Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers Under Quality of Service Constraints

Anton Beloglazov and Rajkumar Buyya, *Senior Member, IEEE*

Abstract—Dynamic consolidation of Virtual Machines (VMs) is an effective way to improve the utilization of resources and energy efficiency in Cloud data centers. Determining *when* it is best to reallocate VMs from an overloaded host is an aspect of dynamic VM consolidation that directly influences the resource utilization and Quality of Service (QoS) delivered by the system. The influence on the QoS is explained by the fact that server overloads cause resource shortages and performance degradation of applications. Current solutions to the problem of host overload detection are generally heuristic-based, or rely on statistical analysis of historical data. The limitations of these approaches are that they lead to sub-optimal results and do not allow explicit specification of a *QoS goal*. We propose a novel approach that for any known stationary workload and a given state configuration *optimally* solves the problem of host overload detection by maximizing the mean inter-migration time under the specified QoS goal based on a Markov chain model. We heuristically adapt the algorithm to handle unknown non-stationary workloads using the Multisize Sliding Window workload estimation technique. Through simulations with real-world workload traces from more than a thousand PlanetLab VMs, we show that our approach outperforms the best benchmark algorithm and provides approximately 88% of the performance of the optimal offline algorithm.

Index Terms—Distributed systems, Cloud computing, virtualization, dynamic consolidation, energy efficiency, host overload detection.

1 Introduction

Cloud computing has revolutionized the ICT industry by enabling on-demand provisioning of computing resources based on a pay-as-you-go model. An organization can either outsource its computational needs to the Cloud avoiding high up-front investments in a private computing infrastructure and consequent maintenance costs, or implement a private Cloud data center to improve the resource management and provisioning processes. However, the problem of data centers is high energy consumption, which has risen by 56% from 2005 to 2010, and in 2010 accounted to be between 1.1% and 1.5% of the global electricity use [1]. Apart from high operating costs, this results in substantial carbon dioxide (CO_2) emissions, which are estimated to be 2% of the global emissions [2]. The problem has been partially addressed by improvements in the physical infrastructure of modern data centers. As reported by the Open Compute Project[1], Facebook's Oregon data center achieves a Power Usage Effectiveness (PUE) of 1.08, which means that ≈93% of the data center's energy consumption are consumed by the computing resources. Therefore, now it is important to focus on the resource management aspect, i.e., ensuring that the computing resources are efficiently utilized to serve applications.

• *A. Beloglazov and R. Buyya are with the Department of Computing and Information Systems, the University of Melbourne, Australia. E-mail: abe@csse.unimelb.edu.au, raj@csse.unimelb.edu.au*

1. The Open Compute project. http://opencompute.org/

One method to improve the utilization of data center resources, which has been shown to be efficient [3]–[14], is dynamic consolidation of Virtual Machines (VMs). This approach leverages the dynamic nature of Cloud workloads: the VMs are periodically reallocated using live migration according to their current resource demand in order to minimize the number of active physical servers, referred to as hosts, required to handle the workload. The idle hosts are switched to low-power modes with fast transition times to eliminate the static power and reduce the overall energy consumption. The hosts are reactivated when the resource demand increases. This approach has basically two objectives, namely minimization of energy consumption and maximization of the Quality of Service (QoS) delivered by the system, which form an energy-performance trade-off.

The QoS requirements can be defined in terms of a variety of metrics and are formalized in the Service Level Agreements (SLAs). In this work, to specify the QoS requirements we apply a modification of the workload independent metric proposed in our previous work [15]. Therefore, the problem transforms into minimization of energy consumption under QoS constraints. This problem is too complex to be treated analytically as a whole, as just the VM placement, which is a part of dynamic VM consolidation, is an NP-hard problem [4], [9], [13]. Moreover, many aspects of the problem have to be addressed, e.g., the heterogeneity of physical resources and VMs; non-stationary and unknown workloads, as observed in Infrastructure as a Service (IaaS) environments; power and performance costs of VM migrations; and the large

scale of Cloud data center infrastructures. Another argument for splitting the problem is decentralization of the resource management algorithm, which is important for scaling the resource management system for efficient handling of thousands of servers. Therefore, to make the problem of dynamic VM consolidation tractable and provide decentralization, in our previous work [15] we have proposed its division into 4 sub-problems:

1) Deciding when a host is considered to be overloaded, so that some VMs should be migrated from it to other hosts to meet the QoS requirements.
2) Deciding when a host is considered to be underloaded, so that its VMs should be migrated, and the host should be switched to a low-power mode.
3) Selecting VMs to migrate from an overloaded host.
4) Allocating the VMs selected for migration to other active or re-activated hosts.

In this paper, we focus on the first sub-problem – the problem of host overload detection. Detecting when a host becomes overloaded directly influences the QoS, since if the resource capacity is completely utilized, it is highly likely that the applications are experiencing resource shortage and performance degradation. What makes the problem of host overload detection complex is the necessity to optimize the time-averaged behavior of the system, while handling a variety of heterogeneous workloads placed on a single host. To address this problem, most of the current approaches to dynamic VM consolidation apply either heuristic-based techniques, such as static utilization thresholds [5]–[8]; decision-making based on statistical analysis of historical data [12], [13]; or simply periodic adaptation of the VM allocation [3], [4]. The limitations of these approaches are that they lead to sub-optimal results and do not allow the administrator to explicitly set a *QoS goal*. In other words, the performance in regard to the QoS delivered by the system can only be adjusted indirectly by tuning parameters of the applied host overload detection algorithm. In contrast, our approach enables the system administrator to explicitly specify a QoS goal in terms of a workload independent QoS metric. The underlying analytical model allows a derivation of an optimal randomized control policy for any known stationary workload and a given state configuration. Our **contributions** in this paper are:

1) We analytically show that to improve the quality of VM consolidation, it is necessary to maximize the mean time between VM migrations initiated by the host overload detection algorithm.
2) We propose an *optimal offline algorithm* for host overload detection, and prove its optimality.
3) We introduce a novel Markov Chain model that allows a derivation of a randomized control policy that *optimally* solves the problem of maximizing the mean time between VM migrations under an explicitly specified QoS goal for any known stationary workload and a given state configuration in the *online* setting.
4) To handle unknown non-stationary workloads, we apply the Multisize Sliding Window workload estimation approach [16] to heuristically build an adapted algorithm, which leads to approximately 15% higher mean inter-migration time compared to the best benchmark algorithm for the input workload traces used in our experiments. The adapted algorithm leads to approximately 88% of the mean inter-migration time produced by the optimal offline algorithm.

We evaluate the algorithm by simulations using real-world workload traces from more than a thousand PlanetLab[2] VMs hosted on servers located in more than 500 places around the world. Our experiments show that the introduced algorithm outperforms the benchmark algorithms, while meeting the QoS goal in accordance with the theoretical model. The algorithm uses a workload independent QoS metric and transparently adapts its behavior to various workloads using a machine-learning technique; therefore, it can be applied in an environment with unknown non-stationary workloads, such as IaaS.

It is important to note that the model proposed in this paper is based on Markov chains requiring a few fundamental modeling assumptions. First of all, the workload must satisfy the Markov property, which implies memoryless state transitions and an exponential distribution of state transition delays. These assumptions must be taken into account in an assessment of the applicability of the proposed model to a particular system. A more detailed discussion of the modeling assumptions and validation of the assumptions is given in Section 6.4.

The remainder of the paper is organized as follows. In Section 2, we discuss the related work followed by the objective of host overload detection and workload independent QoS metric in Sections 3 and 4 respectively. We introduce an optimal offline algorithm for the problem of host overload detection in Section 5. In Section 6, we introduce a Markov model for the problem of host overload detection and approximate it for unknown non-stationary workloads in Section 7. In Section 8, we propose a control algorithm followed by a multi-core CPU model in Section 9 and an experimental evaluation in Section 10. We conclude the paper with Section 11 discussing the results and future research directions.

2 Related Work

Prior approaches to host overload detection for energy-efficient dynamic VM consolidation proposed in the literature can be broadly divided into 3 categories: periodic adaptation of the VM placement (no overload detection), threshold-based heuristics, and decision-making based on statistical analysis of historical data. One of the first works, in which dynamic VM consolidation has been applied to minimize energy consumption in a data center, has been performed by Nathuji and Schwan [3]. They

2. The PlanetLab project. http://www.planet-lab.org/

explored the energy benefits obtained by consolidating VMs using migration and found that the overall energy consumption can be significantly reduced. Verma et al. [4] modeled the problem of power-aware dynamic VM consolidation as a bin-packing problem and proposed a heuristic that minimizes the data center's power consumption, taking into account the VM migration cost. However, the authors did not apply any algorithm for determining *when* it is necessary to optimize the VM placement – the proposed heuristic is simply periodically invoked to adapt the placement of VMs.

