Annals of Operations Research manuscript No.
(will be inserted by the editor)

Matching Performance Objectives for Open and Closed Workloads by Consolidation and Replication

Davide Cerotti · **Marco Gribaudo** ·
Pietro Piazzolla · **Giuseppe Serazzi**

Received: date / Accepted: date

Abstract One of the most crucial task during the design of a computing infrastructure is the decision about the proper amount of equipments required to handle a specific workload while satisfying a set of performance objectives. This problem is emphasized even more in actual computer infrastructure such as clouds, where the user can provision the resources very easily thanks to the use of virtual machines. If the system has to handle a low workload, resources can be consolidated together to reduce the costs. If however the workload is very high, resources must be replicated to gain an acceptable service level. In this paper we derive the impact on several performance indexes for both consolidation and replication when considering both open and closed workloads. In particular, we present an analytical model to determine the best consolidation or replication options that match given performance objectives specified through a set of constraints. Depending on the particular type of workload and constraints, we present either closed form expressions, heuristics or an iterative algorithm to compute the minimum number of resources required.

Keywords Consolidation and replication · Open and closed workload · Cloud computing and virtualization · Analytical techniques

1 Introduction

Consolidation and replication techniques are commonly used to manage efficiently large datacenters. According to the former technique, the load of several systems are merged in a reduced number of servers minimizing operational costs. The latter technique partition the load among several physical machines executing replicated applications: in this way, the requests flow each server has to handle is reduced and the performance improved.
Both these techniques have several positive aspects but they may also lead to complex management and technical problems that require wide knowledge in several computer science topics to be satisfactorily solved.
While the introduction of virtualization concept alleviated some of the difficulties related to

Dip. di Elettronica e Informazione, Politecnico di Milano,
via Ponzio 34/5, 20133 Milano, Italy
E-mail: {cerotti, gribaudo, piazzolla, serazzi}@elet.polimi.it

the management of large infrastructures (see, e.g., VirtualBox (2013); VMware (2013)), it also increased the logical distance between the users and the physical resources making more complex the performance forecast. This problem is particularly evident in virtual environments, such as clouds, where users have a limited or no control of the hardware allocated to execute their requests. These drawbacks, coupled with the heterogeneity of the actual workload service demands Ganapathi et al (2010) and the variability of arrival patterns make from a user perspective the matching of its performance expectations a very difficult task.
This paper extends the results proposed in Gribaudo et al (2012), exploring the relationships between the servers consolidation/replication actions and the performance experienced by users in systems running mixes of different classes of applications when dealing with both open and closed workloads. Indeed, these actions play a fundamental role in determining the overall performance since they have a direct impact on the bottleneck creation and migration.
In the considered infrastructure the subjects of consolidation and replication actions are Virtual Machines (VMs) that users may startup or shutdown. Users provision VMs in a quantity assumed sufficient to satisfy their requirements. The number of instanced VMs has a strong impact on the performance experienced. Under-provisioning will provide unsatisfactory performance, that may lead to violating its expectations, while over-provisioning will result in a waste of money.
We will focus on the forecast of performance resulting from consolidation and replication actions from a user perspective. In particular, we present a technique that allows to determine the optimal consolidation or replication actions to match a user performance objectives subject to a number of constraints. When possible, the technique uses closed form expression to determine the best possible resource allocation. If the considered performance objectives do not allow a unique closed form expression, either heuristics or an iterative procedure are proposed. The latter technique allows to take into account closed workloads, generated for example by batch elaboration processes.

The structure of the paper is as follows. In the next section we present other works aiming to the allocation of resources in virtual environments. In Section 3 we present the main definitions used throughout the paper and the performance indexes considered. In Section 4 we derive the minimum number of replications needed to handle a given multiclass workload, and we extend the methodology to deal with performance constraints. In Section 5 we propose an iterative algorithm to find a minimum replicas solution to satisfy a given set of PCs. Both techniques are then analysed in Section 6. Section 7 concludes the paper.

2 Related Work

In the literature, there are several works that deal with the optimal allocation of resources in virtual environments. Several techniques and models focus on database consolidation, some like in Curino et al (2011) by means of workload monitoring for load balancing, others like in Kokkinos et al (2008) using data migration and task scheduling. Other techniques, as in Benevenuto et al (2006); Bennani and Menascé (2005); Khanna et al (2006) are aimed to maintain acceptable application performance levels while minimizing the costs of migration/consolidation of resources. Many works propose different approaches to enable autonomic controller to satisfy service level objectives by dynamically provisioning resources, as Bushehrian (2011); Padala et al (2007); Watson et al (2010). In particular, in Bobroff et al (2007); Menascé (2005) the dynamic allocation of VMs in cloud environment is described.

A probabilistic approach to enable autonomic controllers is proposed in Watson et al (2010). Different techniques and models for the monitoring of workload to provide load balancing and database deployment consolidation are described in Curino et al (2011). Data consolidation in grid networks using data migration and task scheduling is analyzed in Kokkinos et al (2008). In Khanna et al (2006) heuristics are proposed to minimize the costs of migration/consolidation and maintain acceptable application performance levels. In Benevenuto et al (2006) several models are developed to predict the performance of applications running on virtual servers consolidated on few physical machines. In Padala et al (2007) the problem of dynamically control resource allocation to individual components of multi-tier enterprise application in a shared hosting environment is addressed. Analytical queueing network models combined with combinatorial search techniques is described in Bennani and Menascé (2005) to dynamically redeploy resources to the various applications of a datacenter. The dynamic allocation of VMs in cloud environment is described in Bobroff et al (2007) Menascé (2005). In Bushehrian (2011) exploits bin packing and time series forecasting to minimize the number of physical machines required to support a workload. Virtual Machine Manager allocation policies is described in Ongaro et al (2008). Managing the physical resources to correctly allocate them among the different Virtual Environments is discussed in Menascé and Bennani (2006).

The technique proposed in this paper is different from the previous one, since we study the impact of consolidation/replication actions on performance indexes subject to constraints and we consider VMs executing concurrently applications having heterogeneous service demands, i.e., running a multiclass workload. Also, the suggested approach to the identification of the optimal number of VMs that satisfy performance objectives is proactive while the approaches proposed in literature are reactive.

3 System Model

To analyse the consolidation/replication effects on performance indexes we will use the queuing network theory. Consider a system with a multiclass workload composed by M servers and C customer classes. Initially in Sections 3 and 4, each server will be modeled by a single station of the queeuing network. Such assumption is suitable to represent services which strongly impact on a single resource of the system (e.g. a CPU-bound applications). In such a way, the behavior of a server can be characterized by a single value of the demand. More formally, the mean service demand of a class c (with $1 \leq c \leq C$) job at server m (with $1 \leq m \leq M$) is defined as the product of the mean service time of a class c job for each visit to server m and the mean number of visits by a class c job to server m, and it is denoted by a $M \times C$ matrix D whose element d_{mc} represents the service demand that a class c job requires from the m-th server, i.e., the mean time required by server c to its complete execution. In Section 5 we will relax such assumption modeling a single server as a set of stations where each station represents a different resource (e.g. CPU or disk) of the server. In such a case to characterize the server behavior, different values of the demand are needed (i.e. one for each resource).

