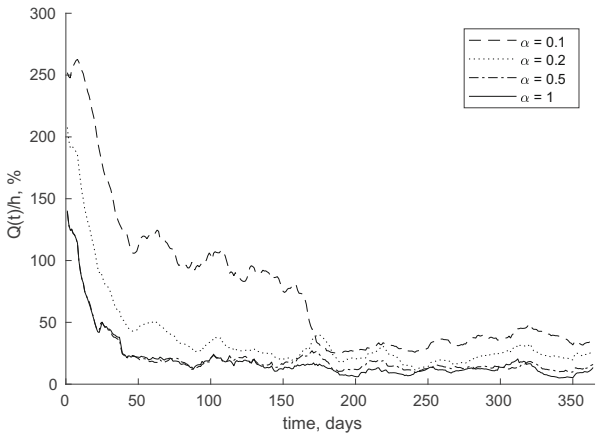

Fig. 8 14-day simple moving average of the $Q(t)/h$ value for 365 days

for $\alpha = 0.1$ are the worst. At the beginning, there is a decrease of performance instead of an increase. For this value, the plan adaptation process is incomparably slower.

Having analyzed the results, we may formulate the following conclusions. The system operates correctly and allows the plan to be adapted to decrease Q values. For plans created one day ahead, α should not be much smaller than 0.5 because for such values plan updates are too small. To achieve good results with Wikipedia data, the system needs data from about two months, starting from a very simple plan according to which four VMs are needed all the day. This period will be shorter if the initial plan is closer to the optimal one, which will be the case when such a system is applied in practice. Also importantly, any cloud monitoring data recorded beforehand may be used in advance to train the initial network and prepare it for real-life applications.

The next three charts in Fig. 9 present 7-day length periods for the α value equal to 1: at the beginning of learning, 6–12 January (top chart), intermediate results, 1–7 May (middle chart), and at the end of the process, 7–14 December (bottom chart). Each chart contains four values:

- r_{min} – lower desired usage bound (VM_count * 50%);
- r_{max} – upper desired usage bound (VM_count * 70%);
- *Real* – real total CPU usage summed over all VMs (in %);
- Q – sum of loads that are outside of the L range over all VMs and hours of a day divided by 24 ($Q(t)/h$).

where VM_count is the number of virtual machines active during any particular hour.

A significant improvement in reservation quality can be seen between each of the periods. In the last period, even rapid peaks are well handled and fit perfectly into the desired workload per machine defined by the user. At the beginning, daily fluctuations in usage (daily cycle) are the largest cause of highly inadequate provisioning (see Fig. 9, top chart). Machine learning makes it possible to predict this cycle (see Fig. 9, bottom chart). As a result, $Q(t)/h$ values drop as much as around 10 times.

