Minarolli *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2017) 6:4
DOI 10.1186/s13677-017-0074-3

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

# Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing

Dorian Minarolli[1*], Artan Mazrekaj[2] and Bernd Freisleben[3,4]

**Abstract**

Dynamic workloads in cloud computing can be managed through live migration of virtual machines from overloaded or underloaded hosts to other hosts to save energy and/or mitigate performance-related Service Level Agreement (SLA) violations. The challenging issue is how to detect when a host is overloaded to initiate live migration actions in time. In this paper, a new approach to make long-term predictions of resource demands of virtual machines for host overload detection is presented. To take into account the uncertainty of long-term predictions, a probability distribution model of the prediction error is built. Based on the probability distribution of the prediction error, a decision-theoretic approach is proposed to make live migration decision that take into account live migration overheads. Experimental results using the CloudSim simulator and PlanetLab workloads show that the proposed approach achieves better performance and higher stability compared to other approaches that do not take into account the uncertainty of long-term predictions and the live migration overhead.

**Keywords:** Virtual machine live migration, Long-term resource prediction, Probabilistic model, Cloud computing

## Introduction

Cloud computing is a promising approach in which resources are provided as services that can be leased and released by users through the Internet in an on-demand fashion [1]. One of the widely used cloud computing service models is Infrastructure as a Service (IaaS) [2] where raw computing resources are provided in the form of Virtual Machines (VMs) to cloud consumers charged for the resources consumed. Virtualization approaches such as Xen [3] and VMware [4] allow infrastructure resources to be shared in an effective manner. VMs also make it possible to allocate resources dynamically according to varying demands, providing opportunities for the efficient use of computing resources, as well as the optimization of application performance and energy consumption.

One of the main features virtualization technology offers for dynamic resource allocation is live migration of VMs [5]. It allows cloud providers to move away VMs from overloaded hosts to keep VM performance to SLA levels and to dynamically consolidate VMs to fewer hosts to save energy when the load is low. Using live migration and applying online algorithms that make live migration decisions, it is possible to manage cloud resources efficiently by adapting resource allocation to VM loads, keeping VM performance levels according to SLAs and lowering energy consumption of the infrastructure.

An important problem in the context of live migration is to detect when a host is overloaded or underloaded. Most of the state-of-the-art approaches are based on monitoring resource usage, and if the actual or the predicted next value exceeds a specified threshold, then a host is declared as overloaded. However, live migration is an expensive action, expressed as VM performance violations. The problem with existing approaches is that basing decisions for host overload detection on a single resource usage value or a few future values can lead to hasty decisions, unnecessary live migration overhead and stability issues.

A more promising approach is to base live migration decisions on resource usage predictions several steps ahead in the future. This not only increases stability by

*Correspondence: dminarolli@fti.edu.al
[1]Department of Computer Engineering, Polytechnic University of Tirana, Tirana, Albania
Full list of author information is available at the end of the article

performing migration actions only when the load persists for several time intervals, but also allows cloud providers to predict overload states before they happen. On the other hand, predicting further into the future increases the prediction error and the uncertainty, thus diminishing the benefits of long-term prediction. Another important issue is that live migration actions should only be performed if the penalty of SLA violations is larger than the penalty of the live migration overhead.

In this paper, a new approach for host overload and underload detection is presented based on long-term resource usage predictions that take into account the prediction uncertainty and the live migration overhead. More specifically, the paper makes the following contributions:

- A novel approach to dynamically allocate resources to VMs in an IaaS cloud environment is presented. It combines local and global VM resource allocations. Local resource allocation means allocating CPU resource shares to VMs according to the current load. Global resource allocation means performing live migration actions when a host is overloaded or underloaded in order to mitigate VM performance violations and reduce the number of hosts to save energy.
- A novel approach based on long-term resource usage predictions is presented to detect when a host is overloaded or underloaded. For long-term predictions, a supervised machine learning approach based on Gaussian Processes [6] is used.
- To take into account the uncertainty of long-term predictions for overload detection, a novel probabilistic model of the prediction error is built online using the non-parametric kernel density estimation [7] method.
- To take into account VM live migration overheads, a novel decision-theoretic approach based on a utility function is proposed. It performs live migration actions only when the predicted utility value (penalty) of SLA violations is greater than the utility value of live migration overhead.

The proposed approach is experimentally compared to other approaches: (a) an approach that relies on short-term predictions, (b) an approach that makes long-term predictions without taking into account prediction uncertainty, (c) an approach that makes long-term predictions taking into account prediction uncertainty, but not applying decision theory for considering live migration overhead, and (d) a the state-of-the-art approach based on Local Regression Detection [8] for host overload detection. Experimental evaluations based on the CloudSim [9] simulator and PlanetLab [10] workloads show that the proposed approach achieves better performance and stability compared to the other approaches.

The paper is organized as follows. "Resource manager architecture" section presents the overall architecture of the resource management approach. "VM agent" section discusses the functionality of the VM agent. "Host agent" section explains the duties of the host agent: probabilistic and decision-theoretic overload, underload and not-overload detection. "Global agent" section presents the global agent, and "VM SLA violation" section discusses VM SLA violation metrics. In "Experimental evaluation" section, the experimental evaluation is presented. Related work is discussed in "Related work" section. The last section concludes the paper and outlines areas for future research.

## Resource manager architecture

This work focuses on managing an IaaS cloud in which several VMs run on physical hosts. The overall architecture of the resource manager and its main components are shown in Fig. 1. There is a VM agent for each VM that determines the resource shares to be allocated to its VM in each time interval. There is a host agent for each host that receives the resource allocation decisions of all VM agents and determines the final allocations by resolving any possible conflicts. It also detects when a host is overloaded or underloaded and transmits this information to the global agent. The global agent initiates VM live migration decisions by moving VMs from overloaded or underloaded hosts to not-overloaded hosts to mitigate SLA violations and reduce the number of hosts. In the following sections, a more detailed discussion is provided for each of the components of the resource manager.

## VM agent

The VM agent is responsible for local resource allocation decisions by dynamically determining the resource shares to be allocated to its own VM. Allocation decisions are made in discrete time intervals where in each interval the resource share to be given in the next time interval is determined. In this work, the time interval is set to 10 seconds to adapt quickly to changing load. The interval is not set to less than 10 seconds, since in long-term prediction this would increase the number of time steps to predict into the future, lowering the prediction accuracy. This time interval value is also used in previous work [11] for long-term prediction, where the same reasoning is used to make it possible to predict further into the future. Setting a larger time interval can lead to inefficiencies and SLA violations due to the lack of quick adaptation to the load variation. This dynamic allocation of resource shares permits the cloud provider to adapt the resources given to each VM according to the current load, thus keeping the required performance level with the minimum resource costs. Our work focuses on CPU allocation, but in principle the approach can be extended to other resources as

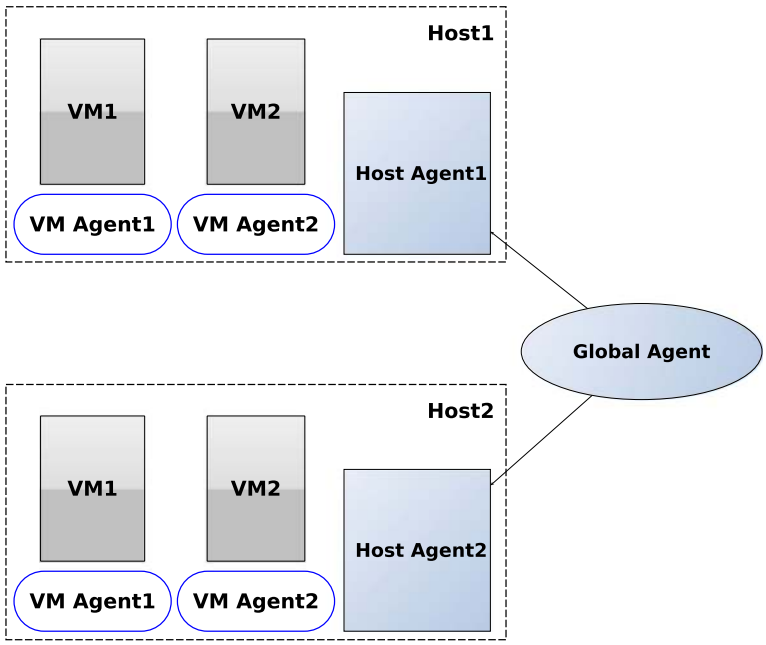


**Fig. 1** Resource manager architecture

well. More specifically, for CPU share allocation, the CPU CAP setting that most modern virtualization technologies offer is used. The CAP is the maximum CPU capacity that a VM can use, given as a percentage of the total capacity, which provides good performance isolation between VMs.

To estimate the CPU share allocated to each VM, first the value of the CPU usage for the next time interval is predicted. Then, the CPU share is calculated as the predicted CPU usage plus 10% of the CPU capacity, similar to previous work [12]. By setting the CPU CAP to leave 10% room above the required CPU usage allows us to account for prediction errors and reduces the possibility of performance-related SLA violations. To predict the next CPU usage value, a time series forecasting technique, based on the history of previous CPU usage values, is used. More specifically, a machine learning approach based on Gaussian Processes [6] is employed. Although for local resource allocation only a one step ahead prediction is needed, our VM agent predicts several steps ahead into the future to support overload detection through long-term prediction.

## Host agent

One of the duties of the host agent is to play the role of an arbitrator. It gets the CPU requirements from all VM agents, and by resolving any conflicts between them, it decides about the final CPU allocations for all VMs. Conflicts can arise when the CPU requirements of all VMs exceed the total CPU capacity. If there are no conflicts, the final CPU allocation is the same as the allocations requested by the VM agents. If there is a conflict, the host agent computes the final CPU allocations according to the following formula:

$$FinalA = \frac{A}{SumA} * TotalCapacity \quad (1)$$

where *FinalA* is the final allocation, *A* is the required allocation, *SumA* is the sum of all VMs' requested allocations and *TotalCapacity* is the total CPU capacity.

Another duty of the host agent, which is the main focus of this work, is to detect whether the host is overloaded or underloaded. This information is passed to the global agent that then initiates live migration actions for moving VMs away from overloaded or underloaded hosts according to the global allocation algorithm.

