Dynamic Load Management of Virtual Machines in Cloud Architectures

Mauro Andreolini, Sara Casolari, Michele Colajanni, and Michele Messori

Department of Information Engineering
University of Modena and Reggio Emilia, Italy
{mauro.andreolini,sara.casolari,michele.colajanni,
michele.messori}@unimore.it

Abstract. Cloud infrastructures must accommodate changing demands for different types of processing with heterogeneous workloads and time constraints. In a similar context, dynamic management of virtualized application environments is becoming very important to exploit computing resources, especially with recent virtualization capabilities that allow live sessions to be moved transparently between servers. This paper proposes novel management algorithms to decide about reallocations of virtual machines in a cloud context characterized by large numbers of hosts. The novel algorithms identify just the real critical instances and take decisions without recurring to typical thresholds. Moreover, they consider load trend behavior of the resources instead of instantaneous or average measures. Experimental results show that proposed algorithms are truly selective and robust even in variable contexts, thus reducing system instability and limit migrations when really necessary.

1 Introduction

Existing data centers are characterized by high operating costs, inefficiencies, and by myriads of distributed and heterogeneous servers that add complexity in terms of security and management. In order to improve data center efficiency, most enterprises are going to consolidate existing systems through virtualization solutions up to cloud centers. Logically pooling all system resources and centralizing resource management allow to increase overall utilization and lowering management costs. There are various approaches to virtualization (hardware virtualization up to micro-partitioning, operating system virtualization, software virtualization), but consolidation and virtualization by themselves do little to improve application performance. The question is whether huge increases in terms of system utilization correspond to an actual better efficiency or they are due to applications running poorly in those virtual environments.

Consolidation and virtualization deliver more computing resources to the organizations, but failure to tune applications to run on virtualized resources means that un-tuned applications are wasting processing cycles. In order to avoid to waste computing and storage resources it is necessary to optimize management

D.R. Avresky et al. (Eds.): Cloudcomp 2009, LNICST 34, pp. 201–214, 2010.

of these novel cloud systems architectures and virtualized servers. Overall performance analysis and runtime management in these contexts are becoming extremely complex, because they are a function not only of guest applications, but also of their interactions with other guest machines as they contend for processing and I/O resources of their host machine. We should consider that these modern cloud infrastructures must accommodate varying demands for different types of processing within certain time constraints, hence dynamic management of virtualized application environments is becoming very important. Indeed, automated workload management and balancing facilities can also lead to performance improvements while greatly reducing management cost. For these reasons, all recent virtualization management capabilities allow loads and live sessions to be moved transparently between processors or even servers, thus allowing applications to exploit unused computing resources regardless of whether those resources are located on local or remote servers. Dynamic capacity management can increase productivity but it requires continuous monitoring services and innovative runtime decision algorithms that represent the focus of this paper. In particular, we propose quite innovative algorithms for deciding when a physical host should migrate part of its load, which part of the load must be moved, and where should be moved. The difficulty of answering to these questions is also due to the observation that the performance measures referring to cloud system resources are characterized by spikes and extreme variability to the event that it is impossible to identify stable states if not for short periods.

The paper is organized as follow. Section 2 evidences main contributions to the state of the art. Section 3 describes the operating context and outlines the main phases of the proposed management algorithms. Section 4 considers the problem of identifying when a host really requires a load migration because of its critical state conditions, and proposes an innovative selection algorithm. Section 5 is devoted to the identification of the virtual machines that is convenient to migrate and of the physical hosts that can receive them. Section 6 concludes the paper with some final remarks and future work.

2 Related Work

There are several proposals for live migration of virtual machines in clusters of servers, and the most recent techniques aim to reduce downtime during migration. For example, the solution in Clark et al. [6] is able to transfer an entire machine with a downtime of few hundreds of milliseconds. Travostino et al. [7] migrate virtual machines on a WAN area with just 1-2 seconds of application downtime through lightpath [8]. Unlike these solutions that are based on a pre-copy of the state, Hines et al. [9] propose a post-copy which defers the transfer of a machine memory contents after its processor state has been sent to the target host. Migration techniques through Remote Direct Memory Access (RDMA) further reduce migration time and application downtime [10]. Although these mechanisms are rapidly improving, live migration remains an expensive operation that should be applied selectively especially in a cloud context characterized

by thousands of physical machines and about one order more of virtual machines. The focus of this paper on decision and management algorithms differentiates our work from literature on migration mechanisms. We evidence three main phases of the migration management process: to decide when a dynamic redistribution of load is necessary; how to choose which virtual machines is convenient to migrate; to place virtual machines to other physical machines.