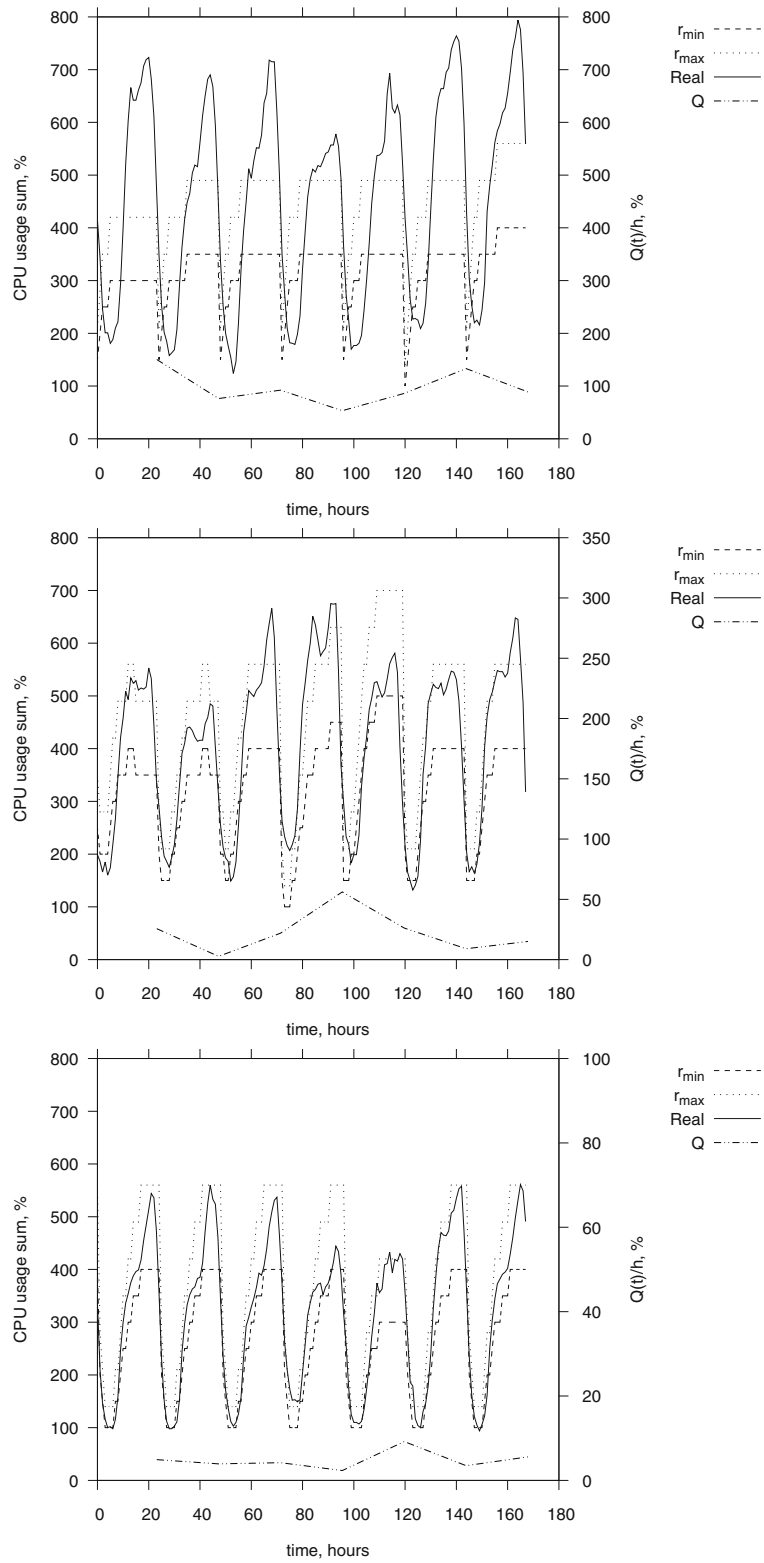

Fig. 9 Plan adaptation results from January (top chart), May (middle chart), December (bottom chart) for Neural Network with $\alpha = 1$

Table 3 Comparison of selected machine learning algorithms

Algorithm	Sum of Q values	Learning time, s
Neural Network	116445	3.568
Linear regression	323706	**0.01**
RepTree	**107958**	0.02
M5P	185589	0.06

Bold entries means the best results, i.e. the smallest sum of values and the shortest time

We have also compared performance of the best Neural Network with three other machine learning algorithms: linear regression, RepTree and M5P implemented in Weka [27]. To compare results, we have used the sum of Q values over all 365 days. For linear regression, we have checked ridge parameter values from the $[10^{-10}, 1000]$ range. The best performance was achieved for 10^{-5}. RepTree was executed with and without pruning for the maximum tree depth limit set to 1, 2, 5 and with no limit. The best result was achieved without pruning and without a tree depth limit. M5P was executed with pruning and smoothing switched on and off. The minimum number of instances per leaf was set to 2, 4 and 6. The best results were achieved without pruning and smoothing for a minimum number of instances per leaf set to 2. We have also measured learning time. The results are presented in Table 3.

As we can see, the RepTree algorithm performed the best, 7% better than the Neural Network. We have also checked performance of RepTree during 7-day periods, similarly as for the Neural Network. The results are presented in Fig. 10. The results show that RepTree learns faster than the Neural Network. Even in January, it achieves relatively good results. This explains the lower sum of Q values than for the Neural Network, which is in the second place. However, the quality of RepTree predictions does not improve as much as in the case of the Neural Network. In December, Q values are around 20, while for the Neural Network these are below 10. As a result, in the long term, the Neural Network model outperforms.