### Overload detection

For overload detection, a long-term time series prediction approach is used. Long-term prediction in the context of this work means predicting 7 time intervals ahead into the future. A straightforward way for host overload detection is as follows. A host is declared as overloaded if the actual and the predicted total CPU usage of 7 time intervals ahead into the future exceed an overload threshold. The predicted total CPU usage of a time interval into the future is estimated by summing up the predicted CPU usage values of all VMs of the corresponding time interval. The value of predicting 7 time intervals into the future is chosen such that it is greater than the estimated

average live migration time (around 4 time intervals). In this work, the average live migration time is assumed to be known and its value of 4 time intervals is estimated by averaging over all VM live migration times over several simulation experiments. In real world scenarios, this value is not known in advance, but it can be estimated based on the previous history of live migration times. Another more fine-grained approach would be to apply VM live migration modelling [13] for live migration time prediction based on relevant VM parameters. Based on this estimated live migration time, the number of time steps to predict into the future can be set to a value greater than the migration time. This is done in order to signal overload states that last longer than the live migration time. Performing live migration actions for overload states that last less than the live migration time is useless, since in this case the live migration action does not eliminate the overload state. Having a larger value than 7 time intervals is not really useful either, since this can lead to skipping some overload states that do not last long, but that can be eliminated by live migration actions. Some preliminary experiments have shown that increasing the number of prediction time intervals further into the future does not increase the stability and performance of the approach. The overload threshold value is determined dynamically based on the number of VMs and is related to the VM SLA violation metric, as explained in "VM SLA violation" section .

#### Underload detection

The host agent also detects whether a host is underloaded in order to apply dynamic consolidation by live migrating all its VMs to other hosts and turning off the host to save energy. Here, long-term time series predictions of CPU usage are also used. The host is declared as underloaded if the actual and the predicted total CPU usage of 7 time intervals ahead into the future are less than an underload threshold. Again, the value of 7 time intervals is long enough to skip short-term underload states, but not too long as to miss any opportunity for consolidation. The underload threshold value is a constant value, and in this work it is set to 10% of the CPU capacity, but it can be configured by the administrator according to his or her preferences for consolidation aggressiveness.

#### Not-overload detection

To make live migration decisions, the global agent needs to know the hosts that are not overloaded in order to use them as destination hosts for VM live migrations. A host is declared as not overloaded if the actual and the predicted total CPU usage of 7 time intervals ahead into the future is less than the overload threshold. The actual and the predicted total CPU usage of any time interval is estimated by summing up the actual and predicted CPU usage of all existing VMs plus the actual and the predicted CPU usage of the VM to be migrated. The purpose is to check whether the destination host remains not overloaded after the VM has been migrated.

#### Uncertainty in long-term predictions

Overload or underload detection based on long-term predictions carries with it the uncertainty of correct predictions, which can lead to erroneous decisions. To take into account the uncertainty of long-term predictions, the above detection mechanisms are augmented with the inclusion of a probabilistic distribution model of the prediction error.

First, the probability density function of the prediction error for every prediction time interval is estimated. Since the probability distribution of the prediction error is not known in advance and different workloads can have different distributions, a non-parametric method to build the density function online is required. In this work, a non-parametric method for probability density function estimation based on kernel density estimation [7] is used. It estimates the probability density function of the prediction error every time interval based on a history of previous prediction errors. In this work, the probability density function of the absolute value of the prediction error is used. Since there are 7 time interval predictions into the future, 7 different prediction error probability density functions are built online.

#### Probabilistic overload detection

Based on the probability density function of the prediction error, it can be estimated probabilistically, for each predicted time interval, if the future total CPU usage will be greater than the overload threshold. In the following, for convenience, the future total CPU usage is just called the future CPU usage. This is achieved by Algorithm 1 that returns true or false with some probability whether the future CPU usage will be greater than the overload threshold.

First, the algorithm finds the probability that the future CPU usage will be greater than the overload threshold. If the predicted CPU usage is greater than the overload threshold, the difference, called max_error, between the predicted CPU usage and overload threshold, is found. For the future CPU usage to be greater than the overload threshold, the absolute value of the error (i.e., the difference between predicted and future value) should be less than max_error. Based on a cumulative distribution function of the prediction error, the probability that the prediction error is less than max_error, i.e., the future CPU usage is greater than the overload threshold, is found. Since it can happen that the future CPU usage will be greater than the overload threshold, and also that the prediction error will be greater than max_error, the

**Algorithm 1: IsUtilizationOver**

```
1  if Pred_Total_Util >= OverThreshold then
2  |   max_error=Pred_Total_Util - OverThreshold
3  |   probability=CumulativeProbability(max_error)
4  |   probability=(probability+1)/2
5  end
6  else
7  |   max_error=OverThreshold - Pred_Total_Util
8  |   probability=CumulativeProbability(max_error)
9  |   probability=(probability+1)/2
10 |   probability=1-probability
11 end
12 probability=(probability)*100
13 randnum=rand.nextInt(100)
14 if randnum < probability then
15 |   return true
16 end
17 else
18 |   return false
19 end
```

probability that this happens, given as (1-probability)/2, is added to the calculated probability to yield the final probability (probability+1)/2. If the predicted CPU usage is less than the overload threshold, by the same approach, first, the probability that the future CPU usage will be less than the overload threshold is found. Then, the probability that the future CPU usage will be greater than the overload threshold is given as (1-probability). Finally, the algorithm returns true with the estimated probability.

Algorithm 1 returns the overload condition probabilistically only for a single prediction time interval. Therefore, to declare the host as overloaded, the actual CPU usage should exceed the overload threshold, and the algorithm should return true for all 7 prediction time intervals in the future.

The interpretation of taking into account prediction uncertainty in overload detection is as follows. Although CPU prediction can lead to values above the overload threshold, there is some probability, due to the uncertainty of prediction, that the CPU utilization will be lower than the threshold. This means that for some fraction of the time the host will not be considered as overloaded. This increases the stability of the approach, as shown by the lower number of live migrations for the probabilistic overload detection approach, compared to other approaches. Furthermore, when CPU prediction is lower than the overload threshold, there is some probability that the CPU utilization will be greater than the threshold. This means that for some fraction of the time the host will be considered as overloaded. In summary, we can say that the host is considered as overloaded or not in proportion to the uncertainty of prediction, which is the right thing to do, as supported by our good experimental results compared to approaches that do not take prediction uncertainty into account.

**Algorithm 2: IsUtilizationNotOver**

```
1  if Pred_Total_Util >= OverThreshold then
2  |   max_error=Pred_Total_Util - OverThreshold
3  |   probability=CumulativeProbability(max_error)
4  |   probability=(probability+1)/2
5  |   probability=1-probability
6  end
7  else
8  |   max_error=OverThreshold - Pred_Total_Util
9  |   probability=CumulativeProbability(max_error)
10 |   probability=(probability+1)/2
11 end
12 probability=(probability)*100
13 randnum=rand.nextInt(100)
14 if randnum < probability then
15 |   return true
16 end
17 else
18 |   return false
19 end
```

#### Probabilistic not-overload detection

To take into account the uncertainty of long-term predictions in detecting whether a host is not overloaded, Algorithm 2 is proposed. It returns true, with some probability, if the future CPU usage of some prediction time interval will be less than the overload threshold. The host is declared as not overloaded if the actual CPU usage is less than the overload threshold, and Algorithm 2 returns true for all 7 prediction time intervals in the future.

#### Probabilistic underload detection

To detect whether a host is underloaded, Algorithm 3 is proposed. It returns true, with some probability, if the future CPU usage of some prediction time interval will be less than the underload threshold. The host is declared as underloaded if the actual CPU usage is less than the underload threshold, and Algorithm 3 returns true for all 7 prediction time intervals into the future.

#### Decision-theoretic overload detection

The above improvements make it possible to take into account the uncertainty of long-term predictions in the detection process, but do not take into account the live migration overhead. In this section, a further approach, based on decision theory, is presented that performs live

**Algorithm 3: IsUtilizationUnder**

```
1  if Pred_Total_Util >= UnderThreshold then
2  |   max_error=Pred_Total_Util - UnderThreshold
3  |   probability=CumulativeProbability(max_error)
4  |   probability=(probability+1)/2
5  |   probability=1-probability
6  end
7  else
8  |   max_error=UnderThreshold - Pred_Total_Util
9  |   probability=CumulativeProbability(max_error)
10 |   probability=(probability+1)/2
11 end
12 probability=(probability)*100
13 randnum=rand.nextInt(100)
14 if randnum < probability then
15 |   return true
16 end
17 else
18 |   return false
19 end
```

migration actions only if SLA violations due to future host overload states are greater than the penalty of VM live migration.

Applying decision theory requires us to define a utility function that should be optimized. In this work, the utility function value represents the penalty of the host SLA violation or the penalty of live migration overhead. A SLA is a contract between the cloud provider and the cloud consumer that defines, among others, the performance level the cloud provider should conform to and the penalty costs of violating it. In this work, a host SLA violation is defined as the situation when the total CPU usage of the host exceeds the overload threshold for 4 consecutive time intervals. The penalty of host SLA violation is the percentage of the CPU capacity that the total CPU usage exceeds the overload threshold for all 4 consecutive time intervals. The penalty value can be converted to a monetary value with some conversion function, but here it is treated as a CPU capacity percentage value.

Since each VM live migration is associated with some performance degradation, a penalty value for each VM live migration action can be defined in a SLA contract. More concretely, a SLA violation penalty value (expressed also as a percentage of the CPU capacity) for each time interval during VM live migration is defined. The VM live migration SLA violation penalty is defined as the sum of all SLA violation penalties for all time intervals that the VM live migration lasts.

The proposed decision-theoretic approach tries to minimize the host SLA violation penalty (utility value), taking into account the VM live migration SLA violation penalty. In the following, the term utility value will be used instead of host SLA violation penalty. First, the expected utility value of the future host overload state is estimated. The expected utility is given by the sum of the expected utility values of all 4 consecutive future time intervals from interval 4 to interval 7. It is started from time interval 4 instead of time interval 1 in order to capture an overload state before it happens and eliminates it through VM live migration that takes, on the average, 4 time intervals.