Khanna et al. [4] monitor the resources (CPU and memory) of physical and virtual machines. If a resource exceeds a predefined threshold and some SLA is at risk, then the system migrates a virtual machine to another physical host. Sandpiper [11] is a mechanism that automates the task of monitoring and detecting hotspots; Bobroff et al. [12] propose an algorithm for virtual machine migration that aims to guarantee probabilistic SLAs. All these works decide when a dynamic redistribution of load is necessary through some threshold-based algorithms. We propose a completely different approach that decides about migration by avoiding thresholds on the server load, but considering the load profile evaluated through a CUSUM-based stochastic model [1].

The issues about to choose which virtual machines is convenient to migrate and where to place virtual machines have been often addressed through some global optimization approach. Entropy [13] decides about a dynamic placement of virtual machines on physical machines with the goal of minimizing the number of active physical servers and the number of migrations to reach a new configuration. Nguyen Van et al. [14] use the same approach but they integrate SLAs. Sandpiper [11] proposes two algorithms: a black-box approach that is agnostic about operating system and application; a gray-box approach that exploits operating system and application level statistics. It monitors CPU, memory and network resources to avoid SLA violations. The gray-box can also analyze application logs. The scheme proposed by Khanna et al. [4] moves the virtual machines with minimum utilization to the physical host with minimum available resources that are sufficient to host that virtual machines without violating the SLA. If there is no available host, it activates a new physical machine. Similarly, if the utilization of a physical machine falls below a threshold, the hosted servers are migrated elsewhere and the physical machine is removed from the pool of available hosts. Stage et al. [5] consider bandwidth consumed during migration. They propose a system that classifies the various loads and consolidate more virtual machines on each host based on typical periodic trends, if they exist. The paper in [12] adopts prediction techniques and a bin packing heuristic to allocate and place virtual machines while minimizing the number of activated physical machines. The authors propose also an interesting method for characterizing the gain that a virtual machine can achieve from dynamic migration. Our proposals differ from all these global optimization models that are applicable at runtime when there is a small set of machines to consider, but they cannot work in a cloud context characterized by thousands of physical machines. For these reasons, we analyze separately each physical host and its related virtual machines with the main goal of minimizing migrations just to the most severe instances. Instead of distributing the load evenly across a set of physical machines in order

to get an optimal resource utilization, we think that in a cloud context exposed to unpredictable demand and heterogeneous workload, a load sharing approach for migration of virtual machine is more realistic, in that it is possible to share the load across multiple servers, even if in an unequal way.

3 Management Algorithms for Load Migration

A typical cloud architecture consists of a huge set of physical machines (*host*), each of them equipped with some virtualization mechanisms, from hardware virtualization up to micro-partitioning, operating system virtualization, software virtualization. These mechanisms allow each machine to host a concurrent execution of several virtual machines (*guest*) each with its own operating system and applications.

To accommodate varying demands for different types of processing, the most modern cloud infrastructures include dynamic management capabilities and virtual machine mobility that is, the ability to move transparently virtual machines from one host to another. By migrating a guest from an overloaded host to another not critical host, it is possible to improve resource utilization and better load sharing. Independently of the migration techniques, they share a common management model: any decision algorithm for migration has to select one or more *sender* hosts from which some virtual machines are moved to other destination hosts, namely *receivers*. This paper addresses the main issues related to migration decisions, that is, it aims to answer to the following questions: when it is necessary to activate a migration, which guests of a sender host should migrate, and where they should be moved.

We are aware that any dynamic guest migration remains an expensive operation that consumes resources of the sender and receiver hosts as well as network bandwidth because of transfers of large chunks of data representing the memory state of the guests. In a cloud architecture with thousands of hosts, an abuse of guest migrations would devastate system and application performance. Hence, we should recur to migration in few severe instances during the cloud platform operations. In other words, a good algorithm for governing of dynamic migrations in a cloud architecture must guarantee a *reliable classification* of the host behavior (as sender, receiver and neutral) that can reduce the number of useless guests migrations, and a *selective precision* in deciding which (few) guests should migrate to another host.