The overhead of the solution proposed consists of two components: telemetry services and model learning. In the production system, telemetry should be turned on independently of our solution (e.g. for alarms to be generated in case of problems). We have no influence over this overhead component. Model learning is executed once a day and this process depends on the learning algorithm chosen. The learning time for the Neural Network is the highest: 3.5 seconds. However, it is low enough not to introduce any significant overhead.

7 Conclusions

The approach proposed allows for online (during system usage), autonomous plan adaptation. This is a closed-loop solution: it automatically generates and adopts the plan using monitoring data and comparing reservations to demands. As it was successfully demonstrated, it is possible to verify the updated plan in advance and discover potential problems within both short and longer periods. This is a very important aspect for planning the operation of both private and public clouds.

The paper presents a rather general framework. The reservation process is based on load prediction, and thus various ML models can be used in the solution proposed. The application of machine learning models is very much dependent on the availability of real-life historical data representing load changes over fairly long periods and also on online access to current system load. Detailed results of the evaluation performed in this study are dependent upon characteristic features of the load observed in the system, therefore it is difficult to arrive at general conclusions about numerical results.

In a real-world application, the historical data could be used for the initial training and parameter setting of ML models before their deployment and further refinement during system runtime. The initial training, as it was demonstrated, could be performed in a compressed time scale. This makes it possible to use already pre-trained models and observe the advantage resulting from the use of ML algorithms during system operation almost immediately.

Experimental results demonstrate that this solution improves cloud resource utilization. It was tested on VM reservations, but the solution is general and may be applied to other resources as well.

Performance of four machine learning algorithms was compared: Neural Networks, linear regression,