The considered workload can be either open or closed. In an open model, the jobs arrive to the system according to Poisson processes. In particular, we call λ_c the arrival rate of class c jobs to the system, and we define the total arrival rate as $\Lambda = \sum_{c=1}^{C} \lambda_c$. In a closed model, there is a fixed number of jobs for each class that circulates inside the system. Let us call N_c the number of class c jobs in the system, and let us denote with $N = \sum_{c=1}^{C} N_c$ the total population of the model. In both cases, each class c can provide a different contribution to the total load of the system: we can measure the individual contribution of each class with a vector $\boldsymbol{\beta} = |\beta_1 \ldots \beta_C|$, referred to as the *population mix*. Each term β_c accounts for the fraction of the total workload of the system due to class c jobs. The definition of β_c is different for open and closed models: in the first case we have $\beta_c = \frac{\lambda_c}{\Lambda}$, and in the second we have $\beta_c = \frac{N_c}{N}$. As further assumptions, we exclude system with mixed-workload (e.g. systems where some classes have an open workload, while some others are closed), and we do not allow *class switching* (that is the possibility for a job to change its class during its service).

We assume that servers can be either *consolidated* or *replicated*. Two servers are consolidated when they are implemented as two different VMs on the same physical system. To simplify the presentation, we consider that the servers that are consolidated in a single physical machine are the ones of indexes $M-1$ and M. We assume that the effects of server consolidation is the sum of the service demands of the two servers. With this assumption, the matrix $\mathbf{D}^{\#}$ resulting from the consolidation of servers $M-1$ and M has $M-1$ rows; row $M-1$ represents the consolidated server:

$$\begin{array}{ll} d^{\#}_{m,c} = d_{m,c} & \forall c \in \{1,2,...,C\}, \forall m \in \{1,2,...,M-2\} \\ d^{\#}_{M-1,c} = d_{M-1,c} + d_{M,c} & \forall c \in \{1,2,...,C\} \end{array} \tag{1}$$

With the replication technique a service is deployed through several physical machines reducing the workload each server has to handle. We assume to have k_m instances for each server m with $k_m \geq 1, \forall m \in \{1,2,...,M\}$. Each instance of a server m is a replica of such server implemented as a single VM running on a physical machine. In the following we will use the term instance and replica with the same meaning and, to simplify the notation, we will denote a particular configuration of instances simply as $k_1 - k_2 - \ldots - k_m$. Let k be the total number of instances that the system with replications will have $k = \sum_{m=1}^{M} k_m$, we have that $k \geq M$. We assume that traffic is equally shared among the k_m instances of the m-th server. Thus, the service demand matrix of a system with replications depends on the configuration of instances $k_1 - k_2 - \ldots - k_m$. In particular, it is described by a matrix $\mathbf{D}_{k_1-k_2-\ldots-k_m}$ with k rows and C columns where the subscript identifies the configuration of instances. Rows are partitioned in M groups, each of them composed by k_m identical rows, corresponding to the k_m instances of the m-th server. The demand associated to each row in a group can be derived from $\mathbf{D}$, by considering that each instance of the server has the same demand as the original model, with a visit ratio equal to $\frac{1}{k_m}$. In particular, we can define:

$$\mathbf{D}_{k_1-k_2-\ldots-k_m} = \left| k \left\{ \begin{array}{l} k_1 \left\{ \begin{array}{ccc} \frac{d_{11}}{k_1} & \cdots & \frac{d_{1C}}{k_1} \\ & \vdots & \\ \frac{d_{11}}{k_1} & \cdots & \frac{d_{1C}}{k_1} \end{array} \right. \\ \vdots \\ k_M \left\{ \begin{array}{ccc} \frac{d_{M1}}{k_M} & \cdots & \frac{d_{M1}}{k_M} \\ & \vdots & \\ \frac{d_{M1}}{k_M} & \cdots & \frac{d_{MC}}{k_M} \end{array} \right. \end{array} \right. \right| \tag{2}$$

The purpose of consolidation is to reduce the number of physical machines required to handle workloads characterized by very low demands. Replication on the other hand allows to share requests among several machines to handle very high workloads. It is based on

Table 1 Performance indexes and constraints

Index	Description	Threshold
R_{mc}	Residence time at resource *m* of a given class *c*	ρ_{mc}
$R_m = \sum_c R_{mc}$	Aggregated residence time at a given resource *m*	ρ_m
$R_c = \sum_m R_{mc}$	System response time of a class *c*	ρ_c
$R = \sum_m \sum_c R_{mc}$	System response time	ρ
u_{mc}	Utilization at a resource *m* of a given class *c*	σ_{mc}
u_m	Aggregated utilization at a given resource *m*	σ_m

the assumption that a load balancer can equally share the demands among the replicated servers: of course if the load is extremely high, the load balancer becomes the bottleneck of the system, and it must be replicated as well.

3.1 Performance indexes and constraints

Several *performance constraints* (PCs) can be defined by a user in order to match his own expectations or objectives, in this paper we will focus on requirements concerning: the utilization u and the mean response time R. Both types of indexes can be computed at different level of granularity of the model, i.e. for each instance of a server, for each server, or for the whole model. Notice that the values of a performance index of each instance of a given server are the same, because the replication shares uniformly the workload among them. Thus, the residence time of a server is the sum of the residence times over all its instances, or, equivalently, the product of the residence time of one instance and the number of instances. The utilization of a server is equal to the utilization of one of its instances, or, equivalently, the mean utilization over all its instances. As usual, the system response time is the sum of the residence times at all servers. Moreover, both response time and utilization can be computed for a specific workload class, or aggregated over all classes.

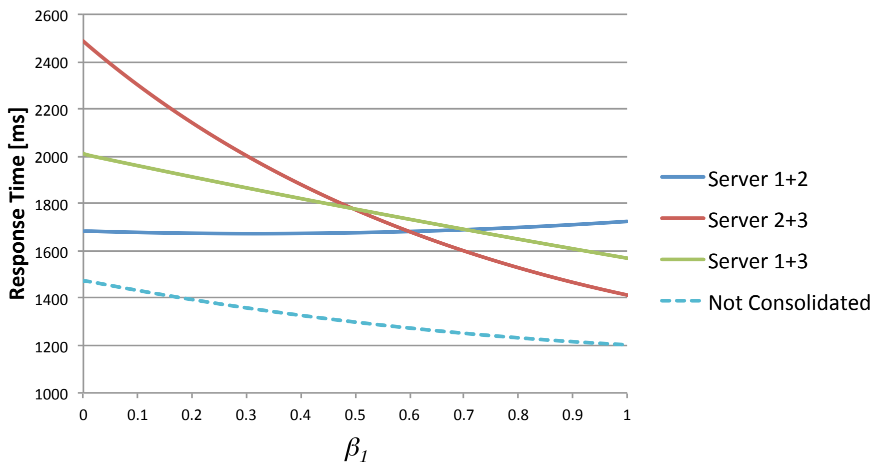
We will use the subscript '*mc*' to denote an index at resource m of class c (R_{mc}, u_{mc}), the subscript '*m*' for index at a resource m aggregated over all classes (R_m, u_m), and '*c*' for index of a given class c of the whole system (R_c). Analogous subscripts are used for the thresholds needed to define the PCs. Table 1 summarizes the performance indexes and corresponding thresholds analyzed in the paper.