The load state of a host is obtained through a periodic collection of measures from server monitors. These measures are typically characterized by noises and non stationary effects in the short-medium term, while there is some periodic behavior in a long term vision (day, week) that we do not consider in this paper. Figure 1 shows four load profiles (concerning host CPU utilizations) in a cloud architecture where physical machines host any type of virtual machines and applications, such as Web sites, databases, access controls, CMSes, mail servers, management software. In a similar context, the traditional threshold-based approach [4, 11] that classifies a host as a sender or receiver because its

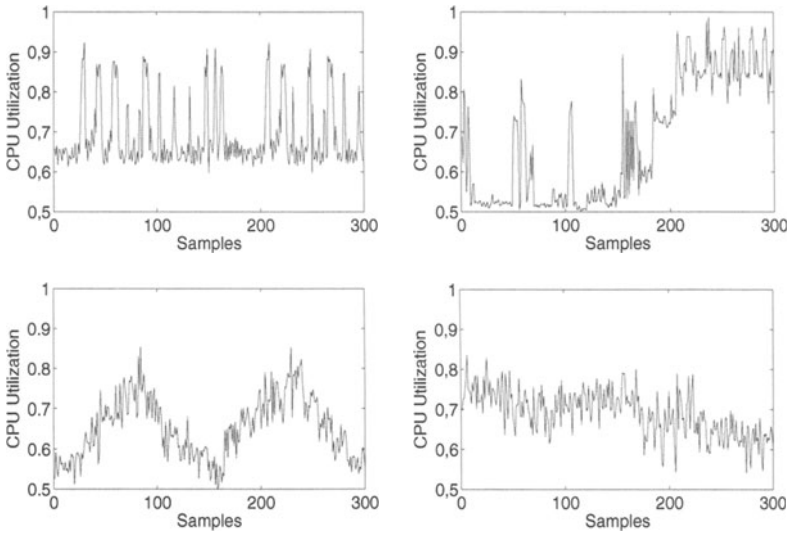


Fig. 1. Load profiles of hosts in a cloud architecture

load is beyond or below some given lines cannot work. This problem is even more serious in a cloud context with thousands of hosts where, at a checkpoint, a threshold may signal hundreds of senders and, at the successive checkpoint, the number of senders can become few dozen or, even worse, remain in the order of hundreds but where most servers are different from those of the previous set. The decision about which guests is useful to migrate from one server to another is affected by similar problems if we adopt some threshold-based method.

The primary goal of this paper is to provide robust and selective reallocations of guests in a context of thousands of hosts, under the consideration that high performance and low overheads are guaranteed only if we are able to limit the number of migrations to few really necessary instances. To this purpose, we propose novel algorithms for dynamic load management in a cloud architecture that take decisions without fixed thresholds and that consider trend behavior instead of instantaneous or average load measures.

The proposed management algorithm is activated periodically (typically in the order of few minutes) and, at each checkpoint, it aims at defining three sets: sender hosts, receiver hosts, and migrating guests, where their cardinalities are denoted as S, R, and G, respectively. Let also N be the total number of hosts. We have to guarantee that $N \geq S + R$, and that the intersection between the set of sender hosts and of receiver hosts is null. The algorithm is based on the following four phases.

- **Phase 1: Selection of sender hosts.** The first action requires the selection of the set of sender hosts that require the migration of some of their guests. We describe our strategy that is based on the CUSUM models [1] in Section 4.

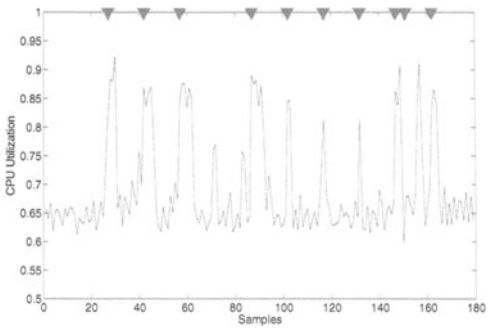
The idea is to have a selective and robust algorithm so that the cardinality S of the set of senders is much smaller than the total number of hosts that is, $S \ll N$.