If the future CPU usage is known, then the utility of a time interval can be given just as the difference between future CPU usage and the overload threshold. Since only the predicted CPU usage is known, the expected utility value of one time interval can be calculated as follows. First, the CPU usage interval between the total CPU capacity and the overload threshold is divided into a fixed number of levels (5 in this work). Then, the CPU usage above the overload threshold (i.e., the utility value) of each level is calculated as shown in Algorithm 4.

**Algorithm 4: LevelUsage**

```
1 Interval=100-OverThreshold
2 Delta=Interval/UsageLevels
3 Start=OverThreshold+Level*Delta
4 return ((Start+(Delta/2))-OverThreshold)
```

In Algorithm 4, *Interval* is the CPU usage interval width above the overload threshold, *Delta* is the CPU usage interval width of the corresponding level, *Level* is the level number (from 0 to 4), whose utility value will be found, *UsageLevels* is the total number of levels and *Start* is the CPU usage of the start of level interval. The algorithm returns as the utility value the CPU usage value taken from the middle of the level interval. Algorithm 4 is run for each possible level to find its utility value.

Second, for any time interval, the probability that the CPU usage of some level will indeed be the future CPU usage is calculated by Algorithm 5.

*Start* and *Delta* are calculated as in Algorithm 4, *Pred_Util* is the total predicted CPU usage of the corresponding time interval, *CumProbability*() represent the cumulative distribution function used to find the probability that the prediction error is less than a certain value and *prob* represents the probability that the CPU usage of the corresponding level will be the future CPU usage. The algorithm considers three possible situations in which the level interval can be: one in which the interval does not include the predicted CPU usage and is below it, one in which the interval includes the predicted CPU usage, and one in which the interval does not include the predicted CPU usage and is above it. In each case, based on the cumulative distribution function of the prediction error,

**Algorithm 5: LevelProbability**

```
End=Start+Delta;
if Pred_Util > End then
    prob=(CumProbability(Pred_Util - Start) -
    CumProbability(Pred_Util - End))/2
end
else if (Pred_Util <=End)AND(Pred_Util>=Start)
then
    prob1=CumProbability(Pred_Util - Start)/2
    prob2=CumProbability(End - Pred_Util)/2
    prob=prob1+prob2
end
else
    prob=(CumProbability(End - Pred_Util) -
    CumProbability(Start - Pred_Util))/2
end
return prob
```

the probability that the future CPU usage value will fall inside the level interval is calculated. Since the probability density function of the absolute value of the prediction error is used, the probability that the prediction error is less than a certain value but on the other side of the predicted CPU usage should be excluded from the calculations. Therefore, the estimated probability should be divided by two.

The expected utility of each time interval into the future is given by the sum, over all levels, of the product of the level utility value (the level CPU usage) with the corresponding probability of getting that CPU usage value (level probability). The expected utility of a future host overload state is given as the sum of the expected utilities of 4 consecutive time intervals into the future starting from time interval 4. The host is declared as overloaded and therefore a VM live migration action should be taken if the expected utility (which is the expected host SLA violation penalty) of the future overload state is greater than the live migration SLA violation penalty.

One point that should be stressed is that the above decision is based on the short-term optimization of the utility value but does not consider the long-term utility value accumulation that can result from utility values of overload states that are less than live migration SLA violation penalties but over time can accumulate to bigger values. To address this issue, the utility values of overload states that are less than the live migration SLA violation penalty are accumulated, and at each time interval, a check is made. If the accumulated utility value is greater than the live migration SLA violation penalty, than a VM live migration action is performed regardless whether there is no overload state at that time interval. The same modification is also added to the probabilistic detection approaches explained in the "Probabilistic overload detection", "Probabilistic not-overload detection" and "Probabilistic underload detection" sections.

#### Decision-theoretic not-overload detection

To detect whether the destination host is not overloaded after a possible VM live migration, a check is made whether the expected utility of 4 consecutive future time intervals starting from interval 4 is greater than zero. If it is zero, then the host will be not overloaded after a VM live migration and can serve as a destination of the VM.

#### Decision-theoretic underload detection

To detect whether a host is underloaded, the same approach of probabilistic underload detection that is presented in "Probabilistic underload detection" section is used. The utility value is not used for underload detection, since it represents a host SLA violation that can happen only when the host is in the overload state.

### Global agent

The global agent makes global resource allocation decisions by live migrating VMs from overloaded or underloaded hosts to other hosts to reduce SLA violations and energy consumption. It gets notifications from the host agent if a host will be overloaded or underloaded in the future and performs the appropriate VM live migration action if it is worth the cost.

The global agent applies the general resource allocation algorithm used in previous work [8] for global VM resource allocation and the Power Aware Best Fit Decreasing (PABFD) [8] algorithm for VM placement, with the following modifications. For overload or underload detection, our approaches presented above to apply long-term prediction with uncertainty consideration are used. For VM selection, the Minimum Migration Time (MMT) [8] policy is used, but with the modification that only one VM is selected for migration in each decision round even if the host can possibly remain overloaded after migration. This is done to reduce the number of simultaneous VM live migrations and the associated overhead. For the consolidation process, unlike the previous work [8] that considers all hosts excluding overloaded and turned off hosts, we consider only underloaded hosts that are detected by the proposed approaches based on long-term prediction. From the list of underloaded hosts, the ones that have lower average CPU usage of previous history values are considered first. As VM live migration destinations, the hosts detected as not overloaded by the presented approaches with long-term predictions are chosen.

### VM SLA violation

Since it is difficult for the cloud provider to measure a performance violation metric outside VMs that depends on

the performance metric of various applications, such as response time or throughput, we defined a general SLA violation metric that can be easily measured outside VMs. It is based only on VM resource usage. More specifically, it is called VM SLA violation and represents the penalty of the cloud provider for violating the performance of the VMs of the cloud consumer. The performance of an application running inside a VM is at an acceptable level if the required VM resource usage is less than the resource share allocated.

Following the previous argument, a VM SLA violation is defined to happen if the difference between the allocated CPU share and CPU usage of a VM is less than 5% of the CPU capacity for 4 consecutive time intervals. For example, if the CPU share allocated to a VM is 35% of the CPU capacity and the actual CPU usage is more than 30% for 4 consecutive time intervals, then there is a VM SLA violation. The idea of this definition is that application performance gets worse as the required CPU usage is near to the allocated CPU share. The penalty of a VM SLA violation is the CPU share by which the actual CPU usage exceeds the 5% threshold difference from the allocated CPU, for all 4 consecutive time intervals. Although the SLA violation penalty is defined in terms of CPU usage, it can be easily converted to a monetary value by some conversion function. Thus, one of the goals of the global agent is to mitigate VM SLA violations by providing sufficient free CPU capacity through VM live migration, in order to have the CPU share allocation above the required usage by more than 5% for each VM.

Having defined the VM SLA violation metric, the overload threshold can be defined as follows. It is calculated dynamically based on the number of VMs. Let us define $N$ as the number of VMs on a host. To avoid a VM SLA violation, each VM should have more than 5% capacity above CPU usage, so the total free CPU capacity of the host should be more than $N * 5\%$. Based on this, the overload threshold is calculated as the total CPU capacity (100%) minus $N*5\%$. This means that the overload threshold represents the CPU usage level above which some VMs will have SLA violations.

## Experimental evaluation

In this section, an experimental evaluation of the proposed approach is presented. First, the experimental setup is described. Then, the experimental results are discussed.

### Experimental setup

To conduct controllable and repeatable experiments in a large cloud infrastructure, the CloudSim [9] simulator is used. It is a well known simulator that permits the simulation of dynamic VM resource allocation and energy consumption in virtualized environments. We have made several modifications and extensions to CloudSim in order to integrate the proposed approach and to provide support for setting the CPU CAP to VMs for local resource allocation.

A virtualized data center with 100 heterogeneous hosts is simulated in our experiments. Two types of hosts are simulated, each with 2 CPU cores. One host has CPU cores with 2,100 MIPS and the other one has CPU cores with 2,000 MIPS, while both have 8 GB of RAM. One host simulates the power model of the HpProLiantMl110G4 Xeon3040 computer, and the other one simulates the power model of the HpProLiantMl110G5 Xeon3075.

In the beginning of the simulation, on each host, on the average, 3 VMs (leading to a total of 300 VMs) are scheduled. Four types of VMs are used, and each VM requires one VCPU. Three VMs require a maximum VCPU capacity of 1000 MIPS, while the other one requires 500 MIPS. Two VMs require 1740 MB of RAM, one requires 870 MB, and the last one requires 613 MB. To test realistic workloads, the CPU usage data of real VMs running on the PlanetLab [10] infrastructure are chosen to simulate VM workloads. Each VM runs one application (cloudlet in CloudSim terminology) and the cloudlet length, given as the total number of instructions, is set to a large value in order to prohibit cloudlets to finish before the experiment ends. The experiment is run for 116 time intervals, and the duration of a time interval is set to 10 seconds. The overload detection approach that does not apply prediction error probability modelling yields the same results every time it is run with the same workload, while the probabilistic approaches lead to different results. This prevents running repeated experiments with the same workload in order to compare the two approaches in a fair manner. For this reason, a small random value from a normal distribution with 0 mean and 0.001 standard deviation is added to the CPU usage value of the PlanetLab workload for each time interval. This adds enough perturbation for the experiment to give different results for different runs, as required.

For long-term time series prediction, the WEKA [14] machine learning framework with Gaussian Processes for regression is used through its Java API. A history of previous CPU usage data with a length of 20 samples is used for prediction and forecasting model training. To keep the simulation time to acceptable levels, the forecasting model is trained every 5 time intervals with new CPU usage data. For kernel density estimation, the empirical probability distribution implementing the Variable Kernel Method with Gaussian Smoothing of the Apache Commons Math 3.6 API [15] is used. A history of previous prediction errors with a length of 30 samples is used for probability density function model training, which is done in each time interval.