3.2 Example of consolidation/replication actions

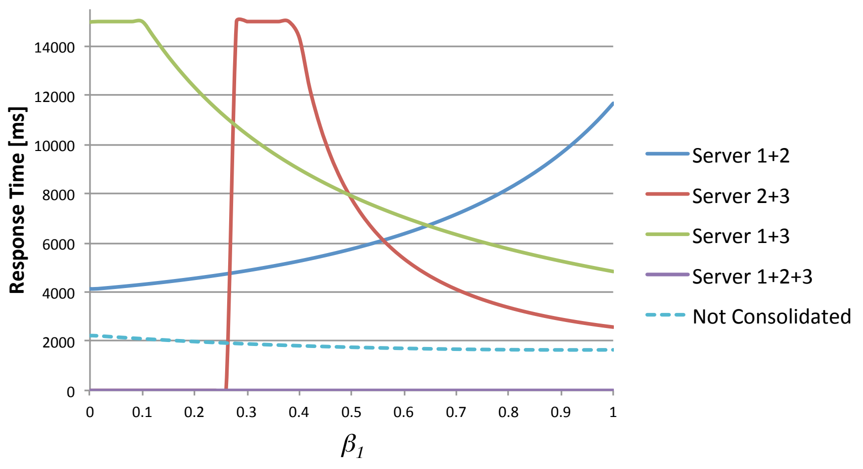
Let us consider a system with a multiclass open workload composed of $C = 2$ classes of customers, and $M = 3$ servers. The service demand matrix $\boldsymbol{D}$ in milliseconds is:

$$\boldsymbol{D} = \begin{vmatrix} 391 & 238 \\ 281 & 346 \\ 223 & 450 \end{vmatrix} \tag{3}$$

Depending on the population mix $\boldsymbol{\beta}$, server 1 or server 3 can be saturated. The system performs a bottleneck switch at $\beta_1 = 0.5579$ (see Balbo and Serazzi (1996) for the computation of the bottleneck switching point). This means that with $0 \leq \beta_1 < 0.5579$, server 3 is the bottleneck of the system, otherwise the bottleneck is server 1. Using standard queueing theory results (see e.g.,Lazowska et al (1984); Jackson (1963)), we plot the system response time as function of the population mix $\boldsymbol{\beta} = |\beta_1 \;\; (1-\beta_1)|$ for different arrival rate Λ and different consolidation (Figure 1) or replication (Figure 2) patterns. The actual demand matrices, obtained after consolidation and replication are shown in Figure 1(c) and 2(c), respectively.



a)

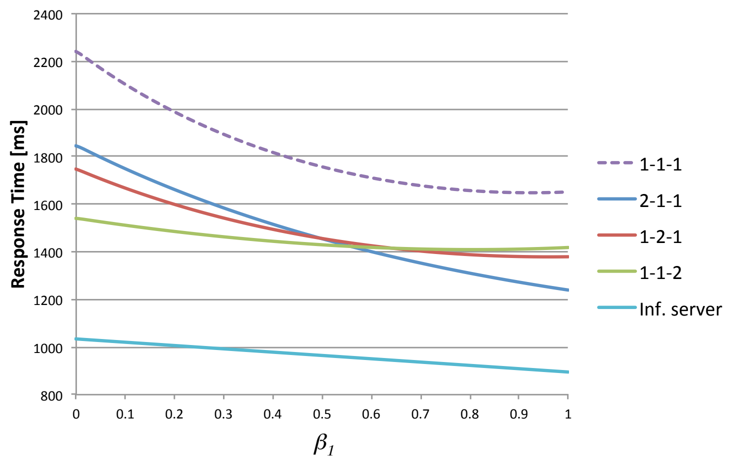


b)

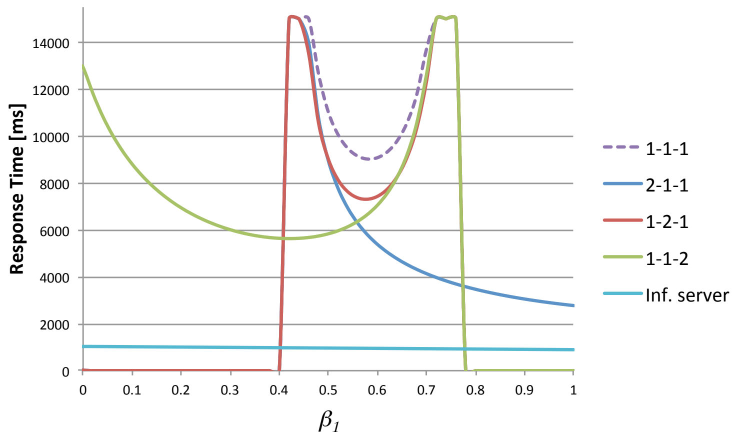
Consolidations of servers							
Server 1+2		Server 2+3		Server 1+3		Server 1+2+3	
672	584	391	238	614	688	895	1034
223	450	504	796	281	346		

c)

Fig. 1 System response time in *msec* as function population mix β_1 of various consolidation configurations of the system described by the demand matrix $\boldsymbol{D}$ presented in Eq. 3 for different arrival rate: (a) $\Lambda = 0.0008$ *jobs*/*msec*; (b) $\Lambda = 0.0014$ *jobs*/*msec*; (c) service demands of the system. NOTE: in (b) system response time is set to zero in case of unstable conditions and it is capped at 15000*msec* in order to avoid out-of-scale values.



a)



b)

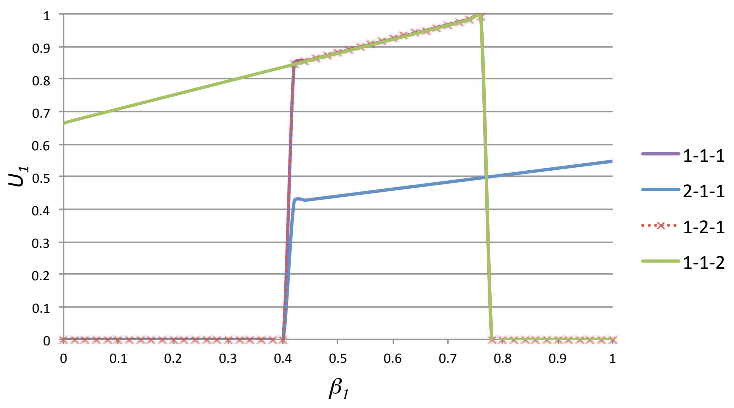
Replication of servers					
2-1-1		1-2-1		1-1-2	
195.5	119	391	238	391	238
195.5	119	140.5	173	281	346
281	346	140.5	173	111.5	225
223	450	223	450	111.5	225

c)