Zhu et al. [5] studied the dynamic VM consolidation problem and applied a heuristic of setting a static CPU utilization threshold of 85% to determine when a host is overloaded. The host is assumed to be overloaded when the threshold is exceeded. The 85% utilization threshold has been first introduced and justified by Gmach et al. [6] based on their analysis of workload traces. In their more recent work, Gmach et al. [7] investigated benefits of combining both periodic and reactive threshold-based invocations of the migration controller. VMware Distributed Power Management [8] operates based on the same idea with the utilization threshold set to 81%. However, static threshold heuristics are unsuitable for systems with unknown and dynamic workloads, as these heuristics do not adapt to workload changes and do not capture the time-averaged behavior of the system. We have enhanced the static threshold heuristic in our previous work [15] by dynamically adapting the value of the threshold according to statistical analysis of the workload history. In this paper, we use static and dynamic threshold heuristics as benchmark algorithms in the experimental evaluation of the proposed approach.

Jung et al. [9] investigated the problem of dynamic consolidation of VMs running multi-tier web-applications to optimize a global utility function, while meeting SLA requirements. The approach is workload-specific, as the SLA requirements are defined in terms of the response time precomputed for each transaction type of the applications. When the request rate deviates out of an allowed interval, the system adapts the placement of VMs and the states of the hosts. Zheng et al. [10] proposed automated experimental testing of the efficiency of a reallocation decision prior to its application, once the response time, specified in the SLAs, is violated. In the approach proposed by Kumar et al. [11], the resource allocation is adapted when the application's SLAs are violated. Wang et al. [17] applied control loops to manage resource allocation under response time QoS constraints at the cluster and server levels. If the resource capacity of a server is insufficient to meet the applications' SLAs, a VM is migrated from the server. All these works are similar to threshold-based heuristics in that they rely on instantaneous values of performance characteristics but do not leverage the observed history of system states to estimate the future behavior of the system and optimize time-averaged performance metrics.

Guenter et al. [12] implemented an energy-aware dynamic VM consolidation system focused on web-applications, whose SLAs are defined in terms of the response time. The authors applied weighted linear regression to predict the future workload and proactively optimize the resource allocation. This approach is in line with the Local Regression (LR) algorithm proposed in our previous work [15], which we use as one of the benchmark algorithms in this paper. Bobroff et al. proposed a server overload forecasting technique based on time-series analysis of historical data [13]. Unfortunately, the algorithm description given in the paper is too high level, which does not allow us to implement it to compare with our approach. Weng et al. [18] proposed a load-balancing system for virtualized clusters. A cluster-wide cost of the VM allocation is periodically minimized to detect overloaded and underloaded hosts, and reallocate VMs. This is a related work but with the opposite objective – the VMs are deconsolidated to balance the load across the hosts.

As mentioned above, the common limitations of the prior works are that, due to their heuristic basis, they lead to sub-optimal results and do not allow the system administrator to explicitly set a QoS goal. In this work, we propose a novel approach to the problem of host overload detection inspired by the work of Benini et al. [19] on power management of electronic systems using Markov decision processes. We build a Markov chain model for the case of a known stationary workload and a given state configuration, and using a workload independent QoS metric derive a Non-Linear Programming (NLP) problem formulation. The solution of the derived NLP problem is the optimal control policy that maximizes the time between VM migrations under the specified QoS constraint in the online setting. Since most real-world systems, including IaaS, experience highly variable non-stationary workloads, we apply the Multisize Sliding Window workload estimation technique proposed by Luiz et al. [16] to heuristically adapt the proposed model to non-stationary stochastic environments and practical applications. Although the final approach is a heuristic, in contrast to the related works it is based on an analytical model that allows the computation of an optimal control policy for any known stationary workload and a given state configuration.

3 The Objective of a Host Overload Detection Algorithm

In this section, we show that to improve the quality of VM consolidation, it is necessary to maximize the time intervals between VM migrations from overloaded hosts. Since VM consolidation is applied to reduce the number of active physical hosts, the quality of VM consolidation is inversely proportional to H, the mean number of active hosts over n time steps:

$$H = \frac{1}{n} \sum_{i=1}^{n} a_i, \tag{1}$$

where a_i is the number of active hosts at the time step $i = 1, 2, \ldots, n$. A lower value of H represents a better quality of VM consolidation.

To investigate the impact of decisions made by host overload detection algorithms on the quality of VM consolidation, we consider an experiment, where at any time step the host overload detection algorithm can initiate a migration from a host due to an overload. There are two possible consequences of a decision to migrate a VM relevant to host overload detection: *Case 1*, when a VM to be migrated from an overloaded host cannot be placed on another active host due to insufficient resources, and therefore, a new host has to be activated to accommodate the VM; and *Case 2*, when a VM to be migrated can be placed on another active host. To study host overload detection in isolation, we assume that no hosts are switched off during the experiment, i.e., once a host is activated, it remains active until n.

Let p be the probability of *Case 1*, i.e., an extra host has to be activated to migrate a VM from an overloaded host determined by the host overload detection algorithm. Then, the probability of *Case 2* is $(1 - p)$. Let T be a random variable denoting the time between two subsequent VM migrations initiated by the host overload detection algorithm. The expected number of VM migrations initiated by the host overload detection algorithm over n time steps is $n/E[T]$, where $E[T]$ is the expected inter-migration time.

Based on the definitions given above, we can define $X \sim B(n/E[T], p)$, a binomially distributed random variable denoting the number of extra hosts switched on due to VM migrations initiated by the host overload detection algorithm over n time steps. The expected number of extra hosts activated is $E[X] = np/E[T]$. Let A be a random variable denoting the time during which an extra host is active between the time steps 1 and n. The expected value of A can be defined as follows:

$$\begin{aligned} E[A] =& \sum_{i=1}^{\lfloor \frac{n}{E[T]} \rfloor} (n - (i-1)E[T])p \\ =& \left\lfloor \frac{n}{E[T]} \right\rfloor \frac{p}{2} \left(n + n - \left(\left\lfloor \frac{n}{E[T]} \right\rfloor - 1 \right) E[T] \right) \\ \leq& \frac{np}{2} \left(1 + \frac{n}{E[T]} \right). \end{aligned} \tag{2}$$

Let us rewrite (1) as follows:

$$\begin{aligned} H =& \frac{1}{n} \sum_{i=1}^{n} a_i \\ =& \frac{1}{n} \sum_{i=1}^{n} a_1 + \frac{1}{n} \sum_{i=1}^{n} (a_i - a_1) \\ =& a_1 + \frac{1}{n} \sum_{i=1}^{n} (a_i - a_1). \end{aligned} \tag{3}$$

The first term a_1 is a constant denoting the number of hosts that have been initially active and remain active until the end of the experiment. The second term $H^* = \frac{1}{n} \sum_{i=1}^{n} (a_i - a_1)$ is the mean number of hosts switched on due to VM migrations being active per unit of time over n time steps. We are interested in analyzing the average behavior, and thus estimating the expected value of H^*. It is proportional to a product of the expected number of extra hosts switched on due to VM migrations and the expected activity time of an extra host normalized by the total time, as shown in (4).

$$\begin{aligned} E[H^*] \propto& \frac{1}{n} E[X] E[A] \\ \leq& \frac{1}{n} \frac{np}{E[T]} \frac{np}{2} \left(1 + \frac{n}{E[T]} \right) \\ =& \frac{np^2}{2E[T]} \left(1 + \frac{n}{E[T]} \right). \end{aligned} \tag{4}$$

Since the objective is to improve the quality of VM consolidation, it is necessary to minimize $E[H^*]$. From (4), the only variable that can be directly controlled by a host overload detection algorithm is $E[T]$; therefore, to minimize $E[H^*]$ the objective of a host overload detection algorithm is to maximize $E[T]$, i.e., to maximize the mean time between migrations from overloaded hosts.

4 A Workload Independent QoS Metric

To impose QoS requirements on the system, we apply an extension of the *workload independent QoS metric* introduced in our previous work [15]. We define that a host can be in one of two states in regard to its load level: (1) serving regular load; and (2) being overloaded. It is assumed that if a host is overloaded, the VMs allocated to the host are not being provided with the required performance level leading to performance degradation. To evaluate the overall performance degradation, we define a metric denoted Overload Time Fraction (OTF):

$$OTF(u_t) = \frac{t_o(u_t)}{t_a}, \tag{5}$$

where u_t is the CPU utilization threshold distinguishing the non-overload and overload states of the host; t_o is the time, during which the host has been overloaded, which is a function of u_t; and t_a is the total time, during which the host has been active. Using this metric, SLAs can be defined as the maximum allowed value of OTF. For example, if in the SLAs it is stated that OTF must be less or equal to 10%, it means that on average a host is allowed to be overloaded for not more than 10% of its activity time. Since the provider is interested in maximizing the resource utilization while meeting the SLAs, from his perspective this requirement corresponds to the QoS goal of $OTF \to 10\%$, while $OTF \leq 10\%$. The definition of the metric for a single host can be extended to a set of hosts by substituting the time values by the aggregated time values over the set of hosts.