- **Phase 2: Selection of guests.** Once selected the senders, we have to evaluate how many and which guests it is convenient to migrate. To this purpose, in Section 5 we propose an algorithm that is able to select the most critical guests for each server on the basis of a *load trend-based* model instead of traditional approaches based on instantaneous or average load measures. Even for this phase, the goal is to limit the number of guests for each host that should migrate, so that $G < (N - S)$. If this does not occur after the first evaluation, the guest selection proceeds iteratively until the constraint is satisfied. (It is worth to observe that no experiment required an iteration.)
- **Phase 3: Selection of receiver hosts.** Once selected the guests that have to migrate, we have to define the set of receiver hosts. To this purpose, we do not propose any specific innovative algorithm. From our past experience in other geographically distributed architectures and initial experiments on cloud architectures, we can conclude that the major risk we want to avoid is a dynamic migration that tends to overload some receiver hosts so that at the successive checkpoint a receiver may become a sender. Similar fluctuations devastate system performance and stability. Hence, our idea is to set $R = G$ so that each receiver host receives at most one guest. The selected receivers are the R hosts that exhibit the lowest load computed on the basis of the trend model described in Section 5.
- **Phase 4: Assignment of guests.** The guests selected in the Phase 2 are assigned to the receivers through a classical greedy algorithm where we begin to assign the most onerous guests to the lowest loaded hosts. (It is worth to observe that in actual cloud architectures there are other architectural and application constraints that should be satisfied in the guest migration phase. These constraints limit the combinations of possible assignments to different sets thus reducing the computational cost of sorting.)

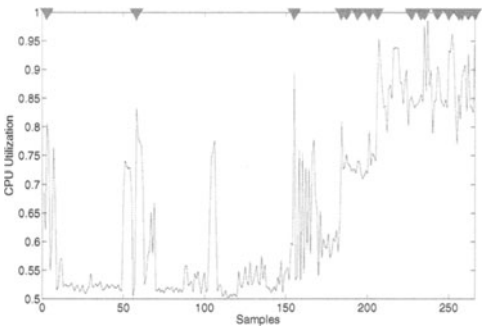
The most innovative contribution of this paper is on the first two phases that represent the core of the following two sections. In the other two phases, we adopt more traditional algorithms not deserving an accurate treatment in this paper.

4 Selection of Sender Hosts

The identification of the set of sender hosts represents the most critical problem for the dynamic management of a cloud architecture characterized by thousands of machines. The fundamental idea to determine selective and robust detections is to pass from more or less sophisticated threshold-based models, that consider the amount of load of a host in a certain interval, to a model that analyzes the load profile of the resources. The goal is to signal only the hosts subject to *significant state changes* of their load, where we define a state change *significant* if it



(a) Profile 1



(b) Profile 2

Fig. 2. CPU load in two hosts (each sample refers to an interval of 1 minute)

is intensive and persistent. To determine abrupt modifications of a host load profile, we propose a reliable and robust detection model especially useful when the application context consists of large numbers of hosts subject to: many instantaneous spikes, non-stationary effects, and unpredictable and rapidly changing load.

As examples, Figure 2(a) and Figure 2(b) show two typical profiles of the CPU utilization of two hosts in a cloud architecture. The former profile is characterized by a stable load with some spikes but there is no *significant state change* in terms of the previous definition. On the other hand, the latter profile is characterized by some spikes and by two significant state changes around sample 180 and sample 220. A robust detection model should arise no alarm in the former case, and just two alarms in the latter instance. In a similar scenario, it is clear that any detection algorithm that takes into consideration an absolute or average load value as alarm mechanism tends to cause many false alarms. This is the case of threshold-based algorithms [4,11] that are widely adopted in several management contexts. Just to give an example, let us set the load threshold to define a sender host to 0.8 of its CPU utilization (done for example in [15]). In the Figures 2, the small triangles on the top of the two figures denote the checkpoints where

Table 1. Evaluation of ARL

h	1	2	3	4	5	6	7	8	9	10
ARL_0	6	20	59	169	469	1286	3510	9556	25993	70674
ARL_1	2	4	6	8	10	12	14	16	18	20

the threshold-based detection algorithm signals the host as a sender. There are 10 signals in the former case and 17 in the latter case instead of the expected 0 and 2. This initial result denotes a clear problem with a critical consequence on performance: we have an excessive number of guest migrations even when not strictly necessary. If we extend this example to a cloud context characterized by thousands of hosts, then we can understand why dynamic guest migration is not yet so popular.