**Experimental results**

In this section, experimental results of comparing six different approaches are presented. The first one called NO-Migrations (NOM) is the approach that just allocates CPU resources locally to VMs, but does not perform live migration actions. The second one called Short-Term Detection (SHT-D) detects an overload state if the actual and the predicted CPU usage values of the next two time intervals in the future are above the overload threshold. Also, to detect not-overload and underload states, the actual and predicted CPU values for the next two time intervals into the future are used. This approach represents detection based on short-term CPU usage predictions and is expected to be quite sensitive to short spikes of overload conditions. The third approach called Long-Term Detection (LT-D) bases overload, underload and not-overload detections on long term CPU usage predictions of the next 7 control intervals into the future. The fourth approach called Long-Term Probabilistic Detection (LT-PD) bases overload, underload and not-overload detections on long-term CPU usage predictions of the next 7 control intervals into the future, but takes into account prediction uncertainty through prediction error probability distribution modelling. The next approach called Long-Term Decision Theory Detection (LT-DTD) bases overload, underload and not-overload detections on long-term CPU usage predictions of the next 7 control intervals into the future, but takes into account prediction uncertainty and live migration overhead by applying decision theory. The last approach called Local Regression Detection (LR-D) is one of the approaches used in related work [8] that uses the local regression technique to predict the resource usage in the future. We selected it as a representative state-of-the-art technique, since it achieves the best performance as shown by the authors [8] compared to other techniques that use static or adaptive utilization thresholds.

In our evaluation, the following performance metrics are used:

- VM SLA violation (VSV) as explained in "VM SLA violation" section represents the penalty of the cloud provider for each VM. It is important to stress that a VM SLA violation can also happen because of wrong local CPU share allocations as a result of wrong CPU predictions. In the experiments, only a VM SLA violation, as a result of shortage of CPU capacity due to overload states of hosts, is shown.
- Energy consumption (E) of the data center for the whole experimental time measured in KWh.
- Number of VM live migrations (NM) for the whole experimental time.
- Since there is a trade-off between energy consumption and SLA violations, another metric that integrates both VM SLA violations and energy in a single value is defined. This is called the Utility metric and is given by the formula below:

$$Utility = \frac{CVSV}{NOM\_CVSV} + \frac{Energy}{NOM\_Energy} \quad (2)$$

  where *CVSV* is the cumulative VSV value of all VMs for the whole experimental time, *Energy* is the energy consumption, *NOM_CVSV* is the cumulative VSV value of the NOM approach, *NOM_Energy* is the energy consumption of the NOM approach. Both *NOM_CVSV* and *NOM_Energy* are used as reference values for the normalization of the respective metrics. Normalization is performed to permit the integration of two metrics with different measuring units in a single utility function. The best approach is the one that achieves the minimal Utility value.
- Another metric that also has been used in previous work [8] and can capture both energy and VM SLA violations is ESV. This metric is given by:

$$ESV = E * CVSV \quad (3)$$

  where $E$ is energy consumption and *CVSV* is the cumulative VSV value of all VMs for the entire experimental time.

The simulation experiment is run for two different load levels called LOW and HIGH and three different VM live migration SLA violation penalties, mp=2%, mp=4% and mp=6% (MP2, MP4, MP6). For convenience, in the following, the term VM live migration SLA violation penalty is shortened to live migration penalty. By load level we mean the CPU usage consumed by each VM. The loads LOW and HIGH are taken by multiplying the PlanetLab CPU usage values for each time interval with a constant value of 8 and 14, respectively. Each of the previously given live migration penalties represents the migration penalty of one time interval. The experiment is repeated five times for each combination of approach, load level and migration penalty, and the results are exposed to a statistical ANOVA analysis.

In Fig. 2, the cumulative VM SLA violation penalty (cumulative VSV) for each approach averaged over all combinations of load levels and migration penalties is shown. The cumulative VSV value is the sum of VSV values of all VMs for the whole experimental time. It is evident from the graph that the LT-DTD and LT-PD approaches that consider prediction uncertainty achieve lower VM SLA violation levels than the other approaches, with statistical significance, as shown by the ANOVA analysis. On the other hand, we see that LR-D performs better than SHT-D, but similar to LT-D with no statistically significant difference. This is expected, since both techniques apply prediction of resource usage into the future, but without taking prediction uncertainty into

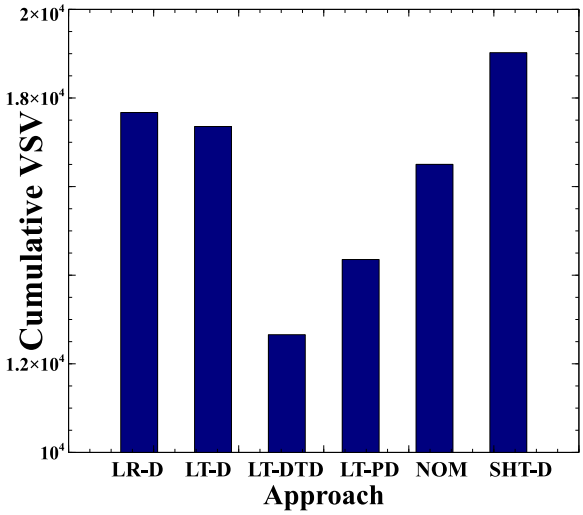


**Fig. 2** Cumulative VSV over all loads and migration penalties

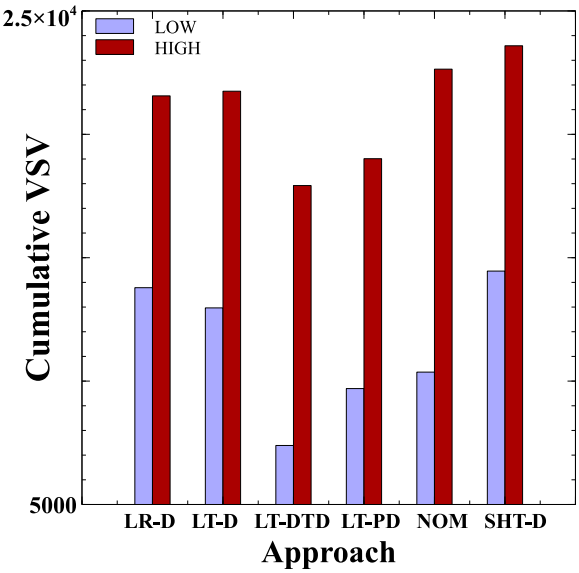


**Fig. 3** Cumulative VSV over all migration penalties for two load levels

account. This shows that considering long-term prediction uncertainty in decision-making is useful for lowering VM SLA violations. More importantly, the LT-DTD approach achieves the lowest level of VM SLA violations compared to the other approaches, confirming the conclusion that applying decision theory to take into account live migration penalty can result in better performance. For example, the LT-DTD approach decreases the cumulative VSV value relative to the LT-D approach by 27% and relative to LT-PD by 12%. Furthermore, by comparing LT-D with SHT-D, applying long-term predictions even without considering prediction uncertainty can lower VM SLA violations.

To see the effect the load level has on VM SLA violations, in Fig. 3 the cumulative VSV value is shown, averaged over all migration penalties, for each approach and the two load levels. First, it can be observed that for each approach, increasing the load increases the VM SLA violations, which is expected since there is more contention for resources. Again, we can see that for both load levels, the LT-DTD approach achieves the lowest VSV value compared to the other approaches. More importantly, the reduction in VSV value by going from high load to low load is larger for the LT-DTD approach than for the other approaches. For example, the reduction in VSV value from high to low load for the LT-D approach is 40%, while for LT-DTD it is 59%. Furthermore, it can be observed that for low load both LR-D and LT-D are worse even compared to the non-migration case, and only for high loads they show better results, with statistical significance, as indicated by the ANOVA tests. This shows that when the load is low, it is not worth, at least with respect to VM SLA violations, to perform live migration actions without taking into account prediction uncertainty.

To understand how different approaches behave regarding the migration penalty, Fig. 4 shows the cumulative VSV value for each approach, averaged over two load levels and three migration penalties. In general, for all approaches, increasing the migration penalty results in increased VM SLA violation values, which is expected since the migration penalty is part of the VM SLA violation value calculation. It is evident that LT-DTD is more robust and does not really follow this trend, as shown by statistically not significant differences of the cumulative VSV values between MP2 and MP4. This is because the

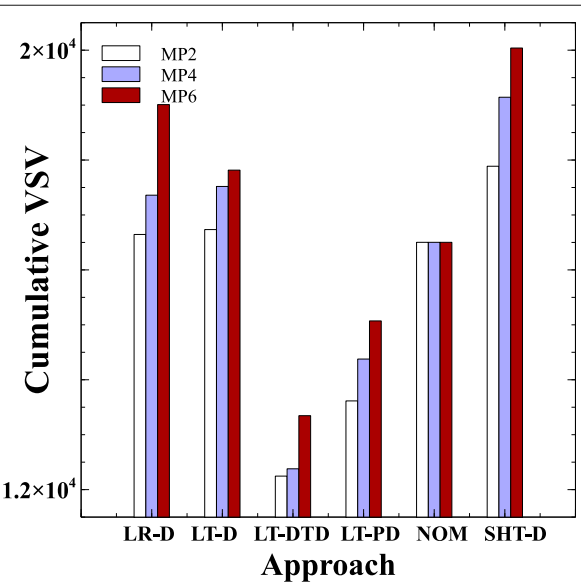


**Fig. 4** Cumulative VSV over all loads for three migration penalties

LT-DTD approach takes into account migration penalties when making decisions.

In Fig. 5, we show the total number of VM live migrations for each approach, averaged over all combinations of load levels and migration penalties. It can be observed that the LT-DTD approach achieves the smallest number of live migrations with a reduction of 46 and 29% compared to LT-D and LT-PD, respectively. First, these results show that by moving from short-term prediction to long-term prediction increases the stability of the approach, reducing the number of live migrations. More importantly, taking into account uncertainty of long-term predictions and live migration penalties increases stability and reduces the number of live migrations further. Interestingly, it can be observed that the LR-D approach has the highest number of VM live migrations compared to the other approaches. This can be explained by the fact that the LR-D approach takes live migration actions if only one predicted usage point in the future is above the threshold, while the other approaches check several points into the future.