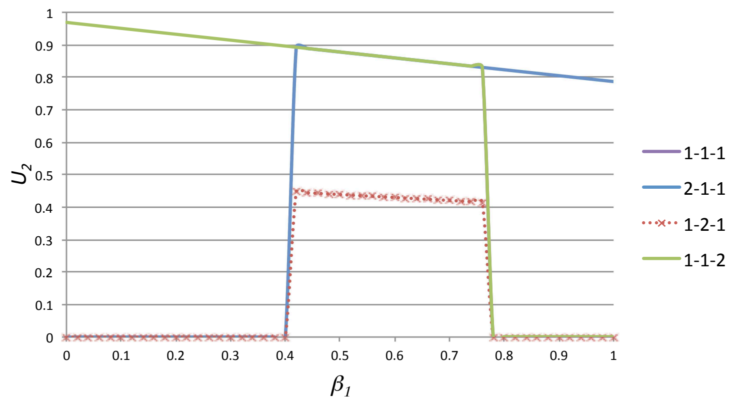
Fig. 2 System response time in *msec* as function population mix β_1 of various replication configurations of the system described by the demand matrix $\boldsymbol{D}$ presented in Eq. 3 for different arrival rate: (a) $\Lambda = 0.0014$ *jobs/msec*; (b) $\Lambda = 0.0028$ *jobs/msec*; (c) service demands for the system. NOTE: in (b) system response time is set to zero in case of unstable conditions and it is capped at 15000*msec* in order to avoid out-of-scale values.

In both Figures 1 and 2 the values of system response time are set to zero when the system is unstable, thus a zero value does not correspond to a response time equal zero, but an undefined value due to an unstable situation. Moreover, the response times are capped at 1500 *msec*. For instance in Figures 1(b) when server 2 and 3 are consolidated, the system is unstable for $\beta \in [0, 0.23]$ and the system provides a response time greater than 1500 *msec* for $\beta \in [0.23, 0.4]$. Figure 1(a) shows the effect of consolidation when the system is lightly utilized, i.e., the global arrival rate is low with respect to the maximum load that the system can handle. This system is stable for all the different population mixes β, and the response time increases with the number of server consolidated on the same physical machine because no workload partitioning can be applied, which is natural when the server runs on separate hardware. It is interesting to see that the choice of the particular consolidation pattern affects the performance, and that the best choice is function of the population mix. When class 1 jobs are dominant, i.e., ($\beta_1 \approx 1$), consolidating server 2 and 3 provides the best results in terms of response time, with respect of the consolidation of the other servers, e.g., 1 and 3 or 1 and 2 or 1,2,3. This behavior is emphasized in Figure 1(b) where the system is unstable when all the servers are consolidated in a single physical resource, and cannot be stable for $\beta < 0.23$ when server 2 and 3 are consolidated. Indeed, the best choice is always to consolidate the machines that are not bottleneck for a particular population mix $\boldsymbol{\beta}$.
Replication on the other hand reduces the response time. As shown in Figure 2(a), the best choice, again depending on the population mix β, corresponds to the replication of the bottleneck server. Figure 2(b) shows the same replication scheme when the system is very heavily loaded, i.e., the global arrival rate is close to the maximum load that the system can handle. In this case replication can make stable a system otherwise unstable. It is also interesting to see that the replication of server 2, the one that is never a bottleneck, has the effect of reducing the response time when $\beta \in [0.4, 0.8]$, but it does not extend the stability region of the system which remains the same as the one of the non-replicated case. The response time of a replicated system has a lower bound, which can be computed by considering all the resources as infinite server resources. The minimum response time of the infinite server case is also shown in Figure 2(a) and (b) to emphasize the difference between the obtained response time and its lower bound.

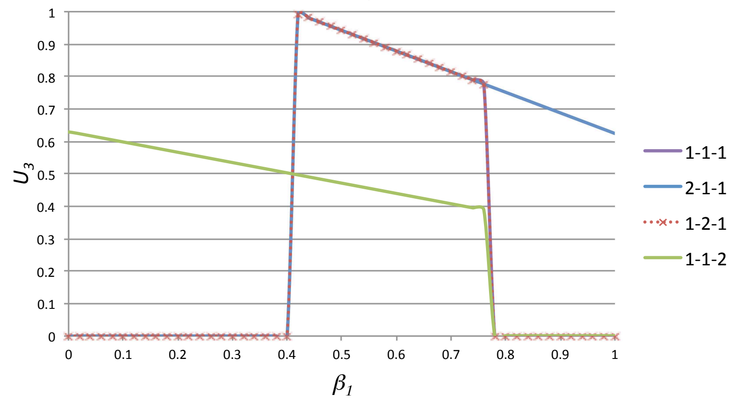
Finally, we investigate the effect of the replication pattern on the aggregated utilizations of each server. In Figure 3 we plot the aggregated utilizations as function of the population mix $\boldsymbol{\beta} = |\beta_1 \quad (1-\beta_1)|$ for different replication patterns and a fixed arrival rate $\Lambda = 0.0028$ *jobs/msec*. We keep the same values of demand (see Figure 2(c)) of the previous analysis. Each graph shows the effect of the different replication patterns on the aggregated utilization of a specific server. The utilization behavior of the different server for the non-replicated configuration confirms that the bottleneck server depends on the population mix. Indeed for $\beta \in [0, 0.5579]$ the utilization of server 3 (Figure 3(c)) is greater that the other servers reaching saturation at $\beta = 0.4$, whereas for $\beta \in [0.5579, 1]$ server 1 is the bottleneck saturating at $\beta = 0.8$ (Figure 3(a)), thus the stability interval without replicas is $[0.4, 0.8]$. As expected, replication of a server halves its utilization, furthermore replication of a bottleneck server has the effect of enlarge the stability interval. Replication of server 1 enlarges it to $[0.4, 1]$, and symmetrically replication of server 3 to $[0, 0.8]$. Instead, as previously stated, replication of the non-bottleneck server 2 does not change the saturation conditions.



a)



b)



c)

Fig. 3 Aggregated utilizations as function of population mix β_1 for various replication configurations of the system described by the demand matrix $\boldsymbol{D}$ presented in Eq. 3. The aggregated utilization is shown for a replica of: (a) Server 1; (b) Server 2; (c) Server 3.

4 Analytical results

In this Section we provide equations to size a system with an open workloads in order to fulfill given PCs. We start investigating a scenario where all virtual machines are consolidated into the same physical server and computing the maximum manageable workload intensity and the minimum number of required virtual machines to preserve the stability of such configuration (Section 4.1). Then, we introduce in the analysis the effect of replication and determine the best allocation strategy of the replicas (Section 4.2). Finally, we extend such approach to take into account the satisfaction of performance constraints too (Section 4.3-4.6).