Our detection model takes a quite different approach that evaluates the entire load profile of a resource and aims to detect abrupt and permanent load changes. To this purpose, we consider a stochastic model based on the CUSUM (Cumulative Sum) algorithm [1] that works well even at runtime. Other anomaly detection techniques based on pattern matching and data mining are preferable for off-line approaches.

The CUSUM algorithm has been shown to be optimal in that it guarantees minimum mean delay to detection in the asymptotic regime when the mean time between false alarms goes to infinity [2]. We consider the one-sided version of the CUSUM algorithm that is able of selecting increasing changes of the load profile in face of variable and non-stationary characteristics. The samples of the loads deriving from the host monitors denote a time series $\{y_i\}$, $i = 1, \ldots, n$, characterized by a target value $\widehat{\mu}_i$ that is computed as the exponentially weighted average of prior data:

$$\widehat{\mu}_i = \alpha y_i + (1 - \alpha)\widehat{\mu}_{i-1} \tag{1}$$

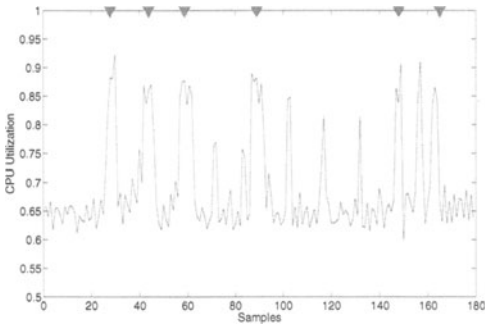
where $0 < \alpha \leq 1$ is typically set to $1/(1 + 2\pi * f)$, and f is the cutoff frequency of the EWMA model [3]. The CUSUM algorithm detects abrupt increases from the target value $\widehat{\mu}_i$ by evaluating the following test statistics:

$$d_0 = 0; \quad d_i = \max\{0, d_{i-1} + y_i - (\widehat{\mu}_i + K)\} \tag{2}$$

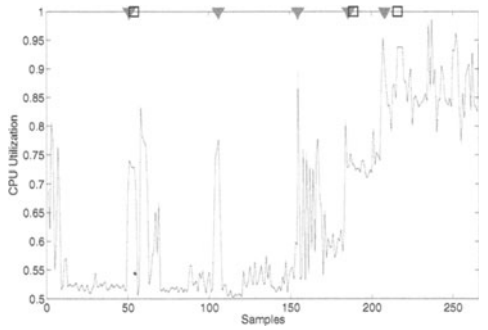
which measures positive deviations from a nominal value $\widehat{\mu}_i$. A counter d_i accumulates all deviations of the measures y_i from the target value $\widehat{\mu}_i$ that are greater than a pre-defined constant K; the counter d_i is reset to 0 when they become negative. The term K, which is known as the allowance or slack value, determines the minimum deviation that the statistics d_i should account for. The suggested default value in literature is $K = \frac{\Delta}{2}$, where Δ is the minimum shift to be detected [2]. A change in the load profile of a host is signaled when d_i exceeds $H = h\sigma_y$, where h is a design parameter and σ_y is the standard deviation of the observed time series.

The choice of the parameter h influences the performance of the CUSUM algorithm in terms of the so called *Average Run Lengths* (ARL), where ARL_0

denotes the average number of samples between false alarms when no significant change has occurred in the load, and ARL_1 denotes the average number of samples to detect a significant change when it does occur. Ideally, ARL_0 should be very large because we want to limit false alarms, while ARL_1 should be rather small because we do not want an excessive delay to signal a significant load change. We know and show in Table 1 that both ARL_0 and ARL_1 tend to grow for increasing values of h, although ARL_0 shows an exponential increment, and ARL_1 a linear increment as a function of h. Hence, the best choice of h is a compromise because too large values would improve ARL_0 but would deteriorate ARL_1 performance. As the reference value proposed in literature [2] is $h = 5$, we initially consider the so called *Baseline CUSUM* having $H = 5\sigma_y$. The performance of this algorithm is shown in Figures 3, where each small triangle denotes a point in which a host is signaled as a sender. If we compare the results in Figure 3 with those in Figure 2 (referring to a threshold-based algorithm), we can appreciate that the total number of detections is significantly reduced because it passes from 27 to 11. In particular, the Baseline CUSUM is able to avoid detections due to load oscillations around the threshold value. On the other hand, it is unable to address completely the issue of unnecessary