In Fig. 6, we show for each approach how the number of live migrations is affected by the load level. In general, apart from LT-DTD and LT-D, increasing the load level increases the number of live migrations, which can be explained by the fact that more live migrations are required to deal with increased load. The LT-DTD approach shows more stability by not increasing the number of live migrations with increased load, and since this still results in better VSV values compared to the other approaches (as shown in Fig. 3), this is a desirable behavior. The LT-D approach shows a slight increase in the number of live migrations, but this is not statistically significant, as shown by an ANOVA analysis.

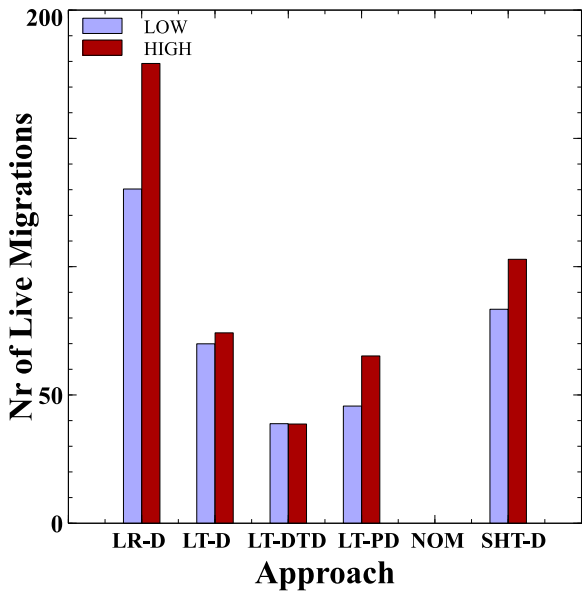


**Fig. 6** Number of live migrations over all migration penalties for two load levels

Figure 7 shows for each approach how the number of live migrations changes by varying the migration penalty. Unlike other approaches, both LT-DTD and LT-PD show decreased numbers of live migrations by increasing the migration penalty. The LT-DTD approach shows a decreased number of live migrations when moving from MP2 to MP4, and this can be explained by the fact that

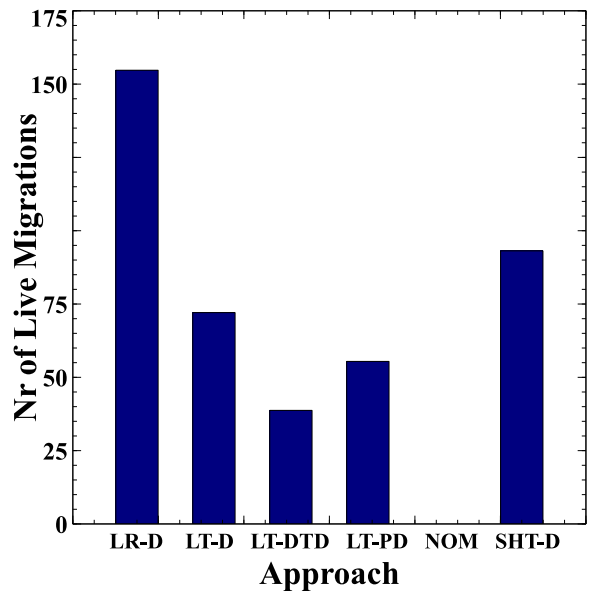


**Fig. 5** Number of live migrations over all loads and migration penalties

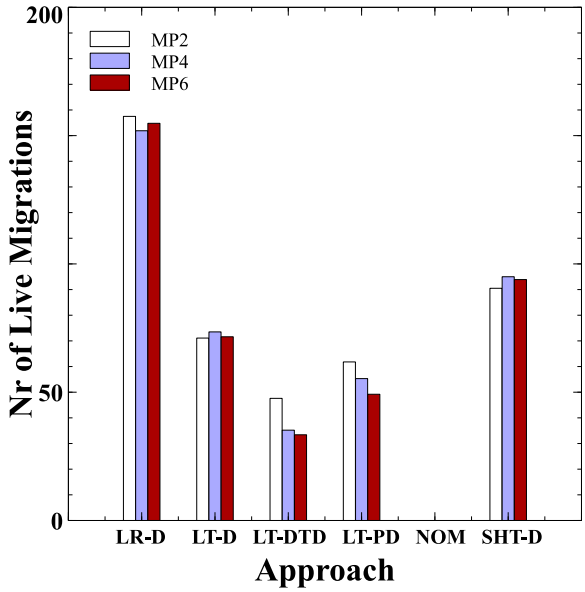


**Fig. 7** Number of live migrations over all loads for three migration penalties

it takes the migration penalty into account when making live migration decisions. This behavior has the benefit of using the migration penalty as a parameter to control the aggressiveness of the consolidation process. On the other hand, the decreased number of live migrations for the LT-DP approach, at least for MP2 to MP6, which is statistically significant, does not have an apparent explanation since this approach does not take into account the migration penalty in decision making. The only explanation is that this behavior is caused because the LT-PD approach also takes into account utility value accumulation in decision making the same way as LT-DTD does, as explained in "Decision-theoretic overload detection" section . To test this claim and to check whether also the LT-DTD approach achieves the above behavior due to this this modification, an experiment has been conducted to measure the number of live migrations for three migration penalties with low load. The experiment is run for LT-PD and LT-DTD, but without taking into account utility value accumulation.

The results of the experiment are shown in Fig. 8. It is evident that for LT-PD, increasing the migration penalty does not change the number of live migrations, supporting the claim that the above behavior is caused only by taking into account utility value accumulation in decision making. However, for LT-DTD, increasing the migration penalty decreases the number of live migrations, showing that this behavior is due to the decision-theoretic approach adopted by it.

Figure 9 shows the energy consumption of the data center for the whole experimental time for each approach

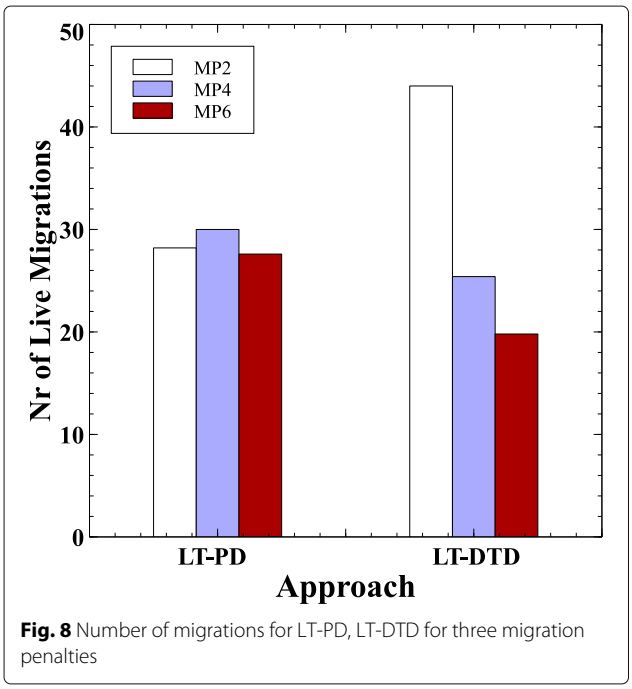


**Fig. 8** Number of migrations for LT-PD, LT-DTD for three migration penalties

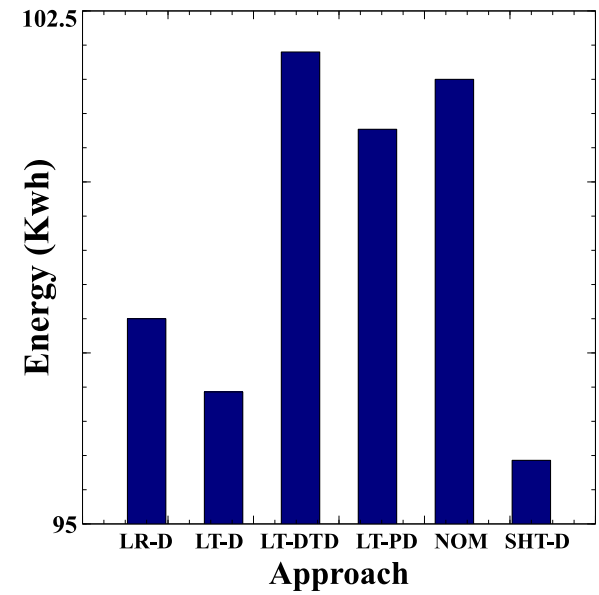


**Fig. 9** Energy over all load levels and migration penalties

averaged over all combinations of load levels and migration penalties. It is evident that the LT-DTD approach shows a slight increase in energy consumption compared to the other approaches. For example, it increases energy consumption by 5 and 0.30% compared to LT-D and NOM, respectively. Although the LT-DTD approach saves less energy, the improvement in the VM SLA violation values outweighs the decrease in energy savings, as shown by the results of the Utility metric. The LR-D, LT-D and SHT-D aproaches achieve more energy savings than the LT-DTD approach at the expense of higher VM SLA violations, resulting in worse ESV and Utility values.

In Fig. 10, we show for each approach how the energy consumption is affected by the load level. It can be observed, as expected, that increasing the load increases the energy consumption for all approaches. Decreased energy consumption with a decrease in the load level can be explained by the fact that low load creates more opportunities for consolidation and turning off hosts.

From the above argument it can be expected that by decreasing the load level further, LT-DTD can save energy compared to NOM. To test this claim, another experiment is conducted with load lower than LOW load, which is called Very LOW (VLOW). VLOW is taken by multiplying the PlanetLab CPU usage values for each time interval by a constant value of 2. The migration penalty is set to MP4. The experiment is repeated for 5 times for each of the LT-DTD and NOM approaches.

The average energy consumption and Utility values are shown in Table 1. The Utility value is shown to understand if any possible energy savings are achieved at the expense of VM performance.