4.1 Computing the best number of replications

With a light load, all the servers can be consolidated in a single physical machine. In this case, the demand matrix reduces to a $1 \times C$ vector $\mathbf{D}^{\#} = |d^{\#}_{1c}|$, with each element defined as:

$$d^{\#}_{1c} = \sum_{m=1}^{M} d_{mc}, \qquad \forall c \in \{1,2,...,C\}. \tag{4}$$

The utilization of the resources is $U^{\#} = \boldsymbol{\lambda}\mathbf{D}^{\#} = \Lambda\boldsymbol{\beta}\mathbf{D}^{\#}$. Since the utilization must be ≤ 1, we can compute the maximum arrival rate that the consolidated system can handle $\Lambda^{\#}(\boldsymbol{\beta})$ as:

$$\Lambda^{\#}(\boldsymbol{\beta}) = \frac{1}{\boldsymbol{\beta}\mathbf{D}^{\#}} = \left(\sum_{c=1}^{C} \beta_c d^{\#}_{1c}\right)^{-1} \tag{5}$$

In other words, given a population mix $\boldsymbol{\beta}$, all the virtual machines can be consolidate in a single physical machine if $\Lambda < \Lambda^{\#}(\boldsymbol{\beta})$. For this reason $\Lambda^{\#}(\boldsymbol{\beta})$ will be referred as the *maximum consolidation workload*. Suppose now that we have a high workload Λ for which some server of the system must be replicated. We can prove that the theoretical minimum number of physical machines $k^{T}_{\min}(\boldsymbol{\beta})$ required to handle a workload with intensity Λ is:

$$k^{T}_{\min}(\boldsymbol{\beta}) = \left\lceil \frac{\Lambda}{\Lambda^{\#}(\boldsymbol{\beta})} \right\rceil = \lceil \Lambda\boldsymbol{\beta}\mathbf{D}^{\#} \rceil \tag{6}$$

Proof.
The minimum number of instances $k = \sum_{m=1}^{M} k_m$ required to handle a workload of intensity Λ and population mix $\boldsymbol{\beta}$ must guarantee that the utilization of all the resources is strictly less than one:

$$\sum_{c=1}^{C} \lambda_c \frac{d_{mc}}{k_m} < 1 \qquad \forall 1 \leq m \leq M. \tag{7}$$

from which we can compute k_m:

$$k_m > \sum_{c=1}^{C} \lambda_c d_{mc} = \Lambda \sum_{c=1}^{C} \beta_c d_{mc}. \tag{8}$$

If we apply the definition of k we obtain:

$$k = \sum_{m=1}^{M} k_m > \sum_{m=1}^{M} \Lambda \sum_{c=1}^{C} \beta_c d_{mc} = \Lambda \sum_{c=1}^{C} \beta_c \sum_{m=1}^{M} d_{mc} = \Lambda \sum_{c=1}^{C} \beta_c d^{\#}_{1c} = \frac{\Lambda}{\Lambda^{\#}(\boldsymbol{\beta})} \tag{9}$$

If we consider that the minimum number of physical machines should be an integer, and we round k to the closest higher integer, we obtain the definition of $k^T_{\min}(\boldsymbol{\beta})$ give in Eq. 6. The theoretical minimum requires the replication of a server that consolidates all the resources, which can be unpractical. If we require that each resource holds at most one service, then the theoretical minimum is just a lower bound to the actual minimum, which could be a little bit higher. In Section 4.2 the actual minimum will be considered.

4.2 Stability issues in open models

We want to study the system as the arrival rate increases. Let γ_m the number of instances of server m normalized with respect to the total number of instances k, we have:

$$\gamma_m = \frac{k_m}{k}, \qquad \text{with: } \sum_{m=1}^{M} \gamma_m = 1. \tag{10}$$

Let $\boldsymbol{\gamma} = |\gamma_1 \dots \gamma_M|$ be the vector representing the *instances mix*. As we have seen, in order to maintain the system stable, the number of instances must grow accordingly to the increased arrival rate. In particular, reversing the definition of $\Lambda^\#(\boldsymbol{\beta})$ given in Eq. 5, we may express the total arrival rate as a function of k (the total number of instances):

$$\Lambda = k\Lambda^\#(\boldsymbol{\beta}). \tag{11}$$

We can thus define the *stability condition*, that is the condition of the system in which the utilization of all the resources should be strictly less than one:

$$\max_m \left\{ \sum_{c=1}^{C} \frac{\Lambda \beta_c d_{mc}}{k_m} \right\} = \max_m \left\{ \frac{\Lambda^\#(\boldsymbol{\beta})}{\gamma_m} \sum_{c=1}^{C} \beta_c d_{mc} \right\} < 1. \tag{12}$$

The best allocation strategy would saturate all the available physical machines, raising their utilization to 1. In other words, it will:

$$\forall m \in \{1,2,\dots,M\} : \frac{\Lambda^\#(\boldsymbol{\beta})}{\gamma_m} \sum_{c=1}^{C} \beta_c d_{mc} = 1. \tag{13}$$

From the previous Eq., we can then compute γ_m:

$$\gamma_m = \Lambda^\#(\boldsymbol{\beta}) \sum_{c=1}^{C} \beta_c d_{mc}. \tag{14}$$

It can be easily proven that the definition given in Eq. 14 is consistent with the definition of γ_m, that is that $\sum_{m=1}^{M} \gamma_m = 1$.
Proof.
If we sum the γ_m for all the resources we obtain:

$$\sum_{m=1}^{M} \gamma_m = \sum_{m=1}^{M} \Lambda^\#(\boldsymbol{\beta}) \sum_{c=1}^{C} \beta_c d_{mc} = \Lambda^\#(\boldsymbol{\beta}) \sum_{c=1}^{C} \beta_c \sum_{m=1}^{M} d_{mc} = \Lambda^\#(\boldsymbol{\beta}) \frac{1}{\Lambda^\#(\boldsymbol{\beta})} = 1 \tag{15}$$

Eq. 6 and Eq. 14 are very important, because they tell us how many virtual machines k should be provisioned, and which fraction of these machines should be used to host a particular service m, to be able to serve an input workload of intensity Λ distributed according to a given population mix $\boldsymbol{\beta}$,. In particular, inserting Eq. 6 in 14, we can obtain:

$$k_m = \lceil k\gamma_m \rceil = \left\lceil \Lambda \sum_{c=1}^{C} \beta_c d_{mc} \right\rceil . \tag{16}$$

We can use the results from Eq. 16 to compute the actual minimum number of physical machines $k^A_{\min}(\boldsymbol{\beta})$ required to handle a workload Λ as:

$$k^A_{\min}(\boldsymbol{\beta}) = \sum_{m=1}^{M} k_m \tag{17}$$

Note that by definition, we have that $k^T_{\min}(\boldsymbol{\beta}) \leq k^A_{\min}(\boldsymbol{\beta})$, but the relative difference between $k^T_{\min}(\boldsymbol{\beta})$ and $k^A_{\min}(\boldsymbol{\beta})$ tends to 0 as Λ tends to infinity.