The exact definition of the state of a host, when it is overloaded, depends on the specific system requirements. However, the value of the CPU utilization threshold u_t defining the states of a host does not affect the

model proposed in this paper, the model allows setting the threshold to any value. For example, in our experiments, we define that a host is overloaded, when its CPU utilization is 100%, in which case the VMs allocated to this host do not get the required CPU capacity leading to performance degradation. The reasoning behind this is the observation that if a host serving applications is experiencing 100% utilization, the performance of the applications is constrained by the host's capacity; therefore, the VMs are not being provided with the required performance level.

It has been claimed in the literature that the performance of servers degrade, when their load approaches 100% [20], [21]. For example, the study of Srikantaiah et al. [21] has shown that the performance delivered by the CPU degrades when the utilization is higher than 70%. If due to system requirements, it is important to avoid performance degradation, the proposed OTF metric allows the specification of the CPU utilization threshold at the required level below 100%. The host is considered to be overloaded, when the CPU utilization is higher than the specified threshold.

In general, other system resources, such as memory, disk, and network bandwidth, should also be take into account in the definition of QoS requirements. However, in this paper we only consider the CPU, as it is one of the main resources that are usually oversubscribed by Cloud providers. Therefore, in our analysis we assume that the other system resources are not significantly oversubscribed and do not become performance bottlenecks.

Verma et al. [22] proposed a similar metric for estimating the SLA violation level in a system, which they defined as the number of time instances, when the capacity of a server is less than the demand of all applications placed on it. However, their metric shows a non-normalized absolute value, which, for example, cannot be used to compare systems processing the same workload for different periods of time. In contrast, the OTF metric is normalized and does not depend on the length of the time period under consideration.

In the next section, based on the objective of a host overload detection algorithm derived in Section 3 and the OTF metric introduced in this section, we propose an optimal offline algorithm for the problem of host overload detection and prove its optimality.

5 An Optimal Offline Algorithm

As shown in Section 3, it is necessary to maximize the mean time between VM migrations initiated by the host overload detection algorithm, which can be achieved by maximizing each individual inter-migration time interval. Therefore, we limit the problem formulation to a single VM migration, i.e., the time span of a problem instance is from the end of a previous VM migration and to the end of the next. Given the results of Sections 3 and 4, the problem of host overload detection can be formulated as an optimization problem (6)-(7).

$$t_a(t_m, u_t) \rightarrow \max \tag{6}$$

$$\frac{t_o(t_m,u_t)}{t_a(t_m,u_t)} \leq M, \tag{7}$$

where t_m is the time when a VM migration has been initiated; u_t is the CPU utilization threshold defining the overload state of the host; $t_o(t_m, u_t)$ is the time, during which the host has been overloaded, which is a function of t_m and u_t; t_a is the total time, during which the host has been active, which is also a function of t_m and u_t; and M is the limit on the maximum allowed OTF value, which is a QoS goal expressed in terms of OTF. The aim of a host overload detection algorithm is to select the t_m that maximizes the total time until a migration, while satisfying the constraint (7). It is important to note that the optimization problem (6)-(7) is only relevant to host *overload* detection, and does not relate to host underload situations. In other words, maximizing the activity time of a host is only important for highly loaded hosts. Whereas for underloaded hosts, the problem is the opposite – the activity time needs to be minimized; however, this problem is not the focus of the current paper and should be investigated separately.

In the offline setting, the state of the system is known at any point in time. Consider an offline algorithm that passes through the history of system states backwards starting from the last known state. The algorithm decrements the time and re-calculates the OTF value $\frac{t_o(t_m,u_t)}{t_a(t_m,u_t)}$ at each iteration. The algorithm returns the time that corresponds to the current iteration if the constraint (7) is satisfied (Algorithm 1).

Algorithm 1 An Optimal Offline Algorithm (OPT)

Input: A system state $history$
Input: M, the maximum allowed OTF
Output: A VM migration time

```
1: while history is not empty do
2:   if OTF of history ≤ M then
3:     return the time of the last history state
4:   else
5:     drop the last state from history
6:   end if
7: end while
```

Theorem 1: Algorithm 1 is an optimal offline algorithm (OPT) for the problem of host overload detection.

Proof: Let the time interval covered by the system state history be $[t_0, t_n]$, and t_m be the time returned by Algorithm 1. Then, according to the algorithm the system states corresponding to the time interval $(t_m, t_n]$ do not satisfy the constraint (7). Since t_m is the right bound of the interval $[t_0, t_m]$, then t_m is the maximum possible time that satisfies the constraint (7). Therefore, t_m is the solution of the optimization problem (6)-(7), and Algorithm 1 is an optimal offline algorithm for the problem of host overload detection. □

6 A Markov Chain Model for the Host Overload Detection Problem

In this section, we base our model on the definitions of Markov chains, a mathematical framework for statistical modeling of real-world processes. Bolch [23] provides a detailed introduction to Markov chains.

6.1 The Host Model

Each VM allocated to a host at each point in time utilizes a part of the CPU capacity determined by the application workload. The CPU utilization created over a period of time by a set of VMs allocated to a host constitutes the host's workload. For the initial analysis, we assume that the workload is known *a priori*, stationary, and satisfies the Markov property. In other words, the CPU utilization of a host measured at discrete time steps can be described by a single time-homogeneous DTMC.

There is a controller component, which monitors the CPU utilization of the host and according to a host overload detection algorithm decides when a VM should be migrated from the host to satisfy the QoS requirements, while maximizing the time between VM migrations. According to Section 5, we limit the problem formulation to a single VM migration, i.e., the time span of a problem instance is from the end of a previous VM migration to the end of the next.

To describe a host as a DTMC, we assign states to N subsequent intervals of the CPU utilization. For example, if $N = 11$, we assign the state 1 to all possible values of the CPU utilization within the interval $[0\%, 10\%)$, 2 to the CPU utilization within $[10\%, 20\%)$, ..., N to the value 100%. The state space $\mathcal{S}$ of the DTMC contains N states, which correspond to the defined CPU utilization intervals. Using this state definition and knowing the workload of a host in advance, by applying the Maximum Likelihood Estimation (MLE) method it is possible to derive a matrix of transition probabilities $\mathbf{P}$. The matrix is constructed by estimating the probabilities of transitions $\widehat{p}_{ij} = \frac{c_{ij}}{\sum_{k \in \mathcal{S}} c_{ik}}$ between the defined N states of the DTMC for $i, j \in \mathcal{S}$, where c_{ij} is the number of transitions between states i and j.

We add an additional state $(N+1)$ to the Markov chain called an *absorbing state*. A state $k \in \mathcal{S}$ is said to be an absorbing state if and only if no other state of the Markov chain can be reached from it, i.e., $p_{kk} = 1$. In other words, once the Markov chain reaches the state k, it stays in that state indefinitely. The resulting extended state space is $\mathcal{S}^* = \mathcal{S} \cup \{(N+1)\}$. For our problem, the absorbing state $(N+1)$ represents the state where the DTMC transitions once a VM migration is initiated. According to this definition, the control policy can be described by a vector of the probabilities of transitions from any non-absorbing state to the absorbing state $(N+1)$, i.e., the probabilities of VM migrations, which we denote m_i, where $i \in \mathcal{S}$. To add the state $(N+1)$ into the model, the initial transition probability matrix $\mathbf{P}$ is extended with a column of unknown transition probabilities $\mathbf{m} = [m_i]$ $\forall i \in \mathcal{S}$ resulting in an extended matrix of transition probabilities $\mathbf{P}^*$:

$$\mathbf{P}^* = \begin{pmatrix} p^*_{11} & \cdots & p^*_{1N} & m_1 \\ \vdots & \ddots & \vdots & \vdots \\ p^*_{N1} & \cdots & p^*_{NN} & m_N \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{8}$$

where p^*_{ij} are defined as follows:

$$p^*_{ij} = p_{ij}(1 - m_i), \quad \forall i, j \in \mathcal{S}. \tag{9}$$

In general, the workload experienced by the host's VMs can lead to any CPU utilization from 0% to 100%; therefore, the original DTMC can be assumed to be ergodic. We will restrict the extended DTMC to the states in $\mathcal{S}$; therefore, using $\mathbf{Q} = \mathbf{P} - \mathbf{I}$ [23], the extended matrix of transition probabilities $\mathbf{P}^*$ can be transformed into a corresponding extended matrix of transition rates $\mathbf{Q}^*$:

$$\mathbf{Q}^* = \begin{pmatrix} p^*_{11} - 1 & \cdots & p^*_{1N} & m_1 \\ \vdots & \ddots & \vdots & \vdots \\ p^*_{N1} & \cdots & p^*_{NN} - 1 & m_N \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{10}$$

In the next section, we formulate the QoS constraint in terms of the introduced model, derived extended matrix of transition rates $\mathbf{Q}^*$, and OTF metric.