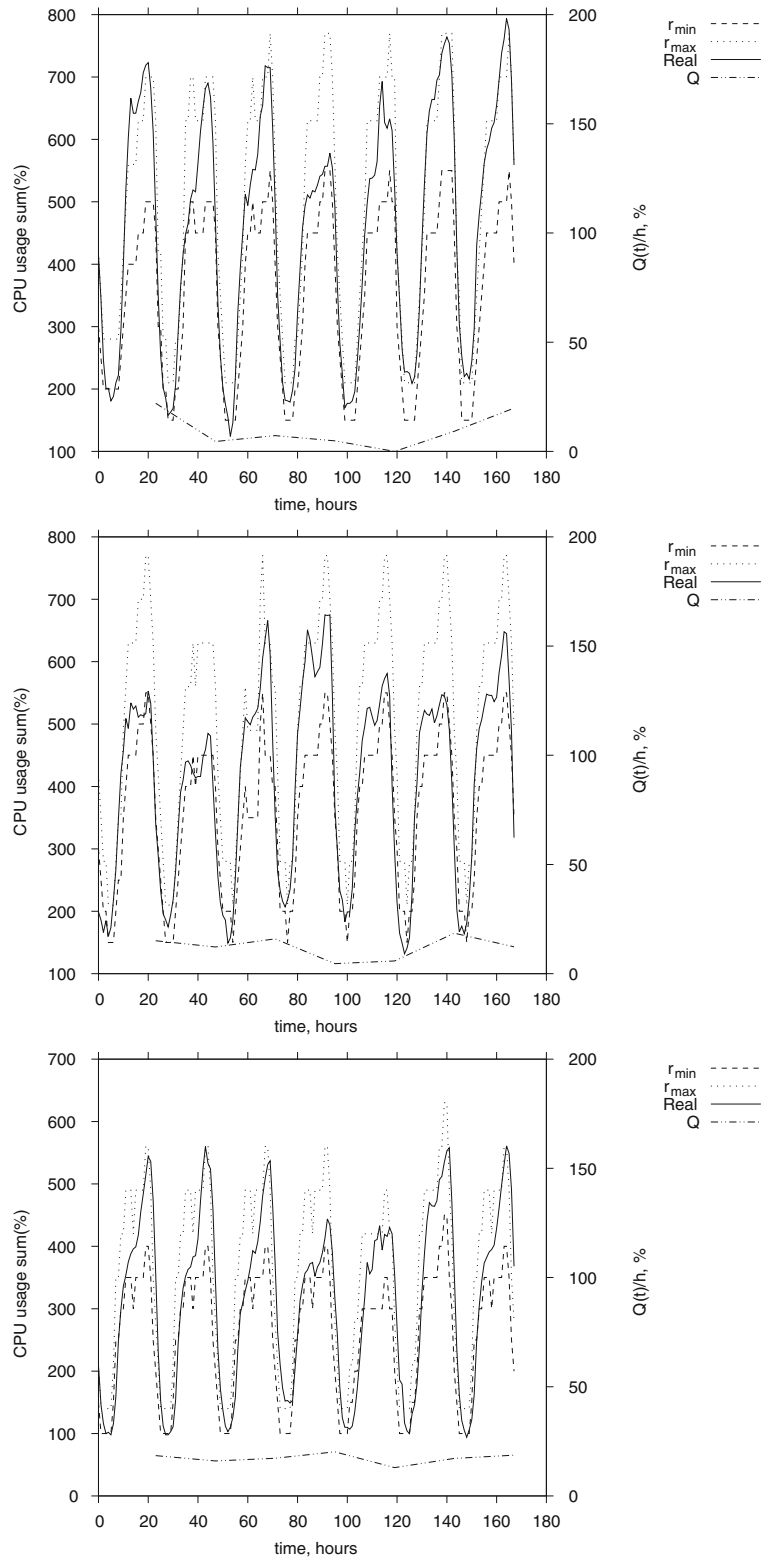

Fig. 10 Plan adaptation results from January (top chart), May (middle chart), December (bottom chart) for RepTree with $\alpha = 1$

RepTree and M5P. RepTree was learning faster than the Neural Network; however, the Neural Network ultimately yielded better predictions.

There are many areas which should be analyzed in the future research. More experiments should be performed in a real-life environment. The solution proposed is general and it should work with other resources as well. Therefore, we would like to conduct experiments (e.g. involving memory, network or GPU compute) in a real-life environment. Additionally, we would also like to consider more complex cases where multiple resources are taken into account simultaneously and also how changes to hardware (e.g. new processors) influence system performance. We are also planning to enable various VM configurations. Last but not least, hybrid models (e.g. consisting of the Neural Network and RepTree) should be examined, since they can combine the fast learning of RepTree with the better accuracy of Neural Networks.

Acknowledgements The research presented in this paper was supported by Samsung Research Poland.