(a) Profile 1



(b) Profile 2

Fig. 3. Baseline and Selective CUSUM models

detections related to short-time spikes, such as those occurring at samples 30, 45, 55 and 90 in Figure 3(a).

To have even a more robust and selective detection algorithm suitable for cloud contexts, we propose a modified version of the Baseline CUSUM model, namely *Selective CUSUM*, that chooses h with the goal of maximizing ARL_0 under some temporal constraints X related to the average delay necessary to signal a significant load change. From this temporal constraint X, that is expressed in terms of samples and ARL_1, we can select the upper bound for h by referring to the Table 1. This is not the best value for X because the choice always depends on the application context. For example, if in our platform we consider that a maximum acceptable delay for detecting a significant load change is around 15 minutes, by considering that samples are taken every minute, we have that $X = 15$. From Table 1, we can easily get that a value of $h \in [7, 8]$ exhibits an $ARL_1 \sim 15$. Hence, a good choice for characterizing the Selective CUSUM is to set $h = 7$.

In Figures 3, the three small boxes on the top denote the activations signaled by the Selective CUSUM. We can appreciate that this algorithm determines robust and selective detections of the sender hosts: indeed, it is able to remove any undesired signal caused by instantaneous spikes in Figure 3(a), and to detect only the most significant state changes at samples 55, 185, 210 in Figure 3(b), actually just one more (at sample 55) than the optimal selection of two signals.

5 Selection of Guests

When a host is selected as a sender, it is important to determine which of its guests should migrate to another host. As migration is expensive, our idea is to select few guests that have contributed to the significant load change of their host. For each host, we apply the following three steps:

1. evaluation of the load of each guest;
2. sorting of the guests depending on their loads;
3. choice of the subset of guests that are on top of the list.

The first step is the most critical, because we have several alternatives to denote the load of a guest. Let us consider for example the CPU utilization of five virtual machines (A-E) in Figure 4 obtained by the VMware monitor.

The typical approach of considering the CPU utilization at a given sample as representative of a guest load (e.g., [4, 11]) is not a robust choice here because the load profiles of most guests are subject to spikes. For example, if we consider samples 50, 62, 160, 300 and 351, the highest load is shown by the guest B, albeit these values are outliers of the typical load profile of this guest. Even considering as a representative value of the guest load the average of the past values may bring us to false conclusions. For example, if we observe the guests at sample 260, the heaviest guest would be A followed by E. This choice is certainly preferable to a representation based on absolute values, but it does not take into account an important factor of the load profiles: the load of the guest E is rapidly decreasing while that of the guest A is continuously increasing.