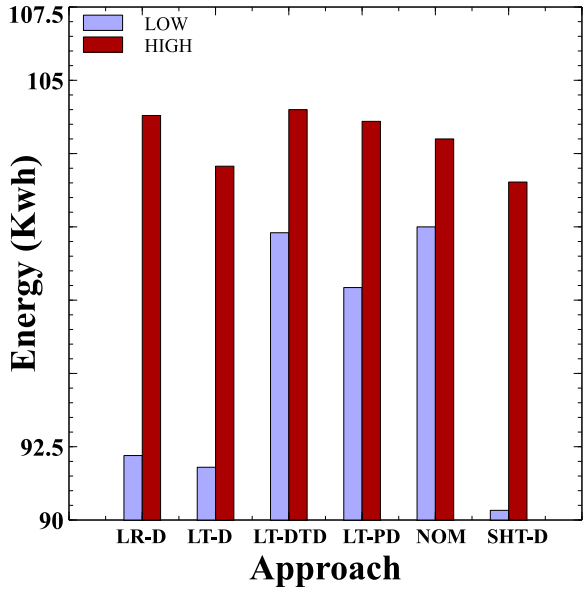


**Fig. 10** Energy over all migration penalties for two load levels

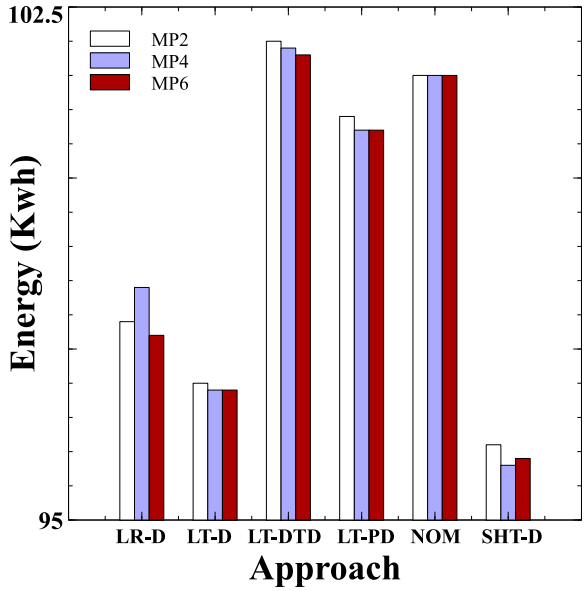


**Fig. 11** Energy over all loads for three migration penalties

The LT-DTD approach achieves energy savings of 22.8% compared to NOM, with a better Utility value. This shows that when the load decreases, the LT-DTD approach gives more priority to the consolidation process, reducing the number of hosts and saving energy. On the other hand, when the load increases, it gives more priority to the load balancing process by saving less energy but lowering the VM SLA violations.

In Fig. 11, we show for each approach how the energy consumption is affected by migration penalty. An ANOVA statistical analysis indicates, for each approach, no statistically significant differences of energy value between different migration penalty cases. This shows that the migration penalty has no significant effect on energy consumption.

To understand better the trade-off between energy savings and performance of VMs, in Fig. 12 we present the Utility for each approach over all load levels and migration penalties. The Utility is the metric that indicates improvements in both energy savings and VM SLA violations and can serve as the metric of measuring the overall performance. It can be observed that the LT-DTD approach achieves the lowest Utility value compared to other approaches with statistical significance, as shown by an ANOVA analysis. It improves the Utility by approximately 9.4% and 4.3% compared to LT-D and LT-PD approaches, respectively. These results show that although the LT-DTD approach achieves slightly less energy savings, it improves the SLA violations, thus finding the best performance-energy trade-off. With respect to the Utility value, the LR-D approach performs better than the SHT-D approach, but slightly worse than the LT-D approach. This is because the LR-D approach achieves

**Table 1** Energy and utility for two approaches with MP4 penalty and VLOW load

| Approach | Energy (KWh) | Utility value |
|---|---|---|
| *LT-DTD* | 70.2 | 0.79 |
| *NOM* | 91 | 0.88 |

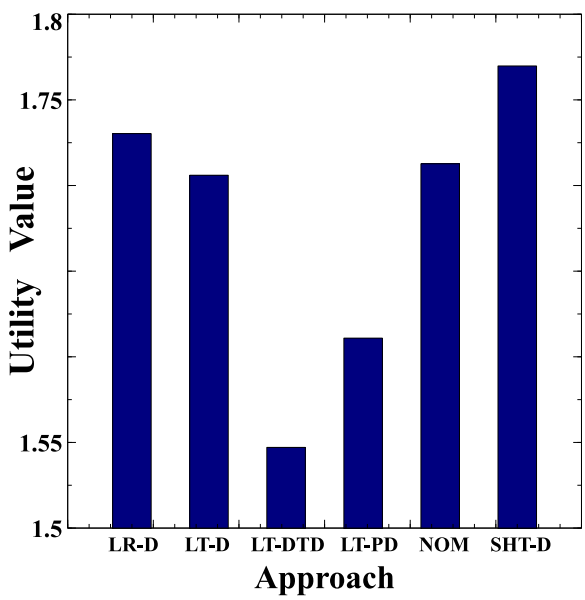


**Fig. 12** Utility value over all loads and migration penalties

less energy savings than the LT-D approach with similar SLA violations, as shown in Figures 2 and 9.

Figure 13 shows for each approach how the Utility is affected by the load levels. It can be observed that increasing the load increases the Utility value, since both energy consumption and SLA VM violations are increased. For each load level, the LT-DTD approach achieves the lowest Utility value compared to the other approaches. Similar to the case of the cumulative VSV value, it is also evident that for high load, the LR-D and LT-D approaches achieve comparably equal Utility values and show improvements compared to the NOM case with statistical significance.

Figure 14 shows for each approach the effect that the migration penalty has on the Utility. In general, for all approaches it can be observed that increasing the migration penalty increases the Utility, since it increases the VM SLA violation penalty. However, similarly to the VSV value, the LT-DTD approach seems to be more resistant in increasing the Utility. This can be observed at least for the case of moving from MP2 to MP4 where there are no statistically significant differences resulting from the ANOVA analysis.

In Figs. 15, 16 and 17, the overall ESV value, the ESV value for two load levels and the ESV value for three migration penalties, are shown, respectively. For display convenience, the ESV value in the graphical illustration is divided by 10.000. It can be observed that the ESV value shows the same trend as the Utility value. The LT-DTD approach achieves the lowest ESV value compared to the other approaches with statistical significance, as shown by the ANOVA analysis. Similarly to the Utility value, the LR-D approach performs comparably equal with LT-D and

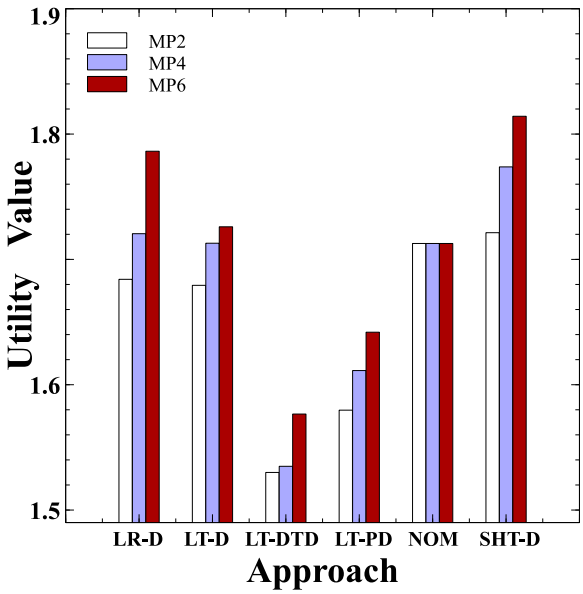


**Fig. 14** Utility value over all loads for three migration penalties

better than NOM and SHT-D with statistical significance, especially for high loads. Furthermore, regarding the ESV, the LT-DTD approach seems to be more resistant with respect to increasing the ESV value with an increased live migration penalty.

## Related work

There are many works in the literature on dynamic resource allocation in cloud computing, tackling the

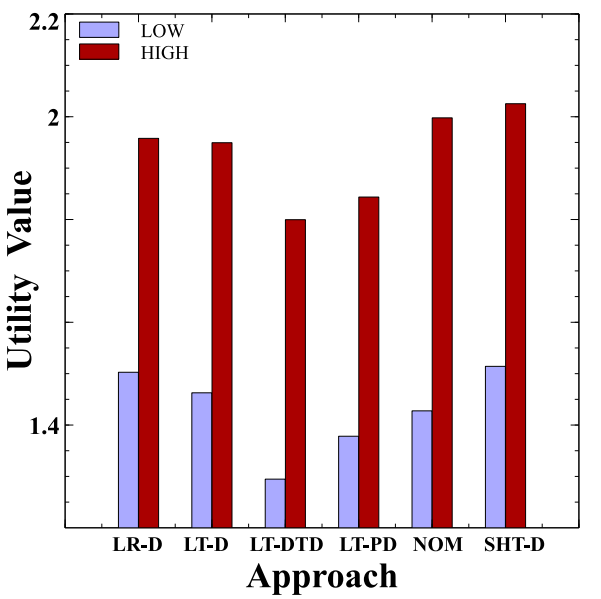


**Fig. 13** Utility value over all migration penalties for two load levels

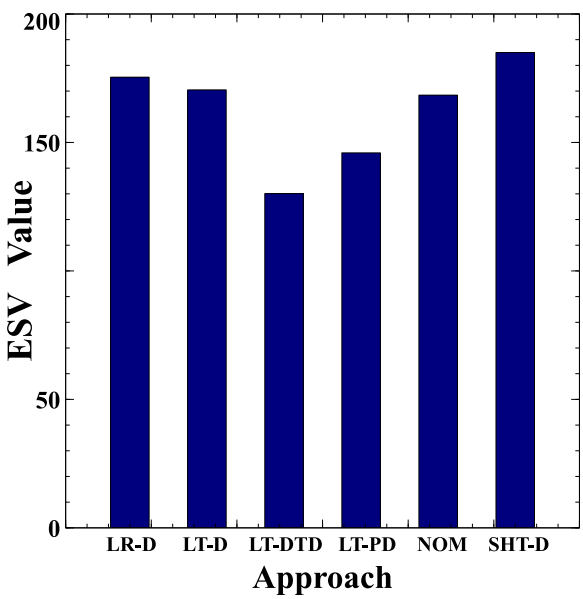


**Fig. 15** ESV value over all loads and migration penalties

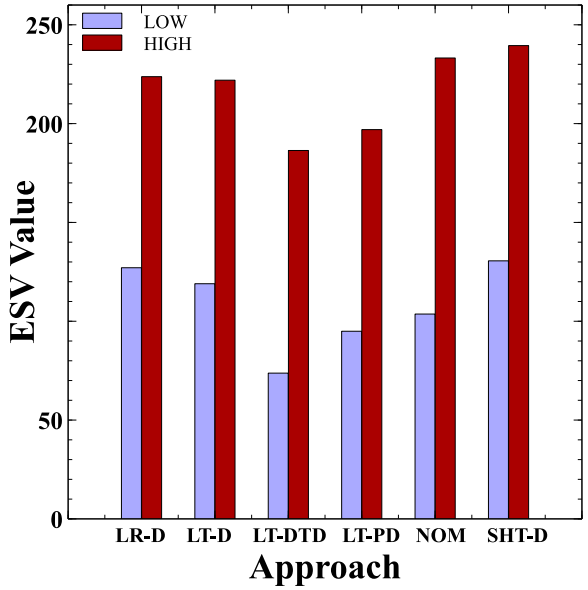


**Fig. 16** ESV value over all migration penalties for two load levels

problem from different angles. Therefore, we cannot provide an exhaustive treatment of related work, but focus mainly on the aspects of VM resource demand prediction and host overload detection.