4.3 Constraints on the utilization of a class in a resource

In Eq. 12, the parameters k and $\boldsymbol{\gamma}$ were compute to make the system stable. If instead of saturating all the resource, we want to limit the utilization of the class c at station m to a value $0 \leq \sigma_{mc} < 1$ (with $\sum_{c=1}^{C} u_{mc} \leq 1, \forall m$), Eq. 12 becomes:

$$\frac{\Lambda^{\#}(\boldsymbol{\beta})}{\gamma_m} \beta_c d_{mc} < \sigma_{mc}. \tag{18}$$

Eq. 18 should be valid for all the classes c. We can thus find the minimum value of γ_m that satisfy the PCs on the utilization for all the classes as:

$$\gamma_m = \max_c \left\{ \frac{\Lambda^{\#}(\boldsymbol{\beta})}{\sigma_{mc}} \beta_c d^{mc} \right\} \tag{19}$$

In this case however, we can have that $\sum_{m=1}^{M} \gamma_m > 1$. The number of replicas k_m for the m-th resource can be computed exactly as in Eq. 16:

$$k_m = \lceil k\gamma_m \rceil = \left\lceil \frac{\Lambda \gamma_m}{\Lambda^{\#}(\boldsymbol{\beta})} \right\rceil . \tag{20}$$

The minimum number of instances k that respect the PCs, $k^{\text{PCs}}_{\min}(\boldsymbol{\beta})$, can thus be computed as follows:

$$k^{\text{PCs}}_{\min}(\boldsymbol{\beta}) = \sum_{m=1}^{M} \left\lceil \frac{\Lambda \gamma_m}{\Lambda^{\#}(\boldsymbol{\beta})} \right\rceil \tag{21}$$

4.4 Constraints on the total utilization of a resource

Suppose instead that we want to limit the total utilization of a resource m to be at most σ_m, and to be equally shared among the classes. In this case we will have that:

$$\frac{\Lambda^{\#}(\boldsymbol{\beta})}{\gamma_m} \sum_{c=1}^{C} \beta_c d_{mc} < \sigma_m, \tag{22}$$

from which we can easily determine γ_m:

$$\gamma_m = \frac{\Lambda^{\#}(\boldsymbol{\beta})}{\sigma_m} \sum_{c=1}^{C} \beta_c d_{mc}. \tag{23}$$

The same considerations given in Section 4.3 about the possibility of having $\sum_{m=1}^{M} \gamma_m > 1$ and its implications are also valid for this PC and for the ones considered in the following sections.

4.5 Heuristics for Constraints on the system response time

If we require that the mean system response time should be less than a given threshold ρ, we can express this constraint as:

$$\sum_{m=1}^{M} k_m \sum_{c'=1}^{C} \frac{\lambda_{c'}}{\Lambda} \frac{\frac{d_{rc'}}{k_m}}{1 - \frac{\Lambda^{\#}(\boldsymbol{\beta})}{\gamma_m} \sum_{c=1}^{C} \beta_c d_{mc}} = \sum_{m=1}^{M} \frac{\sum_{c=1}^{C} \beta_c d_{mc}}{1 - \frac{\Lambda^{\#}(\boldsymbol{\beta})}{\gamma_m} \sum_{c=1}^{C} \beta_c d_{mc}} < \rho, \tag{24}$$

The previous Eq. has infinite solutions in γ_m. Determining the optimal value (i.e., the one that minimizes the total number of instances), requires the solution of a non-linear optimization problem. We can however very easily compute one of the solutions (which might be sub-optimal). If we define γ_m as:

$$\gamma_m = \frac{1}{\alpha} \Lambda^{\#}(\boldsymbol{\beta}) \sum_{c=1}^{C} \beta_c d_{mc}. \tag{25}$$

then Eq. 24 becomes:

$$\sum_{m=1}^{M} \sum_{c=1}^{C} \beta_c d_{mc} \frac{1}{1-\alpha} = \frac{1}{\Lambda^{\#}(\boldsymbol{\beta})(1-\alpha)} < \rho, \tag{26}$$

we can compute α:

$$\alpha = \frac{1}{\rho} \left(\rho - \frac{1}{\Lambda^{\#}(\boldsymbol{\beta})} \right) \tag{27}$$

from which we derive:

$$\gamma_m = \frac{\rho \Lambda^{\#}(\boldsymbol{\beta}) \sum_{c=1}^{C} \beta_c d_{mc}}{\rho - \frac{1}{\Lambda^{\#}(\boldsymbol{\beta})}}. \tag{28}$$

4.6 Heuristics for Constraints on mean resource residence time

Now, let us consider a PC that imposes that the mean response time of a resource should be less than a given ρ_m. Using standard queueing theory results, we can formulate this requirement as:

$$k_m \sum_{c'=1}^{C} \frac{\lambda_{c'}}{\Lambda} \frac{\dfrac{d_{mc'}}{k_m}}{1 - \dfrac{\Lambda^{\#}(\boldsymbol{\beta})}{\gamma_m} \sum_{c=1}^{C} \beta_c d_{mc}} = \frac{\sum_{c=1}^{C} \beta_c d_{mc}}{1 - \dfrac{\Lambda^{\#}(\boldsymbol{\beta})}{\gamma_m} \sum_{c=1}^{C} \beta_c d_{mc}} < \rho_m, \tag{29}$$

Note that, since the server is split in k_m replicas, we have to consider the sum of the residence time at all the replicas, this the first member on the left hand side of Eq. 29 is multiplied by k_m. We can invert the Eq. and compute γ_m, as:

$$\gamma_m = \frac{\rho_m \Lambda^{\#}(\boldsymbol{\beta}) \sum_{c=1}^{C} \beta_c d_{mc}}{\rho_m - \sum_{c=1}^{C} \beta_c d_{mc}}. \tag{30}$$

5 An Algorithm for Closed Model

In Sections 4.5 and 4.6 we have proposed a set of heuristics to compute the minimum number of server needed to satisfy a given PC about response time. Such heuristics are based on an approximated solution of a non-linear optimization problem, thus they may provide a sub-optimal solution. In case we are interested in more efficient solutions, a different approach can be chosen: in this Section we propose an algorithm to support the capacity planning of a system.

Given the greater flexibility of algorithms, such technique can be applied on both open and closed model, and when a single server in a system is modeled by a set of resources. The latter feature can be useful when an accurate representation of each server is needed, for instance the modelling of a database server represented by two resources: one for the CPU and one for the disk. Moreover, the algorithm can be used also when we want to satisfy a set of non homogeneous PCs: for example that both response time and utilization must be lower than given thresholds.