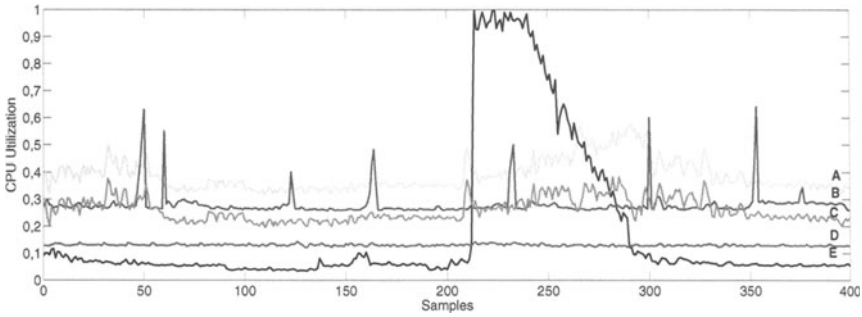


Fig. 4. Profiles of guest machines

Our idea is that a guest selection model should not consider just absolute or average values, but it should also be able to estimate the *behavioral trend* of the guest profile. The behavioral trend gives a geometric interpretation of the load behavior that adapts itself to the non stationary load and that can be utilized to evaluate whether the load state of a guest is increasing, decreasing, oscillating or stabilizing. Consequently, it is possible to generate a load representation of each guest based on the following geometric interpretation. Between every pair of the m consecutive selected points in the time series $\{y_i\}$, $i = 1, \ldots, n$, we compute the *trend coefficient* α_j, with $0 \leq j \leq m-1$, of the line that divides the consecutive points $y_{i-j|\frac{n}{m}|}$ and $y_{i-(j+1)|\frac{n}{m}|}$.

$$\alpha_j = \frac{y_{i-j|\frac{n}{m}|} - y_{i-(j+1)|\frac{n}{m}|}}{|\frac{n}{m}|}; 0 \leq j \leq m-1; i < m \tag{3}$$

In order to quantify the degree of variation of the past data values, we consider a weighted linear regression of the m trend coefficients:

$$\bar{\alpha}_i = \sum_{j=0}^{m-1} p_j \alpha_j; \quad \sum_{j=0}^{m-1} p_j = 1 \tag{4}$$

where $\alpha_0, \ldots, \alpha_{(m-1)}$ are the trend coefficients that are weighted by the p_j coefficients. This is the most general formula that can pass from not weighted p_j values to weighted coefficients obtained through some decay distributions. In this paper, we consider a geometric distribution of the weights p that gives more importance to the most recent trend coefficients. The absolute value of the j-th trend coefficient $\mid \alpha_j \mid$ identifies the intensity of the variation between two consecutive measures $y_{i-j|\frac{n}{m}|}$ and $y_{i-(j+1)|\frac{n}{m}|}$. The sign of α_j denotes the direction of the variation: a plus represents an increase between the $y_{i-j|\frac{n}{m}|}$ and $y_{i-(j+1)|\frac{n}{m}|}$ values, while a minus denotes a decrease. A load representation of the guest g at sample i-th, denoted by L_i^g (for g spanning the entire set of guests hosted by the considered physical machine), is the result of a linear combination

between the quantitative trend, $\bar{\alpha}_i$, and the actual load value, $y_0, \ldots, y_{n-1}$, that is:

$$L_i^g = \bar{\alpha}_i + \sum_{j=0}^{n-1} q_j y_{i-j}; \quad \sum_{j=0}^{n-1} q_j = 1 \tag{5}$$

After having obtained a load representation L_i^g for each guest g, we can sort them from the heaviest to the lightest. This operation is immediate because the total number of guests U running on the considered host is limited.

The third final step must determine which guest(s) should migrate to another host. We recall that the idea is to select only the guests that contribute more to the host load. To this purpose, we estimate the relative impact of the load of each guest on the overall load and we compute $\gamma_i^g = \frac{L_i^g}{\sum_{j=1}^{U} L_i^j}$ for $i = 1, ..., U$, where U is the total number of guests in the host i. As we have already sorted the guests in a decreasing order based on L_i^g values, the order is preserved when we consider the γ_i^g values. The idea is to select for migration the minimum number of guests with the highest relative loads. This is an arbitrary choice, but we found convenient to consider, as an example, the guests that contribute to one-third of the total relative load. To give an idea, let us consider two hosts H_1 and H_2 characterized by the following γ_i^g values: $(0.25, 0.21, 0.14, 0.12, 0.11, 0.10, 0.03, 0.02, 0.01)$, and $(0.41, 0.22, 0.20, 0.10, 0.04, 0.02, 0.01)$, respectively. In H_1, we select the first two guests because the sum of their relative loads 0.46 exceeds one-third. On the other hand, in H_2 we select just the first guest that alone contributes to more than one-third of the total load.