Several works apply VM live migration to allocate resources to VMs for overload mitigation or consolidation of VMs to fewer hosts. For example, Wood et al. [16] propose an approach called *Sandpiper* for overload detection and live migration of VMs from overloaded hosts to not overloaded ones. Their overload detection approach declares a host as overloaded if the past k resource usage values and the next predicted one exceed a given threshold. They use a greedy algorithm that live migrates heavy loaded VMs to least loaded hosts.

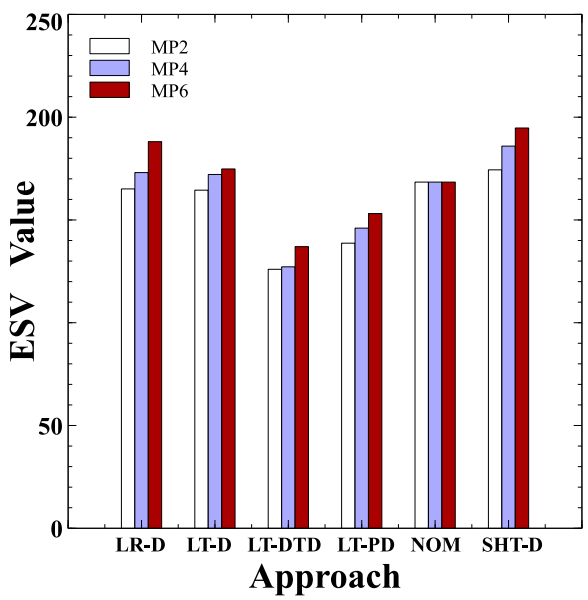


**Fig. 17** ESV value over all loads for three migration penalties

Similarly, Khanna et al. [17] propose an approach for dynamic consolidation of VMs based on live migration. Their approach for host overload detection is also based on resource usage exceeding a threshold value. Their goal is to minimize the number of hosts by maximizing the variance of resource capacity residuals. This is achieved by ordering VMs in non-decreasing order of their resource usage and migrating the least loaded VM to the least residual resource capacity host.

Beloglazov et al. [18] propose energy-aware heuristic algorithms for dynamic allocation of VMs to hosts based on live migration. They decide on the overload or underload state of a host based on whether the CPU usage is higher or lower than the overload or underload thresholds, respectively. The authors apply a modified Best-Fit-Decreasing (BFD) heuristic to pack VMs to fewer hosts, which takes into account the power increase of hosts.

All the above approaches base host overload or underload detection on current or short-term predictions of resource usage and static usage thresholds, which can be sensitive to short spikes of load that can cause stability problems and unnecessary live migrations. In contrast, our approach bases overload or underload detection on long-term predictions of CPU usage by taking into account prediction uncertainty, which results in stability and efficient live migration actions, as shown by the experimental results.

Several other works apply more sophisticated approaches than just static usage thresholds. For example, Beloglazov and Buyya [8], as a continuation of their previous work [18], present different heuristics for host overload and underload detection based on statistical analysis of historical resource usage data. They propose to use adaptive usage thresholds based on statistical parameters of previous data, such as CPU usage Median Absolute Deviation (MAD) or interquartile range (IQR). The authors also apply local regression methods for predicting CPU usage value some time ahead into the future. Our approach also applies CPU usage prediction, but additionally considers prediction uncertainty and live migration penalties in decision making.

Ferreto et al. [19] present an approach called *dynamic consolidation with migration control* in which they formulate the consolidation problem as a linear programming problem with constraints that prohibits migrating VMs with steady workload. As the authors show, this results in lowering the number of VM migrations with a small increase in the number of hosts. Their work is complementary to our work, since it tries to avoid unnecessary

migrations, but can not provide stability if the data center is running only variable load VMs.

Gong and Gu [20] propose a dynamic consolidation approach called Pattern-driven Application Consolidation (PAC) based on extracting patterns of resource usage called signatures using signal processing approaches such as Fast Fourier Transform (FFT) and dynamic time warping (DTW). Based on extracted signatures, they perform dynamic placements of VMs to the hosts that have the highest match between VM resource usage signature and host free capacity signature. Their work focuses on periodic global consolidation for VM resource usage patterns that show periodicity. The authors also consider on-demand VM migrations for instantaneous overloads, but in contrast to our approach, they base overload detection on a single resource usage value exceeding a static threshold.

Andreolini et al. [21] propose an approach for host overload detection in which a host is declared as overloaded when there is a substantial change in the load trend of the host, as a result of applying the CUSUM algorithm. Their goal is similar to the goal of our work for providing a robust and stable approach avoiding unnecessary live migrations, but their load change point detection requires past history usage data to be available, at which point the SLA violations have already happened. In contrast, our approach applies long-term prediction to avoid violations before they happen.

Beloglazov and Buyya [22] propose an approach for host overload detection based on Markov chains and optimization of inter-migration time with Quality of Service (QoS) constraints. The goal of their approach is finding the solution (migration probabilities of each state) of an optimization problem to maximize inter-migration time while keeping the Overload Time Fraction (OTF) metric inside certain values. To take into account dynamic and non-stationary workloads, the authors apply a multi-size sliding window approach. Similarly, we also propose an approach for host overload detection, but in contrast, we apply long-term prediction techniques taking into account the VM live migration penalty. Another difference is that we tackle a different performance metric, i.e., minimization of SLA violations of each VM, while Beloglazov and Buyya focus on keeping the percentage of time that a host is overload inside certain constraints.

There are several works that apply VM resource demand prediction techniques for resource allocation in cloud computing. For example, Gong et al. [23] and later Shen et al. [11] propose an approach for VM fine-grained resource allocation based on resource demand prediction. They base their resource demand prediction on two methods: a) Fast Fourier Transform to find periodicity or signature of resource demand and b) a state based approach using Markov chains. Similarly to our approach for overload detection, they apply these methods for long-term prediction of host resource conflicts. If they predict a conflict, they apply a live migration action to resolve it, taking into account the migration penalty. As the authors point out, using a multi-step Markov model to predict further into the future lowers the prediction accuracy. This is exactly the problem we tackle in this paper by taking into account uncertainty of long-term prediction to deal with low prediction accuracy.

Islam et al. [24] propose resource prediction approaches based on machine learning. More specifically, they propose and experiment with Linear Regression and an Error Correction Neural Network. They show experimentally the superiority of the neural network in making more accurate predictions, but they do not apply their techniques to host overload detection or in general for VM resource allocation.

Farahnakian et al. [25] propose a prediction technique based on linear regression to detect if a host is overloaded or underloaded. They train a model based on past CPU utilization history and predict the next CPU utilization. Based on this prediction, they detect if a host is overloaded or underloaded and apply VM live migration to move VMs to other hosts. The problem with their approach is that they base their overload or underload detection technique on short-term CPU utilization prediction which is susceptible to oscillatory load. In contrast, we apply long-term prediction augmented with uncertainty estimation to provide a more stable approach.

Khatua et al. [26] propose an approach for VM load prediction several time steps into the future by applying an Auto-regressive Integrated Moving Average (ARIMA) model. They apply their approach for horizontal scaling in cloud settings. If an overload situation is detected, based on some threshold value, then the number of VMs is increased. Also, their approach does not consider the uncertainty and prediction errors in their model of long-term prediction, which is important for increasing the quality of allocation decisions.

Ashraf et al. [27] propose a load prediction approach for VM resource allocation and admission control of multi-tier web applications in cloud computing. Their prediction method is based on a two step procedure. In the first step, a so called load tracker, based on Exponential Moving Average (EMA), constructs a representative view of the load by filtering the noise. In the second step, a load predictor based on linear regression takes as input the representative view of the load produced by load tracker and provides as output the predicted load value in some interval $k$ in the future. They apply a hybrid reactive-proactive approach to calculate a weighted utilization. Through a linear interpolation, the authors mix the measured and the predicted value, by including a weight factor

*w* that depends on the prediction error. In contrast to their work, our approach to prediction is different. We apply a long-term prediction method directly to the past resource utilizations and consider long-term prediction uncertainty through prediction error probability distribution. Furthermore, their approach is applied for horizontal VM scaling and admission control of multi-tier web applications, while we tackle the problem of host overload detection and mitigation through VM live migration.

Qiu et al. [28] propose an approach for VM load prediction based on a deep learning prediction model. More specifically, this model is composed of two layers, the Deep Belief Network (DBN) and a regression layer. The DBN is used to extract the high-level workload features from the past VM resource utilizations, while the regression layer is used to predict the future load values. The authors evaluate experimentally only the prediction accuracy of the approach, but do not apply it on any VM resource allocation problem. In contrast, we propose and evaluate a complete approach for VM resource allocation problem through long-term resource prediction and VM live migration.