References

1. Abdelwahab, S., Hamdaoui, B., Guizani, M., Znati, T.: Network function virtualization in 5g. IEEE Commun. Mag. **54**(4), 84–91 (2016)
2. Costa-Requena, J., Santos, J.L., Guasch, V.F., Ahokas, K., Premsankar, G., Luukkainen, S., Pérez, O.L., Itzazelaia, M.U., Ahmad, I., Liyanage, M., Ylianttila, M., de Oca, E.M.: Sdn and nfv integration in generalized mobile network architecture. In: 2015 European Conference on Networks and Communications (EuCNC), pp. 154–158 (2015)
3. Kukreja, S., Dalal, S.: Performance analysis of cloud resource provisioning algorithms. In: Saeed, K., Chaki, N., Pati, B., Bakshi, S., Mohapatra, D.P. (eds.) Progress in Advanced Computing and Intelligent Engineering, pp. 593–602. Springer, Singapore (2018)
4. Jala, J., Rao, K.R.H.: Qos-based technique for dynamic resource allocation in cloud services. In: Smys, S., Bestak, R., Chen, J.I.-Z., Kotuliak, I. (eds.) International Conference on Computer Networks and Communication Technologies, pp. 65–73. Springer, Singapore (2019)
5. Maurer, M., Breskovic, I., Emeakaroha, V.C., Brandic, I.: Revealing the mape loop for the autonomic management of cloud infrastructures. In: 2011 IEEE Symposium on Computers and Communications (ISCC), pp. 147–152 (2011)
6. Arcaini, P., Riccobene, E., Scandurra, P.: Modeling and analyzing mape-k feedback loops for self-adaptation. In: 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 13–23 (2015)
7. Lin, C.-C., Wu, J.-J., Liu, P., Lin, J.-A., Song, L.-C.: Automatic Resource Scaling for Web Applications in the Cloud, pp. 81–90. Springer, Berlin (2013)
8. Hu, R., Jiang, J., Liu, G., Wang, L.: Efficient resources provisioning based on load forecasting in cloud. Sci. World J., 2014 (2014)
9. Wang, W., Niu, D., Li, B., Liang, B.: Dynamic cloud resource reservation via cloud brokerage. In: 2013 IEEE 33rd International Conference on Distributed Computing Systems, pp. 400–409 (2013)
10. Islam, S., Keung, J., Lee, K., Liua, A.: Empirical prediction models for adaptive resource provisioning in the cloud. Futur. Gener. Comput. Syst. **28**, 155–162 (2012)
11. Celesti, A., Mulfari, D., Fazio, M., Puliafito, A., Villari, M.: Evaluating alternative daas solutions in private and public openstack clouds. Softw.: Pract. Exper. **47**(9), 1185–1200 (2017)
12. Kabiri, M.N., Wannous, M.: An experimental evaluation of a cloud-based virtual computer laboratory using openstack. In: 2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), vol. 00, pp. 667–672 (2018)
13. Liaqat, M., Chang, V., Gani, A., Hamid, S.H.A., Toseef, M., Shoaib, U., Ali, R.L.: Federated cloud resource management: Review and discussion. J. Netw. Comput. Appl. **77**, 87–105 (2017)
14. Zhang, Y., Yao, J., Guan, H.: Intelligent cloud resource management with deep reinforcement learning. IEEE Cloud Comput. **4**(6), 60–69 (2017)
15. Witanto, J.N., Lim, H., Atiquzzaman, M.: Adaptive selection of dynamic vm consolidation algorithm using neural network for cloud resource management. Futur. Gener. Comput. Syst. **87**, 35–42 (2018)
16. Qureshi, M.B., Dehnavi, M.M., Min-Allah, N., Qureshi, M.S., Hussain, H., Rentifis, I., Tziritas, N., Loukopoulos, T., Khan, S.U., Xu, C.-Z., Zomaya, A.Y.: Survey on grid resource allocation mechanisms. J. Grid Comput. **12**(2), 399–441 (2014)
17. Mao, M., Humphrey, M.: Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, pp. 49:1–49:12. ACM, New York (2011)
18. Fuerst, C., Schmid, S., Suresh, L., Costa, P.: Kraken: Online and elastic resource reservations for cloud datacenters. IEEE/ACM Trans. Network. **26**(1), 422–435 (2018)
19. Imam, M.T., Miskhat, S., Rahman, M., Amin, M.A.: Neural network and regression based processor load prediction for efficient scaling of grid and cloud resources. In: 14th International Conference on Computer and Information Technology, ICCIT 2011, pp. 333–338, 12 (2011)

20. Singh, H., Randhawa, R.: Dynamic resource prediction and allocation in clouds using pattern matching. Indian J. Sci. Technol., **9**(47) (2016)
21. Shahin, A.A.: Automatic cloud resource scaling algorithm based on long short-term memory recurrent neural network. arXiv:1701.03295 (2016)
22. Szydlo, T., Brzoza-woch, R.: Predictive power consumption adaptation for future generation embedded devices powered by energy harvesting sources. Microprocess. Microsyst. Embedded Hardware Des. **39**(4–5), 250–258 (2015)
23. Nawrocki, P., Sniezynski, B.: Autonomous context-based service optimization in mobile cloud computing. J. Grid Comput. **15**(3), 343–356 (2017)
24. Nawrocki, P., Sniezynski, B.: Adaptive service management in mobile cloud computing by means of supervised and reinforcement learning. J. Netw. Syst. Manag. **26**(1), 1–22 (2018)
25. Dong, D., Xiong, H., Castañé, G.G., Stack, P., Morrison, J.P.: Heterogeneous resource management and orchestration in cloud environments. In: Ferguson, D., Muñoz, V.M., Cardoso, J., Helfert, M., Pahl, C. (eds.) Cloud Computing and Service Science, pp. 61–80. Springer International Publishing, Cham (2018)
26. Chen, J., Chen, Y., Tsai, S., Lin, Y.: Implementing nfv system with openstack. In: 2017 IEEE Conference on Dependable and Secure Computing, pp. 188–194 (Aug 2017)
27. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann (2016)