6.2 The QoS Constraint

Let

$$\mathbf{L}(t) = \int_0^t \boldsymbol{\pi}(u)\,\mathrm{d}u, \tag{11}$$

then $\mathbf{L}_i(t)$ denotes the *total expected time* the CTMC spends in the state i during the interval $[0, t)$. By integrating an equation for the *unconditional state probability vector* $\boldsymbol{\pi}(t)$: $\mathrm{d}\boldsymbol{\pi}(t)/\mathrm{d}t = \boldsymbol{\pi}(t)\mathbf{Q}$ on both sides, a new differential equation for $\mathbf{L}(t)$ is derived [23]:

$$\frac{\mathrm{d}\,\mathbf{L}(t)}{\mathrm{d}t} = \mathbf{L}(t)\mathbf{Q} + \boldsymbol{\pi}(0), \quad \mathbf{L}(0) = 0. \tag{12}$$

The expected time spent by the CTMC before absorption can be calculated by finding the limit $\mathbf{L}_{\mathcal{S}}(\infty) = \lim_{t\to\infty} \mathbf{L}_{\mathcal{S}}(t)$ restricting the state space to the states in $\mathcal{S}$. The limit exists due to a non-zero probability of a transition to the absorbing state $(N+1)$. However, the limit does not exist for the state $(N+1)$. Therefore, to calculate $\mathbf{L}_{\mathcal{S}}(\infty)$, the extended infinitesimal generator matrix $\mathbf{Q}^*$ is restricted to the states in $\mathcal{S}$, resulting in a matrix $\mathbf{Q}^*_{\mathcal{S}}$ of the size $N \times N$. The initial probability vector $\boldsymbol{\pi}(0)$ is also restricted to the states in $\mathcal{S}$ resulting in $\boldsymbol{\pi}_{\mathcal{S}}(0)$. Restricting the state space to non-absorbing states allows the computation of $\lim_{t\to\infty}$ on both sides of (12) resulting in the following *linear* equation [23]:

$$\mathbf{L}_{\mathcal{S}}(\infty)\mathbf{Q}^*_{\mathcal{S}} = -\boldsymbol{\pi}_{\mathcal{S}}(0). \tag{13}$$

Let N denote the state of a host when it is overloaded, e.g., when the CPU utilization is equal to 100%, then the expected time spent in the state N before absorption can be calculated by finding $L_N(\infty)$ from a solution of the system of linear equations (13). Similarly, the total expected time of the host being active can be found as $\sum_{i \in \mathcal{S}} L_i(\infty)$. Letting the VM migration time be T_m, the expected OTF can be calculated as follows:

$$OTF = \frac{T_m + L_N(\infty)}{T_m + \sum_{i \in \mathcal{S}} L_i(\infty)}. \tag{14}$$

6.3 The Optimization Problem

By the solution of (13), *closed-form* equations for $L_1(\infty), L_2(\infty), \ldots, L_N(\infty)$ are obtained. The unknowns in these equations are $m_1, m_2, \ldots, m_N$, which completely describe the policy of the controller. For our problem, the utility function is the total expected time until absorption, as the objective is to maximize the inter-migration time. To introduce the QoS goal in the problem formulation, we specify a limit M on the maximum allowed value of the OTF metric as a constraint resulting in the following optimization problem:

$$\begin{aligned} &\sum_{i \in \mathcal{S}} L_i(\infty) \rightarrow \max \\ &\frac{T_m + L_N(\infty)}{T_m + \sum_{i \in \mathcal{S}} L_i(\infty)} \leq M. \end{aligned} \tag{15}$$

The equations (15) form an NLP problem. The solution of this NLP problem is the vector $\mathbf{m}$ of the probabilities of transitions to the absorbing state, which forms the optimal control policy defined as a PMF $\mathbf{m} = [m_i]\ \forall i \in \mathcal{S}$. At every time step, the optimal control policy migrates a VM with probability m_i, where $i \in \mathcal{S}$ is the current state. The control policy is *deterministic* if $\exists k \in \mathcal{S} : m_k = 1$ and $\forall i \in \mathcal{S}, i \neq k : m_i = 0$, otherwise the policy is *randomized*.

6.4 Modeling Assumptions

The introduced model allows the computation of the optimal control policy of a host overload detection controller for a given stationary workload and a given state configuration. It is important to take into account that this result is based on a few fundamental modeling assumptions. First of all, it is assumed that the system satisfies the Markov property, or in other words, the sojourn times (i.e., the time a CTMC remains in a state) are exponentially distributed. Assuming an exponential distribution of sojourn times may not be accurate in many systems. For instance, state transition delays can be deterministic due to a particular task scheduling, or follow other than exponential statistical distribution, such as a bell-shaped distribution. Another implication of the Markov property is the assumption of memoryless state transitions, which means that the future state can be predicted solely based on the knowledge of the current state. It is possible to envision systems, in which future states depend on more than one past state.

Another assumption is that the workload is stationary and known *a priori*, which does not hold in typical computing environments. In the next section, we show how the introduced model can be heuristically adapted to handle unknown non-stationary workloads. The proposed heuristically adapted model removes the assumption of stationary and known workloads; however, the assumptions implied by the Markov property must still hold. In Section 10, we evaluate the proposed heuristically adapted model and test the assumptions through a simulation study using real workload traces from more than a thousand PlanetLab VMs. The simulation results show that the model is efficient for this type of mixed computing workloads.

With a correct understanding of the basic model assumptions and careful assessment of the applicability of our model to a particular system, an application of the model can bring substantial performance benefits to the resource management algorithms. As demonstrated by our simulation study in Section 10, our approach outperforms the benchmark algorithms in terms of both the mean inter-migration time and the precision of meeting the specified QoS goal.

7 Non-Stationary Workloads

The model introduced in Section 6 works with the assumption that the workload is stationary and known. However, this is not the case in systems with unknown non-stationary workloads, such as IaaS. One of the ways to adapt the model defined for known stationary workloads to the conditions of initially unknown non-stationary workloads is to apply the Sliding Window workload estimation approach proposed by Chung et al. [24]. The base idea is to approximate a non-stationary workload as a sequence of stationary workloads $\mathcal{U} = (u_1, u_2, \ldots, u_{N_u})$ that are enabled one after another. In this model, the transition probability matrix $\mathbf{P}$ becomes a function of the current stationary workload $\mathbf{P}(u)$.

Chung et al. [24] called a policy that makes ideal decisions for a current stationary workload u_i the *best adaptive policy*. However, the best adaptive policy requires the perfect knowledge of the whole sequence of workloads $\mathcal{U}$ and the times, at which the workloads change. In reality, a model of a workload u_i can only be built based on the observed history of the system behavior. Moreover, the time at which the current workload changes is unknown. Therefore, it is necessary to apply a heuristic that achieves results comparable to the best adaptive policy. According to the Sliding Window approach, a time window of length l_w *slides* over time always capturing last l_w events. Let c_{ij} be the observed number of transitions between states i and j, $i, j \in \mathcal{S}$, during the last window l_w. Then, applying the MLE method, the transition probability p_{ij} is estimated as $\widehat{p}_{ij} = \frac{c_{ij}}{\sum_{k \in \mathcal{S}} c_{ik}}$. As the window length $l_w \rightarrow \infty$, the estimator $\widehat{p}_{ij}$ converges to the real value of the transition

probability p_{ij} if the length of the current stationary workload u_i is equal to l_w [24].

However, the Sliding Window approach introduces 3 sources of errors in the estimated workload:

1) The biased estimation error, which appears when the window length l_w is shorter than the length of a sequence of outliers.
2) The resolution error (referred to as the sampling error by Luiz et al. [16]), which is introduced due to the maximum precision of the estimates being limited to $1/l_w$.
3) The adaptation time (referred to as the identification delay by Luiz et al. [16]), which is a delay required to completely fill the window with new data after a switch from a stationary workload u_{i-1} to a new stationary workload u_i.

Luiz et al. [16] extended the Sliding Window approach by employing multiple windows with different sizes, where a window to use is selected dynamically using the information about the previous system state and variances of the estimates obtained from different windows. They referred to the extended approach as the Multisize Sliding Window approach. The proposed algorithm dynamically selects the best window size to eliminate the bias estimate error and benefit from both the small sampling error of large window sizes and small identification error of small window sizes. In this paper, we apply the Multisize Sliding Window approach to the model introduced in Section 6 to adapt it to initially unknown non-stationary workloads.

We adapt the calculation of the expected OTF (14) by transforming it to a function of $t \in \mathbb{R}^+$ to incorporate the information that is known by the algorithm at the time of decision making:

$$OTF(t) = \frac{T_m + y(t) + L_N(\infty)}{T_m + t + \sum_{i \in \mathcal{S}} L_i(\infty)}, \tag{16}$$

where $y(t)$ is a function returning the total time spent in the state N during the time interval $[0, t]$.