## Conclusion

In this paper, a novel approach for VM resource allocation in a cloud computing environment has been presented. It allocates resources locally by changing the CPU share given to VMs according to the current load. Global resource allocation is performed by migrating VMs from overloaded or underloaded hosts to other hosts to reduce VM SLA violations and energy consumption. For overload or underload host detection, long-term predictions of resource usage are made, based on Gaussian processes as a machine learning approach for time series forecasting. To take into account the prediction uncertainty, a probability distribution model of the prediction error is constructed using the kernel density estimation method. To consider the VM live migration overhead, a decision-theoretic approach is applied.

We can draw the following conclusions. First, making long-term predictions of resource demand can increase stability and overall performance of a cloud. Second, making overload detection decisions proportional to uncertainty of predictions is the right thing to do, as supported by our experimental results. Third, taking into account both prediction uncertainty and live migration overhead by applying decision-theoretic optimization methods yields the best decisions and improves the performance further.

There are several areas for future work. First, we want to point out that our approach is based on a long-term prediction model that relies on historical load patterns. This means that our prediction model cannot easily predict sudden and sharp increases of the load (i.e., load bursts). This issue is out of scope of this paper, but it can be addressed by focusing on load burst detection techniques ([29–31]). Thus, an interesting area of future work is combining load burst detection techniques with load prediction techniques to deal with a large variety of cloud load patterns. Second, in addition to the currently used scheme of predicting the next CPU usage value for local resource allocation, more sophisticated schemes based on control theory [32, 33], Kalman filters [34] or fuzzy logic [35, 36] can be explored. Third, a distributed resource allocation approach should be investigated, where each host agent makes live migration decisions in cooperation with nearby host agents. A distributed approach is suitable for large scale cloud infrastructures where centralized optimization complexity and single point of failure are important factors to consider. In distributed approaches, the problem is how local agents with a limited view should coordinate each other to achieve a global optimization objective. Finally, investigating long-term prediction of the usage of multiple resources (e.g., CPU, memory and I/O bandwidth) and their interdependencies in allocation decisions is an interesting area of future work.

#### Abbreviations

ARIMA: Auto-regressive integrated moving average; BFD: Best-fit-decreasing; DBN: Deep belief network; DTW: Dynamic time warping; EMA: Exponential moving average; FFT: Fast fourier transform; IaaS: Infrastructure as a service; IQR: Interquartile range; LT-D: Long-term detection; LT-DTD: Long-term decision theory detection; LT-PD: Long-term probabilistic detection; MAD: Median absolute deviation; MMT: Minimum migration time; NOM: No-migrations; OTF: Overload time fraction; PAC: Pattern-driven application consolidation; PABFD: Power aware best fit decreasing; SHT-D: Short-term detection; SLA: Service level agreement; VM: Virtual machine; VSV: VM SLA violation

#### Acknowledgements

This work is supported by the German Research Foundation (DFG, SFB 1053 - MAKI), by the LOEWE initiative (Hessen, Germany) within the NICER project and and the Albanian Government (Excellence Fund).

#### Authors' contributions

DM provided the main idea of the paper, designed and coded the main algorithms in the simulator, performed some of the experiments and their statistical analysis and wrote the first draft of the manuscript. AM revised the manuscript and added some of the related works, helped finding bugs and revised the code, adapted the local regression algorithm and performed some of the experiments and statistical analysis. BF revised the algorithms and the experimental statistical data analysis, critically reviewed and edited the manuscript. All authors read and approved the final manuscript.

#### Competing interests

The authors declare that they have no competing interests.

#### Author details

[1]Department of Computer Engineering, Polytechnic University of Tirana, Tirana, Albania. [2]Department of Contemporary Sciences and Technologies, South East European University, Tetovo, Macedonia. [3]Department of Mathematics & Computer Science, Philipps-Universität Marburg, Marburg, Germany. [4]Department of Electrical Engineering & Information Techology, TU Darmstadt, Darmstadt, Germany.

Received: 20 October 2016 Accepted: 19 January 2017
Published online: 06 February 2017

**References**


1. Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state-of-the-art and research challenges. J Intern Serv Appl 1(1):7–18
2. (2011) Amazon elastic compute cloud (amazon ec2). http://www.vmware.com/. Accessed 5 Jul 2016
3. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. In: Proc. 19th ACM Symposium on Operating Systems Principles. ACM, New York. pp 164–177
4. (2016) VMWare, Inc. http://www.vmware.com/. Accessed 7 Jul 2016
5. Clark C, Fraser K, Hand S, Hansen JG, Jul E, Limpach C, Pratt I, Warfield A (2005) Live migration of virtual machines. In: Proc. 2nd Conference on Symposium on Networked Systems Design and Implementation. USENIX Association, Berkeley. pp 273–286
6. Rasmussen CE, Williams CKI (2005) Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press, Cambridge
7. Scott DW (1992) Multivariate density estimation : theory, practice, and visualization. In: Wiley series in probability and mathematical statistics : Applied probability and statistics section. Wiley-Interscience, New York, Chichester, Brisbane
8. Beloglazov A, Buyya R (2012) Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr Comput Pract Experience 24(13):1397–1420. doi:10.1002/cpe.1867
9. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R (2011) Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice & Experience 41(1):23–50
10. Park K, Pai VS (2006) Comon: A mostly-scalable monitoring system for planetlab. ACM SIGOPS Oper Syst Rev 40(1):65–74
11. Shen Z, Subbiah S, Gu X, Wilkes J (2011) Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In: Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM, New York, NY. pp 1–14
12. Minarolli D, Freisleben B (2014) Cross-correlation prediction of resource demand for virtual machine resource allocation in clouds. In: Proc. 6th International Conference on Computational Intelligence, Communication Systems and Networks (CICSYN '14). IEEE Computer Society, Washington. pp 119–124
13. Akoush S, Sohan R, Rice A, Moore AW, Hopper A (2010) Predicting the performance of virtual machine migration. In: Proceedings of the 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. IEEE Computer Society, Washington. pp 37–46
14. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: An update. ACM SIGKDD Explor Newsl 11(1):10–18
15. Commons Math: The Apache Commons Mathematics Library (2016). http://commons.apache.org/. Accessed 10 Jul 2016
16. Wood T, Shenoy P, Venkataramani A, Yousif M (2009) Sandpiper: Black-box and gray-box resource management for virtual machines. Comput Netw 53(17):2923–2938
17. Khanna G, Beaty K, Kar G, Kochut A (2006) Application performance management in virtualized server environments. In: Proc. 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006). IEEE Computer Society, Washington. pp 373–381
18. Beloglazov A, Abawajy JH, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Comp Syst 28(5):755–768
19. Ferreto TC, Netto MAS, Calheiros RN, Rose CAFD (2011) Server consolidation with migration control for virtualized data centers. Future Generation Comp Syst 27(8):1027–1034. Elsevier B.V., Amsterdam
20. Gong Z, Gu X (2010) Pac: Pattern-driven application consolidation for efficient cloud computing. In: Proc 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. IEEE Computer Society, Washington. pp 24–33
21. Andreolini M, Casolari S, Colajanni M, Messori M (2009) Dynamic load management of virtual machines in cloud architectures. In: CloudComp, Springer, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 34. Springer, Berlin. pp 201–214
22. Beloglazov A, Buyya R (2013) Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. IEEE Trans Parallel Distrib Syst 24:1366–1379
23. Gong Z, Gu X, Wilkes J (2010) Press: Predictive elastic resource scaling for cloud systems. In: Proc. International Conference on Network and Service Management (CNSM'10). IEEE Computer Society, Washington. pp 9–16
24. Islam S, Keung J, Lee K, Liu A (2012) Empirical prediction models for adaptive resource provisioning in the cloud. Future Gener Comput Syst 28(1):155–162
25. Farahnakian F, Liljeberg P, Plosila J (2013) Lircup: Linear regression based cpu usage prediction algorithm for live migration of virtual machines in data centers. In: Proceedings of the 2013 39th Euromicro Conference on Software Engineering and Advanced Applications. IEEE Computer Society, Washington. pp 357–364
26. Khatua S, Manna MM, Mukherjee N (2014) Prediction-based instant resource provisioning for cloud applications. In: Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing. IEEE Computer Society, Washington. pp 597–602
27. Ashraf A, Byholm B, Porres I (2016) Prediction-based vm provisioning and admission control for multi-tier web applications. J Cloud Comput 5:1–21
28. Qiu F, Zhang B, Guo J (2016) A deep learning approach for vm workload prediction in the cloud. In: 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. IEEE Computer Society, Washington. pp 319–324
29. Lassnig M, Fahringer T, Garonne V, Molfetas A, Branco M (2010) Identification, modelling and prediction of non-periodic bursts in workloads. In: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. IEEE Computer Society, Washington. pp 485–494
30. Mehta A, Dürango J, Tordsson J, Elmroth E (2015) Online spike detection in cloud workloads. In: Proceedings of the 3th IEEE International Conference on Cloud Engineering. IEEE Computer Society, Washington. pp 446–451
31. Zhu Y, Shasha D (2003) Efficient elastic burst detection in data streams. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York. pp 336–345
32. Minarolli D, Freisleben B (2011) Utility-driven allocation of multiple types of resources to virtual machines in clouds. In: Proc. 13th IEEE Conference on Commerce and Enterprise Computing (CEC'11). IEEE Computer Society, Washington. pp 137–144
33. Padala P, Shin KG, Zhu X, Uysal M, Wang Z, Singhal S, Merchant A, Salem K (2007) Adaptive control of virtualized resources in utility computing environments. In: Proc. 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '07). ACM, New York. pp 289–302
34. Kalyvianaki E, Charalambous T, Hand S (2009) Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In: Proceedings of the 6th International Conference on Autonomic Computing. ACM, New York. pp 117–126
35. Minarolli D, Freisleben B (2013) Virtual machine resource allocation in cloud computing via multi-agent fuzzy control. In: Proc. 3rd International Conference on Cloud and Green Computing (CGC'13). IEEE Computer Society, Washington. pp 188–194
36. Rao J, Wei Y, Gong J, Xu CZ (2011) Dynaqos: model-free self-tuning fuzzy control of virtualized resources for qos provisioning. In: Proc. 19th International Workshop on Quality of Service. IEEE Computer Society, Washington. pp 1–9