In order to include these characteristics, we need to enrich the system model presented in Section 3. Consider a system with a multiclass workload composed by a set of server $\mathscr{S} = \{1,2,...,S\}$ and C customer classes where each server can be represented by one or more resources. Define with $\mathscr{M} = \{1,2,...,M\}$ the set of resources indexes and with Π a partition of $\mathscr{M}$ such that the number of partitioning set of Π is equal to S. Each partitioning set represents the mapping between a server and its resources, i.e. each server corresponds to a single partitioning set and the resources belonging to the partitioning set are used to model that server. Matrix $\boldsymbol{D}$ is defined as usual, but is also partitioned according to partition Π. The replication of a single server implies the replication of each of its resources according to the partition Π. In particular, if resource indexes i, j belong to the same partitioning set, we have that $k_i = k_j$ (See Section 3).

Example In the following we provide a simple closed model used in the rest of the paper as a running example. Consider a system composed of $C = 2$ classes of customers and $S = 3$

servers where each of them is modeled by two resources: CPU and disk. In such a case, the set of servers is $\mathscr{S} = \{1,2,3\}$ and the set of resources is $\mathscr{M} = \{1_{CPU}, 1_{Disk}, \ldots, 3_{CPU}, 3_{Disk}\}$, the partition Π is $\{\{1_{CPU}, 1_{Disk}\}, \{2_{CPU}, 2_{Disk}\}, \{3_{CPU}, 3_{Disk}\}\}$, where we have $S = 3$ partitioning sets $\{1_{CPU}, 1_{Disk}\}, \{2_{CPU}, 2_{Disk}\}, \{3_{CPU}, 3_{Disk}\}$.

The replication of a server consists on the replication of both resources CPU and disk, thus the resulting resources set after replication of, for instance server 1, will be the following: $\{\{1^a_{CPU}, 1^a_{Disk}, 1^b_{CPU}, 1^b_{Disk}\}, \{2_{CPU}, 2_{Disk}\}, \{3_{CPU}, 3_{Disk}\}\}$, where 1^a_{CPU} and 1^b_{CPU} are replicas of the CPU of server 1, and 1^a_{Disk} and 1^b_{Disk} are replicas of the disk of server 1. Given that the replication of a server implies the replication of each of its resources, we can denote the configuration of instances in the usual way. Thus, the service demand matrix $\mathbf{D}$ and the matrix $\mathbf{D}_{2-1-1}$ resulting from the replication of server 1 will be:

$$\mathbf{D} = \left|\begin{array}{rr} 5 & 18 \\ 4 & 11 \\ \hline \\ 20 & 41 \\ 6 & 20 \\ \hline \\ 6 & 51 \\ 15 & 56 \end{array}\right| \qquad \mathbf{D}_{2-1-1} = \left|\begin{array}{ll} 2.5 & 9 \\ 2 & 5.5 \\ 2.5 & 9 \\ 2 & 5.5 \\ \hline \\ 20 & 41 \\ 6 & 20 \\ \hline \\ 6 & 51 \\ 15 & 56 \end{array}\right| \tag{31}$$

where the values are in ms.

5.1 The capacity planning algorithm

The main goal of this Section is to define a procedure by which one can correctly dimension the system to satisfy a given set of PCs. In closed model the response time increases proportionally to the load of the system, that is the size of jobs population circulating inside it. After a certain population size, the response time will become larger than the threshold defined in the PC. Also for closed models, as seen is Section 3.2 for open ones, improvements in response time can be obtained by replicating some of the servers and equally sharing the load among them. The proposed algorithm follows an iterative approach: starting from the analysis of the system without replicas and with a minimum intensity workload, the number of jobs is incremented until a PC is violated. In such a case, a server is duplicated splitting uniformly the incoming requests among its replicas. The effectiveness of the algorithm strongly depends on the policy used to choose which server duplicate. The proposed policy is based on the bottleneck analysis: whenever a specific PC is violated, we duplicate the bottleneck of the system, that is, the most utilized server. Then, the new system is re-analyzed to check whether it satisfies the PCs. In affirmative case, the intensity workload is increased; otherwise, another server is identified as bottleneck and replicated. Such process continues until the given maximum intensity is reached.

The formal description of the process is shown in the Capacity Planning Algorithm 1. It takes as parameters the maximum population size of the system, the population mix and the list of performance constraints $|\rho_c|$ on the per-class response time (Algorithm 1: line 1). The algorithm starts testing the satisfiability of the given thresholds (Algorithm 1: line 2+3), in particular if the sum over all resources of the service demands for a given class is greater than the corresponding threshold, the PC is unsatisfiable. Then, the algorithm

Algorithm 1 Capacity Planning Algorithm.

```
1:  $INPUT(\boldsymbol{D}, NMAX, \beta, |\rho_c|)$
2:  if $TestSatisfiability(\boldsymbol{D}, |\rho_c|) == FALSE$ then
3:      $Unsatisfiable$
4:  else
5:      $NVals = [NMIN..NMAX]$
6:      for all $N \in NVals$ do
7:          $m = GeneratePopulation(\boldsymbol{D}, \beta, N)$
8:          $|R_c| = AnalyseModel(m)$
9:          $success = TestPerformanceConstraints(|R_c|, |\rho_c|)$
10:         while $NOT success$ do
11:             $BottleneckServer = IdentifyBottleneck(m)$
12:             $m = AddStationServer(BottleneckServer)$
13:             $|R_c| = AnalyseModel(m)$
14:             $success = TestPerformanceConstraints(|R_c|, |\rho_c|)$
15:         end while
16:     end for
17: end if
```

iteratively generates and analyses the model with an increasing population size and the given population mix, computing the achieved per-class response time $|R_c|$(Algorithm 1: line 7+8). Then it tests whether the results satisfy the given PCs (Algorithm 1: line 9). Whenever the PCs are violated, it proceeds to identify (Algorithm 1: line 11) and to replicate (Algorithm 1: line 12) the bottleneck server, i.e. the one with maximum aggregate utilization. As said, the replication implies that the initial demand is shared uniformly among all the servers of the bottleneck server type. The replication is iterated until the PCs are satisfied. The whole process is done until the maximum population size is reached. The analysis of the model is provided by the *Mean Value Algorithm*(MVA)Lazowska et al (1984), a standard technique of operational analysis.

We apply the algorithm to the example described in Section 5 with a population mix $\beta = 0.7$, assuming thresholds of 0.2 and $0.6s$ for the class c_1 and c_2, respectively. The algorithm starts analyzing the model with the minimum number of jobs, in such a case $N = 2$ [1]. Such model satisfies the PCs, thus the number of jobs in the system is increased till we reach $N = 11$ jobs, where the first PCs violation happens for both classes c_1 and c_2. Inspecting the aggregated utilizations, server S_2 can be identified as the system bottleneck and therefore it is replicated. So, two more replicas (i.e. CPU and Disk) are added to S_2 server sharing uniformly the requests among both pair of CPU and Disk of server S_2. Re-running the model with the new configuration 1-2-1, PC for class c_2 is still violated, but now the bottleneck has shifted to server S_3. A new replica of such server is added resulting on the configuration 1-2-2 which satisfies both PCs, therefore a new model with $N = 12$ jobs is analyzed. Such process is repeated till the scenario with $N = NMAX = 1000$ jobs is reached, where the configuration 37-126-112 satisfies both PCs and the algorithm ends. Table 2 shows the aggregate utilization, i.e. the sum of the utilizations for the two classes, and the achieved per-class response time measured with the various configurations; violation of PCs and corresponding system bottlenecks are highlighted by bold values.