7.1 Multisize Sliding Window Workload Estimation

In this section we briefly introduce the Multisize Sliding Window approach; for more details, reasoning and analysis please refer to Luiz et al. [16]. A high level view of the estimation algorithm is shown in Figure 1. First of all, to eliminate the biased estimation error, the previous history is stored separately for each state in $\mathcal{S}$ resulting in S *state windows* W_i, $i = 1, 2, \ldots, S$. Let J, D, and N_J be positive numbers; $\mathcal{L} = (J, J + D, J + 2D, \ldots, J + (N_J - 1)D)$ a sequence of window sizes; and $l_{w_{\max}} = J + (N_J - 1)D$ the maximum window size. At each time t, the Previous State Buffer stores the system state s_{t-1} at the time $t - 1$ and controls the window selector, which selects a window W_i such that $s_{t-1} = i$. The notation $W_i^k(t)$ denotes the content of the window W_i in a position k at the time t. The selected window shifts its content one position to the right to store the

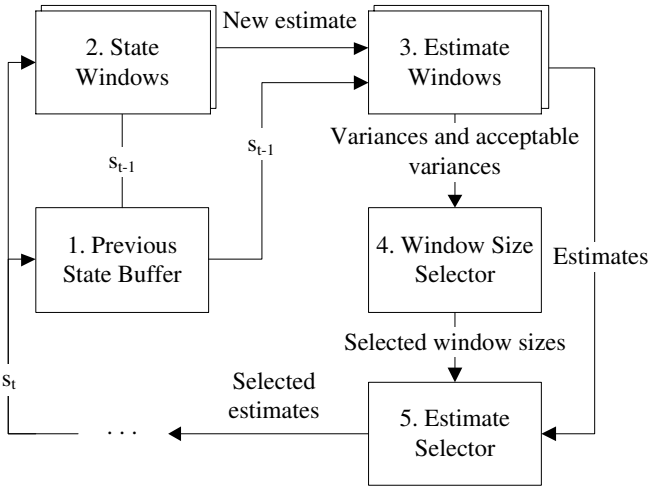


Fig. 1. The Multisize Sliding Window workload estimation

current system state: $W_i^{k+1}(t) = W_i^k(t)$, $\forall k = 1, \ldots, l_{w_{\max}}$; discards the rightmost element $W_i^{l_{w_{\max}}}(t)$; and stores s_t in the position $W_i^1(t)$. Once the selected state window W_i is updated, new probability estimates are computed based on this state window for all window sizes as follows:

$$\widehat{p}_{ij}(t, m) = \frac{\sum_{k=1}^{\mathcal{L}_m} (W_i^k(t) == j)}{\mathcal{L}_m}, \tag{17}$$

where "==" is the equivalence operation, i.e., $(1 == 1) = 1, (1 == 0) = 0$. A computed probability estimate is stored in N_J out of the SSN_J *estimate windows* $E_{ijm}(t)$, where $i, j \in \mathcal{S}$, and m is the estimate window size index, $1 \leq m \leq N_J$. N_J estimate windows $E_{ijm}(t)$ are selected such that $s_{t-1} = i$ and $s_t = j$, $\forall m = 1, \ldots, N_J$. Similarly to the update process of the state windows, the selected estimate windows shift their contents one position to the right, discard the rightmost element $E_{ijm}^{\mathcal{L}_m}(t)$, and store $\widehat{p}_{ij}(t, \mathcal{L}_m)$ in the position $E_{ijm}^1(t)$. To evaluate the precision of the probability estimates, the next step is to estimate the variance $S(i, j, t, m)$ of the probability estimates obtained from every updated estimate window:

$$\begin{aligned} \bar{p}_{ij}(t, m)) &= \frac{1}{\mathcal{L}_m} \sum_{k=1}^{\mathcal{L}_m} E_{ijm}^k(t), \\ S(i, j, t, m) &= \frac{1}{\mathcal{L}_m - 1} \sum_{k=1}^{\mathcal{L}_m} (E_{ijm}^k(t) - \bar{p}_{ij}(t, \mathcal{L}_m))^2, \end{aligned} \tag{18}$$

where $\bar{p}_{ij}(t, m)$ is the mean value of the probability estimates calculated from the state window W_i of length $\mathcal{L}_m$. In order to determine what values of the variance can be considered to be low enough, a function of acceptable variance $V_{ac}(\widehat{p}_{ij}(t, m), m)$ is defined [16]:

$$V_{ac}(\widehat{p}_{ij}(t, m), m) = \frac{\widehat{p}_{ij}(t, \mathcal{L}_m)(1 - \widehat{p}_{ij}(t, \mathcal{L}_m))}{\mathcal{L}_m}. \tag{19}$$

Using the function of acceptable variance, probability estimates are considered to be adequate if $S(i, j, t, m) \leq V_{ac}(\widehat{p}_{ij}(t, m), m)$. Based on the definitions given above, a window size selection algorithm can be defined (Algorithm 2). According to the selected window sizes,

transition probability estimates are selected from the estimate windows.

Algorithm 2 The Window Size Selection Algorithm

Input: J, D, N_J, t, i, j
Output: The selected window size
1: $l_w \leftarrow J$
2: **for** $k = 0$ to $N_J - 1$ **do**
3: **if** $S(i,j,t,k) \leq V_{ac}(\widehat{p}_{ij}(t,k),k)$ **then**
4: $l_w \leftarrow J + kD$
5: **else**
6: **break loop**
7: **end if**
8: **end for**
9: **return** l_w

The presented approach addresses the errors mentioned in Section 7 as follows:

1) The biased estimation error is eliminated by introducing dedicated history windows for each state: even if a burst of transitions to a particular state is longer than the length of the window, the history of transitions from the other states is preserved.
2) The sampling error is minimized by selecting the largest window size constrained by the acceptable variance function.
3) The identification error is minimized by selecting a smaller window size when the variance is high, which can be caused by a change to the next stationary workload.

8 The Control Algorithm

We refer to a control algorithm based on the model introduced in Section 6 as the Optimal Markov Host Overload Detection (MHOD-OPT) algorithm. We refer to the MHOD-OPT algorithm adapted to unknown nonstationary workloads using the Multisize Sliding Window workload estimation technique introduced in Section 7 as the Markov Host Overload Detection (MHOD) algorithm. A high-level view of the MHOD-OPT algorithm is shown in Algorithm 3. In the online setting, the algorithm is invoked periodically at each time step to make a VM migration decision.

Algorithm 3 The MHOD-OPT Algorithm

Input: Transition probabilities
Output: A decision on whether to migrate a VM
1: Build the objective and constraint functions
2: Invoke the brute-force search to find the **m** vector
3: **if** a feasible solution exists **then**
4: Extract the VM migration probability
5: **if** the probability is < 1 **then**
6: **return** **false**
7: **end if**
8: **end if**
9: **return** **true**

Closed-form equations for $L_1(\infty), L_2(\infty), \ldots, L_N(\infty)$ are precomputed offline from (13); therefore, the runtime computation is not required. The values of transition probabilities are substituted into the equations for $L_1(\infty), L_2(\infty), \ldots, L_N(\infty)$, and the objective and constraint functions of the NLP problem are generated by the algorithm. To solve the NLP problem, we applied a brute-force search algorithm with a step of 0.1, as its performance was sufficient for the purposes of simulations. In MHOD-OPT, a decision to migrate a VM is made only if either no feasible solution can be found, or the migration probability corresponding to the current state is 1. The justification for this is the fact that if a feasible solution exists and the migration probability is less than 1, then for the current conditions there is no hard requirement for an immediate migration of a VM.

Algorithm 4 The MHOD Algorithm

Input: A CPU utilization history
Output: A decision on whether to migrate a VM
1: **if** the CPU utilization history size $> T_l$ **then**
2: Convert the last CPU utilization value to a state
3: Invoke the Multisize Sliding Window estimation to obtain the estimates of transition probabilities
4: Invoke the MHOD-OPT algorithm
5: **return** the decision returned by MHOD-OPT
6: **end if**
7: **return** **false**

The MHOD algorithm shown in Algorithm 4 can be viewed as a wrapper over the MHOD-OPT algorithm, which adds the Multisize Sliding Window workload estimation. During the initial learning phase T_l, which in our experiments was set to 30 time steps, the algorithm does not migrate a VM. Once the learning phase is over, the algorithm applies the Multisize Sliding Window technique to estimate the probabilities of transitions between the states and invokes the MHOD-OPT algorithm passing the transition probability estimates as the argument. The result of the MHOD-OPT algorithm invocation is returned to the user.