As we want to spread the migrating load to the largest number of receiver hosts, we want that no receiver should get more than one guest that is, $G = R$. Hence, we have to guarantee that the number of guests we want to migrate is $G < (N - S)$. Typically, this constraint is immediately satisfied because S is a small number, $S \ll N$, and typically $G \leq 2S$. However, if for certain really critical scenarios it results that $G > (N - S)$, we force the choice of just one guest for each sender host. This should guarantee a suitable solution because otherwise we have that $S > R$ that is, the entire cloud platform tends to be overloaded. Similar instances cannot be addressed by a dynamic migration algorithm but they should be solved through the activation of standby machines [4] that typically exist in a cloud data center. It is also worth to observe that all our experiments were solved through the method based on the one-third of the total relative load with no further intervention.

6 Conclusion

Dynamic migrations of virtual machines is becoming an interesting opportunity to allow cloud infrastructures to accommodate changing demands for different types of processing with heterogeneous workloads and time constraints. Nevertheless, there are many open issues about the most convenient choice about when to activate migration, how to select guest machines to be migrated, and the most convenient destinations. These classical problems are even more severe

in a cloud context characterized by a very large number of hosts. We propose novel algorithms and models that are able to identify just the real critical host and guest devices, by considering the load profile of hosts and the load trend behavior of the guest instead of thresholds, instantaneous or average measures that are typically used in literature.

Experimental studies based on traces coming from a cloud platform supporting heterogeneous applications on Linux and MS virtualized servers show significant improvements in terms of selectivity and robustness of the proposed algorithm for sender detection and selection of the most critical guests. These satisfactory results are encouraging us to integrate the proposed models and algorithms in a software package for dynamic management of virtual machines in cloud architectures. On the other hand, we should consider that a cloud architecture consists of heterogeneous infrastructures and platforms, guests that must not migrate or that can migrate only within certain subsets of hardware and operating systems. These real constraints are not taken into account in this paper, but we are working to include them in a future work.

References

1. Page, E.S.: Estimating the point of change in a continuous process. Biometrika 44 (1957)
2. Montgomery, D.C.: Introduction to Statistical Quality Control
3. Kendall, M., Ord, J.: Time Series. Oxford University Press, Oxford (1990)
4. Khanna, G., Beaty, K., Kar, G., Kochut, A.: Application Performance Management in Virtualized Server Environments. In: Proc. of Network Operations and Management Symp. (2006)
5. Stage, A., Setzer, T.: Network-aware migration control and scheduling of differentiated virtual machine workloads. In: Proc. of 31st Int. Conf. on Software Engineering (2009)
6. Clark, C., Fraser, K., Steven, H., Gorm Hansen, J., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live Migration of Virtual Machines. In: Proc. of the 2nd ACM/USENIX Symp. on Networked Systems Design and Implementation (2005)
7. Travostino, F., Daspit, P., Gommans, L., Jog, C., de Laat, C., Mambretti, J., Monga, I., Van Oudenaarde, B., Raghunath, S., Wang, P.Y.: Seamless live migration of virtual machines over the MAN/WAN. Future Gener. Computer System 22(8) (2006)
8. DeFanti, T., de Laat, C., Mambretti, J., Neggers, K., St. Arnaud, B.: TransLight: a global-scale LambdaGrid for e-science. Communications of the ACM (2003)
9. Hines, M.R., Gopalan, K.: Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In: Proc. of the ACM SIGPLAN/ SIGOPS Int. Conf. on Virtual execution environments (2009)
10. Wei, H., Qi, G., Jiuxing, L., Panda, D.K.: High performance virtual machine migration with RDMA over modern interconnects. In: Proc. of the IEEE Int. Conf. on Cluster Computing (2007)
11. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Black-box and Gray-box Strategies for Virtual Machine Migration. In: Proc. of the 4th USENIX Symp. On Networked Systems Design and Implementation (2007)

12. Bobroff, N., Kochut, A., Beaty, K.: Dynamic Placement of Virtual Machines for Managing SLA Violations. In: Proc. of the 10th IFIP/IEEE International Symp. On Integrated Network Management (2007)
13. Hermenier, F., Lorca, X., Menaud, J.-M., Muller, G., Lawall, J.: Entropy: a Consolidation Manager for Cluster. In: Proc. of the Int. Conf. on Virtual Execution Environments (2009)
14. Nguyen Van, H., Dang Tran, F.: Autonomic virtual resource management for service hosting platforms. In: Proc. of the Workshop on Software Engineering Challenges in Cloud Computing (2009)
15. VMware Distributed Power Management Concepts and Use