9 The CPU Model

The models and algorithms proposed in this paper are suitable for both single core and multi-core CPU architectures. The capacity of a single core CPU is modeled in terms of its clock frequency F. A VM's CPU utilization u_i is relative to the VM's CPU frequency f_i and is transformed into a fraction of the host's CPU utilization U. These fractions are summed up over the N VMs allocated to the host to obtain the host's CPU utilization, as shown in (20).

$$U = F \sum_{i}^{N} f_i u_i. \tag{20}$$

For the purpose of the host overload detection problem, we model multi-core CPUs as proposed in our

TABLE 1
An artificial non-stationary workload

	0-60 s	60-86 s	86-160 s
p_{00}	1.0	0.0	1.0
p_{01}	0.0	1.0	0.0
p_{10}	1.0	0.0	1.0
p_{11}	0.0	1.0	0.0

TABLE 2
Comparison of MHOD, MHOD-OPT and OPT

	MHOD-30	MHOD-OPT-30	OPT-30
OTF	29.97%	16.30%	16.30%
Time	87	160	160

previous work [15]. A multi-core CPU with n cores each having a frequency f is modeled as a single core CPU with the nf frequency. In other words, F in (20) is replaced by nf. This simplification is justified, as applications and VMs are not tied down to a specific core, but can by dynamically assigned to an arbitrary core by a time-shared scheduling algorithm. The only physical constraint is that the CPU capacity allocated to a VM cannot exceed the capacity of a single core. Removing this constraint would require the VM to be executed on more than one core in parallel. However, automatic parallelization of VMs and their applications cannot be assumed.

10 Performance Evaluation

10.1 Importance of Precise Workload Estimation

The purpose of this section is to show that the precision of the workload estimation technique is important to achieve high performance of the MHOD algorithm. To show this, we constructed an artificial workload that illustrates a case when the MHOD algorithm with the Multisize Sliding Window workload estimation leads to lower performance compared to MHOD-OPT due to its inability to adapt quickly enough to a highly non-stationary workload.

We define that the host can be in one of two possible states $\{0, 1\}$, where the state 1 means that the host is being overloaded. Let the non-stationary workload be composed of a sequence of three stationary workloads, whose probabilities of transitions between the states are shown in Table 1. We used simulations to evaluate the algorithms. For this experiment, the OTF constraint was set to 30%, and the sequence of window sizes for the Multisize Sliding Window workload estimation was $(30, 40, 50, 60, 70, 80, 90, 100)$. The code of the simulations is written in Clojure[3]. To foster and encourage reproducibility of experiments, we have made the source code of all our simulations publicly available online[4].

3. The Clojure programming language. http://clojure.org/
4. https://github.com/beloglazov/tpds-2013-simulation/

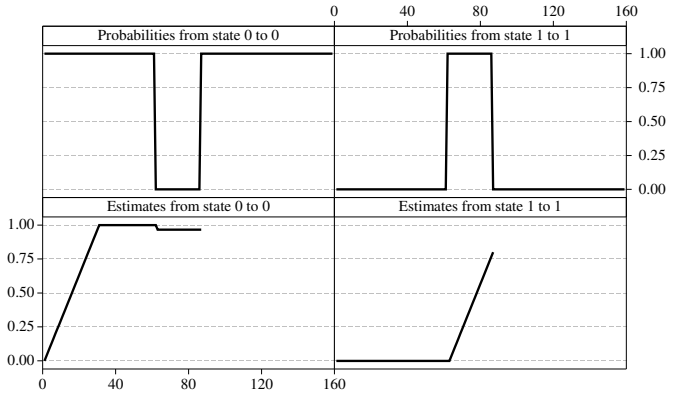


Fig. 2. The estimated $\widehat{p}_{00}$ compared to p_{00}

The simulation results are shown in Table 2. According to the results, for the workload defined in Table 1 the MHOD-OPT algorithm provides exactly the same performance as the optimal offline algorithm (OPT). However, the MHOD algorithm migrates a VM at the beginning of the third stationary workload because it is not able to immediately recognize the change of the workload, as shown for p_{00} and $\widehat{p}_{00}$ in Figure 2.

In summary, even though the Multisize Sliding Window workload estimation provides high quality of estimation [16], in some cases it may result in an inferior performance of the MHOD algorithm compared to MHOD-OPT. This result was expected, as MHOD-OPT skips the estimation phase and utilizes the knowledge of real transition probabilities. The artificial workload used in this section was specifically constructed to show that imprecise workload estimation may lead to unsatisfactory performance of the MHOD algorithm. However, as shown in the next section, the MHOD algorithm performs closely to OPT for real-world workloads.

10.2 Evaluation Using PlanetLab Workload Traces

In an environment with multiple hosts, the MHOD algorithm operates in a decentralized manner, where independent instances of the algorithm are executed on every host. Therefore, to evaluate the MHOD algorithm under a real-world workload, we simulated a single host with a quad-core CPU serving a set of heterogeneous VMs. The clock frequency of a single core of the host was set to 3 GHz, which according to the model introduced in Section 9 transforms into 12 GHz. These CPU characteristics correspond to a mid-range Amazon EC2 physical server type [25]. The amount of the host's memory was assumed to be enough for the VMs. The CPU frequency of a VM was randomly set to one of the values approximately corresponding to the Amazon EC2 instance types[5]: 1.7 GHz, 2 GHz, 2.4 GHz, and 3 GHz. The CPU utilization of the VMs was simulated based on the data provided as a part of the CoMon project, a monitoring infrastructure for PlanetLab [26]. The project provides the data measured every 5 minutes from more than a thousand VMs running in more than

5. http://aws.amazon.com/ec2/instance-types/

500 locations around the world. For our experiments, we have randomly chosen 10 days from the workload traces collected during March and April 2011.

For a simulation run, a randomly generated set of VMs with the CPU utilization traces assigned is allocated to the host. At each time step, the host overload detection algorithm makes a decision of whether a VM should be migrated from the host. The simulation runs until either the CPU utilization traces are over, or until a decision to migrate a VM is made by the algorithm. At the end of a simulation run, the resulting value of the OTF metric is calculated according to (5). The algorithm of assigning the workload traces to a set of VMs is presented in Algorithm 5. To avoid trivial cases and stress the algorithms with more dynamic workloads, we decided to filter the original workload traces. We constrained the maximum allowed OTF after the first 30 time steps to 10% and the minimum overall OTF to 20%. Using the workload assignment algorithm, we pregenerated 100 different sets of VMs that meet the defined OTF constraints and ran every algorithm for each set of VMs. The workload data used in the experiments are publicly available online[6].

Algorithm 5 The Workload Trace Assignment Algorithm

Input: A set of CPU utilization traces
Output: A set of VMs
1: Randomly select the host's minimum CPU utilization at the time 0 from 80%, 85%, 90%, 95%, and 100%
2: **while** the host's utilization $<$ the threshold **do**
3: Randomly select the new VM's CPU frequency
4: Randomly assign a CPU utilization trace
5: Add the new VM to the set of created VMs
6: **end while**
7: **return** the set of created VMs

10.2.1 Benchmark Algorithms

In addition to the optimal offline algorithm introduced in Section 5, we implemented a number of benchmark algorithms and ran them with different parameters to compare with the proposed MHOD algorithm. In this section we give a brief overview of the benchmark algorithms; a detailed description of each of them is given in our previous work [15]. The first algorithm is a simple heuristic based on setting a CPU utilization threshold (THR), which monitors the host's CPU utilization and migrates a VM if the defined threshold is exceeded. This threshold-based heuristic was applied in a number of related works [5]–[8]. The next two algorithms apply statistical analysis to dynamically adapt the CPU utilization threshold: based on the median absolute deviation (MAD), and on the interquartile range (IQR) [15].

Two other algorithms are based on estimation of the future CPU utilization using *local regression* (i.e., Loess method) and a modification of the method robust to outliers, referred to as *robust local regression* [15]. We denote these algorithms Local Regression (LR) and Local Regression Robust (LRR) respectively. The LR algorithm is in line with the regression-based approach proposed by Guenter et al. [12]. Another algorithm continuously monitors the host's OTF and decides to migrate a VM if the current value exceeds the defined parameter; we refer to this algorithm as the OTF Threshold (OTFT) algorithm. The last benchmark algorithm, the OTF Threshold Migration Time (OTFTM) algorithm, is similar to OTFT; however, it uses an extended metric that includes the VM migration time:

$$OTF(t_o, t_a) = \frac{T_m + t_o}{T_m + t_a}, \tag{21}$$

where t_o is the time, during which the host has been overloaded; t_a is the total time, during which the host has been active; and T_m is the VM migration time.

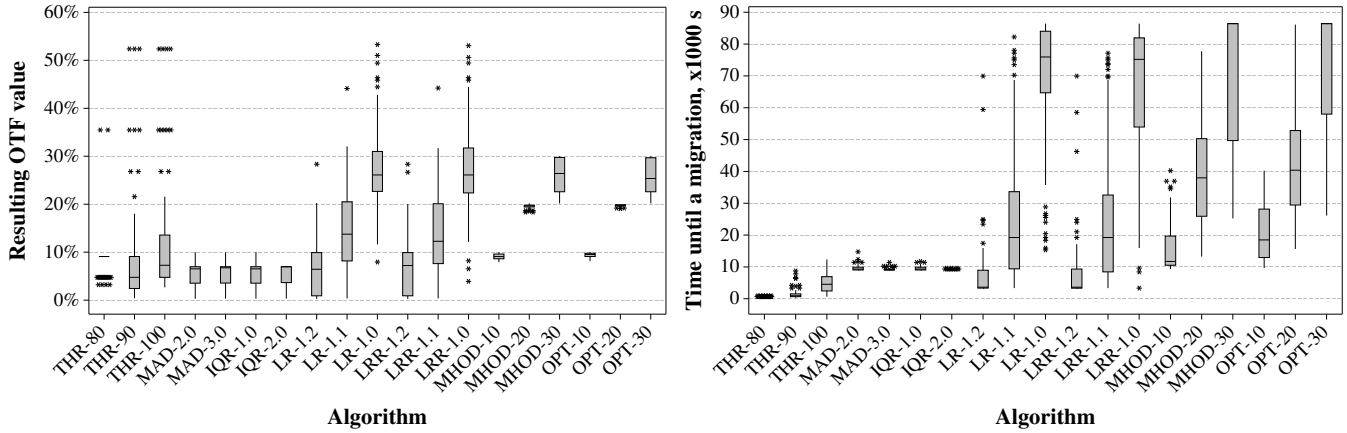
10.2.2 MHOD Compared with Benchmark Algorithms

To shorten state configuration names of the MHOD algorithm, we refer to them by denoting the thresholds between the utilization intervals. For example, a 3-state configuration $([0\%, 80\%), [80\%, 100\%), 100\%)$ is referred to as 80-100. We simulated the following 2- and 3-state configurations of the MHOD algorithm: 80-100, 90-100, and 100 (a 2-state configuration). We simulated each state configuration with the OTF parameter set to 10%, 20% and 30%. In our experiments, the VM migration time was set to 30 seconds.

In order to find out whether different numbers of states and different state configurations of the MHOD algorithm significantly influence the algorithm's performance in regard to the time until a migration and the resulting OTF value, we conducted paired t-tests. The tests on the produced time until a migration data for comparing MHOD 80-100 with MHOD 100 and MHOD 90-100 with MHOD 100 showed non-statistically significant differences with the p-values 0.20 and 0.34 respectively. This means that the simulated 2- and 3-state configurations of the MHOD algorithm on average lead to approximately the same time until a migration. However, there are statistically significant differences in the resulting OTF value produced by these algorithms: 0.023% with 95% Confidence Interval (CI) (0.001%, 0.004%) and p-value $=$ 0.033 for MHOD 100 compared with MHOD 80-100; and 0.022% with 95% CI (0.000%, 0.004%) and p-value $= 0.048$ for MHOD 100 compared with MHOD 90-100. However, differences in the resulting OTF value in the order of less than 0.1% are not practically significant; therefore, we conclude that the simulated 2- and 3-state configurations produce approximately the same results. Further in this section, we compare only the $([0\%, 100\%), 100\%)$ 2-state configuration of MHOD with the benchmark algorithms, as it requires simpler computations compared with the 3-state configurations.

The experimental results comparing the 2-state configuration of the MHOD algorithm (for the MHOD algorithm, the OTF parameter is denoted in the suffix

6. https://github.com/beloglazov/tpds-2013-workload/



(a) The resulting OTF value

(b) The time until a migration

Fig. 3. The resulting OTF value and time until a migration produced by the MHOD and benchmark algorithms

of the algorithm's name, e.g., for 10%, 20% and 30%: MHOD-10, MHOD-20 and MHOD-30) with the benchmark algorithms are depicted in Figures 3a and 3b. It is remarkable how closely the resulting OTF value of the MHOD algorithm resembles the value set as the parameter of the algorithm for 10% and 20%. The wider spread for 30% is explained by the characteristics of the workload: in many cases the overall OTF is lower than 30%, which is also reflected in the resulting OTF of the optimal offline algorithm (OPT-30). The experimental results show that the algorithm is capable of meeting the specified OTF goal, which is consistent with the theoretical model introduced in Section 6.

Figures 3a and 3b show that the THR, MAD and IQR algorithms are not competitive compared with the LR, LRR and MHOD algorithms, as the produced time until a migration is low and does not significantly improve by adjustments of the algorithm parameters. To compare the LR and LRR algorithms with the MHOD algorithms, we ran additional simulations of the MHOD algorithm with the OTF parameter matching the mean value of the resulting OTF produced by LR and LRR. The following values of the OTF parameter of the MHOD algorithm were set to match the mean resulting OTF values of LR and LRR: to match LR-1.2, LR-1.1 and LR-1.0 – 6.75%, 15.35% and 40% respectively; to match LRR-1.2, LRR-1.1 and LRR-1.0 – 6.76%, 14.9% and 36.6% respectively.

As intended, paired t-tests for the comparison of MHOD with LR and MHOD with LRR showed non-statistically significant differences in the resulting OTF values with both p-values > 0.7. Results of paired t-tests for comparing the time until a migration produced by the algorithms with matching resulting OTF values are shown in Table 3. The results of the comparison of the MHOD and LRR algorithms are graphically depicted in Figure 4. According to the results, there is a statistically significant difference in the time until a migration produced by the algorithms: the MHOD algorithm on average leads to approximately 16.4% and 15.4% better time until a migration than LR and LRR respectively with the same mean resulting OTF values. It is also interesting to notice from Figure 4 that the spread of the resulting OTF produced by the MHOD algorithm is much narrower than LRR's, which means that MHOD more precisely meets the QoS goal.

TABLE 3
Paired T-tests with 95% CIs for comparing the time until a migration produced by MHOD, LR and LRR

Alg. 1 ($\times 10^3$)	Alg. 2 ($\times 10^3$)	Diff. ($\times 10^3$)	p-value
MHOD (39.80)	LR (34.20)	5.61 (3.55, 7.66)	< 0.001
MHOD (38.69)	LRR (33.51)	5.17 (2.97, 7.38)	< 0.001

10.2.3 Comparison of MHOD with OTFT and OTFTM

OTFT and OTFTM are two other algorithms that apart from the MHOD algorithm allow explicit specification of the QoS goal in terms of the OTF parameter. To compare the performance of the OTFT, OTFTM and MHOD algorithms we introduce another performance metric. This metric is the percentage of SLA violations

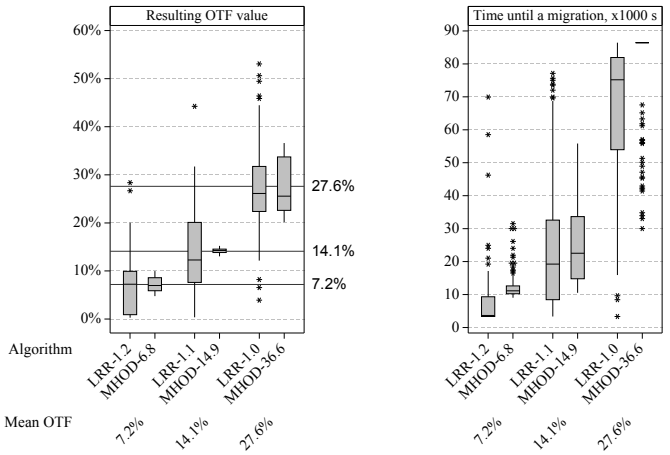


Fig. 4. Comparison of MHOD with LRR

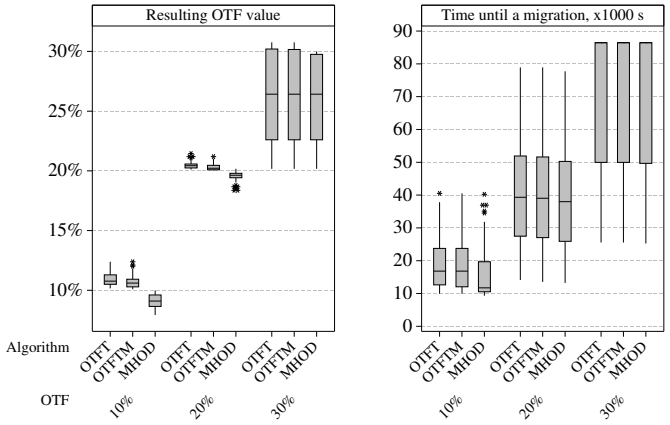


Fig. 5. Comparison of OTFT, OTFTM and MHOD

relatively to the total number of VM migrations, where SLA requirements are defined as $OTF \leq M$, M is the limit on the maximum allowed resulting OTF value. The SLA violation counter is incremented if after a VM migration the resulting OTF value is higher than the value M specified in the SLAs.

We simulated the OTFT, OTFTM and MHOD algorithms using the PlanetLab workload described earlier. We simulated the algorithms setting the following values of the OTF parameter as the SLA requirement: 10%, 20% and 30%. The simulation results are shown in Figure 5. The graphs show that MHOD leads to slightly lower resulting OTF values and time until a migration. The SLA violation levels caused by the algorithms are shown in Table 4. It is clear that the MHOD algorithm substantially outperforms the OTFT and OTFTM algorithms in the level of SLA violations leading to only 0.33% SLA violations, whereas both OTFT and OTFTM cause the percentage of SLA violations of 81.33%.

The obtained results can be explained by the fact that both OTFT and OTFTM are unable to capture the overall behavior of the system over time and fail to meet the SLA requirements. In contrast, the MHOD algorithm leverages the knowledge of the past system states and by estimating future states avoids SLA violations. For instance, in a case of a steep rise in the load, OTFT and OTFTM react too late resulting in an SLA violation. In contrast, MHOD acts more intelligently and by predicting the potential rise migrates a VM before an SLA violation occurs. As a result, for the simulated PlanetLab workload the MHOD algorithm keeps the level of SLA violations at less than 0.5%.

TABLE 4
SLA violations by OTFT, OTFTM and MHOD

OTF Parameter	OTFT	OTFTM	MHOD
10%	100/100	100/100	0/100
20%	100/100	100/100	1/100
30%	44/100	44/100	0/100
Overall	81.33%	81.33%	0.33%

TABLE 5
Paired T-tests for comparing MHOD with OPT

	OPT	MHOD	Difference	p-value
OTF	18.31%	18.25%	0.06% (-0.03, 0.15)	$= 0.226$
Time	45,767	41,128	4,639 (3617, 5661)	< 0.001

10.2.4 Comparison of MHOD with OPT

Figures 3a and 3b include the results produced by the optimal offline algorithm (OPT) for the same values of the OTF parameter set for the MHOD algorithm: 10%, 20% and 30%. The results of paired t-tests comparing the performance of OPT with MHOD are shown in Table 5. The results show that there is no statistically significant difference in the resulting OTF value, which means that for the simulated PlanetLab workload the MHOD algorithm on average leads to approximately the same level of adherence to the QoS goal as the optimal offline algorithm. There is a statistically significant difference in the time until a migration with the mean difference of 4,639 with 95% CI: (3617, 5661). Relatively to OPT, the time until a migration produced by the MHOD algorithm converts to 88.02% with 95% CI: (86.07%, 89.97%). This means that for the simulated PlanetLab workload, the MHOD algorithm on average delivers approximately 88% of the performance of the optimal offline algorithm, which is highly efficient for an online algorithm.

11 Conclusions and Future Directions

In this paper, we have introduced a Markov chain model and proposed a control algorithm for the problem of host overload detection as a part of dynamic VM consolidation. The model allows a system administrator to explicitly set a QoS goal in terms of the OTF parameter, which is a workload independent QoS metric. For a known stationary workload and a given state configuration, the control policy obtained from the Markov model optimally solves the host overload detection problem in the online setting by maximizing the mean inter-migration time, while meeting the QoS goal. Using the Multisize Sliding Window workload estimation approach, we have heuristically adapted the model to handle unknown non-stationary workloads. We have also proposed an optimal offline algorithm for the problem of host overload detection to evaluate the efficiency of the MHOD algorithm. The conducted experimental study has led to the following conclusions:

1) For the simulated PlanetLab workload, 3-state configurations of the MHOD algorithm on average produce approximately the same results as the $([0, 100), 100)$ 2-state configuration of the MHOD algorithm; therefore, we prefer the 2-state configuration, as it requires simpler computations.
2) The 2-state configuration of the MHOD algorithm outperforms the LRR algorithm, the best benchmark algorithm, by producing on average approximately 15.4% better time until a VM migration with

the same mean but much narrower spread of the resulting OTF value, leading to a better quality of VM consolidation according to Section 3.

3) The MHOD algorithm substantially outperforms the OTFT and OTFTM algorithms in the level of SLA violations resulting in less than 0.5% SLA violations compared to 81.33% of OTFT and OTFTM.
4) The MHOD algorithm on average provides approximately the same resulting OTF value and approximately 88% of the time until a VM migration produced by the optimal offline algorithm (OPT).
5) The MHOD algorithm enables explicit specification of a desired QoS goal to be delivered by the system through the OTF parameter, which is successfully met by the resulting value of the OTF metric.

The introduced model is based on Markov chains requiring a few fundamental assumptions. It is assumed that the workload satisfies the Markov property, which may not be true for all types of workloads. Careful assessment of the assumptions discussed in Section 6.4 is important in an investigation of the applicability of the proposed model to a particular system. However, our experimental study involving multiple mixed heterogeneous real-world workloads has shown that the algorithm is efficient in handling them. For the simulated PlanetLab workload the MHOD algorithm performed within a 12% difference from the performance of the optimal offline algorithm, which is highly efficient for an online algorithm. As a part of future work, we plan to implement the MHOD algorithm as an extension of the VM manager within the OpenStack Cloud platform[7] to evaluate the algorithm in a real system as a part of energy-efficient dynamic VM consolidation.

Acknowledgments

The authors would like to thank Rodrigo N. Calheiros, Saurabh Garg, Sivaram Yoganathan, and Andrey Kan from the University of Melbourne, and Egor Gladkikh from G-SCOP Laboratory of Grenoble for their input and constructive comments on improvements of the paper.

References

[1] J. Koomey, *Growth in data center electricity use 2005 to 2010*. Oakland, CA: Analytics Press, 2011.
[2] Gartner, Inc., *Gartner estimates ICT industry accounts for 2 percent of global CO2 emissions*. Gartner Press Release (April 2007).
[3] R. Nathuji and K. Schwan, "Virtualpower: Coordinated power management in virtualized enterprise systems," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 265–278, 2007.
[4] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," in *Proc. of the 9th ACM/IFIP/USENIX Intl. Conf. on Middleware*, 2008, pp. 243–264.
[5] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser *et al.*, "1000 Islands: Integrated capacity and workload management for the next generation data center," in *Proc. of the 5th Intl. Conf. on Autonomic Computing (ICAC)*, 2008, pp. 172–181.
[6] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and quality metrics," in *Proc. of the 38th IEEE Intl. Conf. on Dependable Systems and Networks (DSN)*, 2008, pp. 326–335.
[7] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or lets be friends," *Computer Networks*, vol. 53, no. 17, pp. 2905–2922, 2009.
[8] VMware Inc., "VMware distributed power management concepts and use," *Information Guide*, 2010.
[9] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in Cloud infrastructures," in *Proc. of the 30th Intl. Conf. on Distributed Computing Systems (ICDCS)*, 2010, pp. 62–73.
[10] W. Zheng, R. Bianchini, G. Janakiraman, J. Santos, and Y. Turner, "JustRunIt: Experiment-based management of virtualized data centers," in *Proc. of the 2009 USENIX Annual Technical Conf.*, 2009, pp. 18–33.
[11] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan, "vManage: Loosely coupled platform and virtualization management in data centers," in *Proc. of the 6th Intl. Conf. on Autonomic Computing (ICAC)*, 2009, pp. 127–136.
[12] B. Guenter, N. Jain, and C. Williams, "Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning," in *Proc. of the 30st Annual IEEE Intl. Conf. on Computer Communications (INFOCOM)*, 2011, pp. 1332–1340.
[13] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proc. of the 10th IFIP/IEEE Intl. Symp. on Integrated Network Management (IM)*, 2007, pp. 119–128.
[14] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in Computers, M. Zelkowitz (ed.)*, vol. 82, pp. 47–111, 2011.
[15] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience (CCPE)*, 2012, DOI: 10.1002/cpe.1867, (in press).
[16] S. O. Luiz, A. Perkusich, and A. M. N. Lima, "Multisize sliding window in workload estimation for dynamic power management," *IEEE Transactions on Computers*, vol. 59, no. 12, pp. 1625–1639, 2010.
[17] X. Wang and Y. Wang, "Coordinating power control and performance management for virtualized server clusters," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 22, no. 2, pp. 245–259, 2011.
[18] C. Weng, M. Li, Z. Wang, and X. Lu, "Automatic performance tuning for the virtualized cluster system," in *Proc. of the 29th Intl. Conf. on Distributed Computing Systems (ICDCS)*, 2009, pp. 183–190.
[19] L. Benini, A. Bogliolo, G. A. Paleologo, and G. D. Micheli, "Policy optimization for dynamic power management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 813–833, 1999.
[20] Q. Zheng and B. Veeravalli, "Utilization-based pricing for power management and profit optimization in data centers," *Journal of Parallel and Distributed Computing*, vol. 72, no. 1, pp. 27–34, 2011.
[21] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," *Cluster Computing*, vol. 12, pp. 1–15, 2009.
[22] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proc. of the 2009 USENIX Annual Technical Conf.*, 2009, pp. 28–28.
[23] G. Bolch, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Blackwell, 2006.
[24] E. Y. Chung, L. Benini, A. Bogliolo, Y. H. Lu, and G. D. Micheli, "Dynamic power management for nonstationary service requests," *IEEE Transactions on Computers*, vol. 51, no. 11, pp. 1345–1361, 2002.
[25] K. Mills, J. Filliben, and C. Dabrowski, "Comparing vm-placement algorithms for on-demand clouds," in *Proc. of the 3rd IEEE Intl. Conf. on Cloud Computing Technology and Science (CloudCom)*, 2011, pp. 91–98.
[26] K. S. Park and V. S. Pai, "CoMon: a mostly-scalable monitoring system for PlanetLab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 65–74, 2006.

7. The OpenStack Cloud platform. http://openstack.org/