[1] the minimum number of jobs is equal to the number of workload classes, because there must be at least one job for each class in order to define a proper model

Table 2 Aggregate utilization and per-class response time for c_1 and c_2 classes at different servers. Thresholds on per-class response times are set to 0.2 and 0.6 for classes c_1 and c_2, respectively. Violations of the thresholds and the system bottlenecks are highlighted in bold.

Iteration		Aggregate Utilization						Response time	
		S_1		S_2		S_3			
N	Configuration	CPU	Disk	CPU	Disk	CPU	Disk	c_1	c_2
2	1-1-1	0.150	0.105	0.470	0.173	0.304	0.459	0.067	0.236
...									
11	1-1-1	0.274	0.203	**0.949**	**0.321**	0.468	0.831	**0.211**	**0.632**
11	1-2-1	0.322	0.240	0.567	0.189	**0.531**	**0.976**	0.171	**0.608**
11	1-2-2	0.437	0.325	0.763	0.256	0.368	0.663	0.129	0.415
12	1-2-2	0.456	0.334	0.771	0.266	0.407	0.693	0.136	0.439
...									
1000	37-126-112	0.746	0.551	0.753	0.256	0.427	0.748	0.187	0.599

5.2 Complexity

In the proposed version, the algorithm does not impose any restriction to the total number of instances of the resulting replicated system, but in real world physical resources are limited. As a first solution, resources' consumption can be mitigated implementing instances by different VMs sharing a limited number of physical hosts. In such a case, the assumption that the workload of a server can be evenly partitioned between its instances may not hold, due to the sharing of the same physical host among several VMs. Clearly, a trade-off arises between the strictness of the constraint to satisfy and the number of used physical resources.

The algorithm can be used in two ways: *off-line* to plan the capacity of a system and *on-line* to dynamically scale the system resources to respect the considered PCs. When used off-line for N jobs, the algorithm must repeat the inner loop N times, one for each job. The loop can perform several iterations until the PCs are matched. However, at most one new copy for every server (e.g. when the PCs are so strict that each new job requires an entire copy of a server on its own), so its complexity is $O(S)$, where S is the number of servers[2]. To verify the PCs, MVA is used with complexity $O(NM)$. The total complexity of the algorithm is thus $O(N^2MS)$ when used off-line. When used on-line however, the algorithm can start from the previously computed solution, and its complexity reduces to $O(NMS)$. Since usually S and M are not very large, the proposed procedure can be implemented in common system management hardware even with large number of jobs.

The application of the algorithm on-line poses a further practical issue: the additional overhead introduced by VMs migration that, according to the specific application, may be relevant. Actually such cost is not included in the algorithm, it is assumed that it can be considered negligible with respect to the system response time. We plan to account for it in future works.

6 Numerical results

In this Section we present the results achieved by both the analytical approach proposed in Section 4 for open models, and the algorithm applied for the closed model described in

[2] In some cases, when the PCs are particularly tight, the actual number of iterations can be larger than S to account for the random routing considered by the technique, but its complexity is still $O(S)$

Section 5. We will first show how the Equations given in Section 4 can be used to properly choose the number of replicas required to handle a given workload while respecting a set of PCs. We apply the proposed equations on a test system to investigate the results provided in case of satisfaction of PCs about utilization and response time. The test system was simulated using the JMT tool Bertoli et al (2009): confidence interval at 99% were evaluated, but only mean values are shown to simplify the presentation.

Then, we will study in detail the behavior of the algorithm applied on the closed model of Section 5. In particular, the response time and the number of replicas needed to fulfill PCs related to per-class index. Next, we show the sensitivity of the algorithm to different values of the PCs thresholds and the interaction with the population mix. Finally, we investigate the effectiveness of the proposed system bottleneck policy to provide a minimum solution in term of number of replicas needed to satisfy the PCs. The example was analysed using the JMVA application included in the JMT tool.

6.1 Sizing an open system

Let us consider the three-tier system with $C = 2$ classes, and $M = 3$ servers, characterized by the demand matrix $\boldsymbol{D}$ of Eq. 3. Suppose that the utilization of each server m for each class c must be less than the following given values σ_{mc}:

$$|\sigma_{mc}| = \begin{vmatrix} 0.5 & 0.5 \\ 0.3 & 0.7 \\ 0.65 & 0.35 \end{vmatrix}. \tag{32}$$

If we apply the results presented in Section 4.3, we obtain that the number of replicas for each server is 1-1-3.

Figure 4(a) shows the utilization of all the combination of classes and servers for the 1-1-2 and the 1-1-3 configurations, together with the target value required by the PC. Combinations are labeled with $C_i - M_m$, for instance the utilization of server 1 for class 2 is labeled $C_2 - M_1$.

As it can be seen, the 1-1-3 configuration respects all the PCs, while the 1-1-2 violate the constraint on the second class for the third server (C2-M3), where the utilization is about 40% and the requirement should be less than 35%. Next we put the requirement on the utilizations of single server to be less than σ_m defined as $|\sigma_m| = |0.5 \;\; 0.3 \;\; 0.8|$. Using the results presented in Section 4.4, we can see that at least 2-4-2 replicas are required to satisfy the PCs. In Figure 4(b) we present the utilization of the servers for three configurations: 2-2-2, 2-3-2 and 2-4-2. Clearly the 2-2-2 configuration violates the PC on the second server. At first sight, the 2-3-2 configuration would seem to be adequate to satisfy all the constraints. However, at a closer look, we can see that with this configuration the utilization of the second server would be 30.26%, slightly higher than the 30% required by the PC.

Constraints on the response time of the single server is considered in Figure 4(c), where there PCs are set according to the following values, in *msec*: $|\rho_m| = |1000 \;\; 500 \;\; 1050|$. In this case, applying the results presented in Section 4.6, we have that minimum configuration should have at least 2-3-2, and a 2-2-2 configuration will violate the PC on the second server. Finally, we consider the system response time and we use the expression presented in Section 4.5. In particular we examine a series of possible constraints, $\rho = 14000 msec$, $\rho = 8200 msec$, $\rho = 4200 msec$, $\rho = 2200 msec$, $\rho = 1200 msec$ and we compute the configuration required to obtain such system response time. They are 1-1-2, 1-2-2, 2-2-2, 2-2-2 and 5,6,6 respectively. Figure 4(d) shows that by using the proposed configuration the system response time is always lower than the requirement. However, since the Equations given in Section 4.5 compute only a sub-optimal solution, there could be cases where the constraint can be met with a smaller set of instances. In this case this happens for the $\rho = 4200 msec$ constraint, which is met not only for the 2-2-2 configuration (the one computed by Eq. 28), but also for the 1-2-2 configuration that uses one instance less.

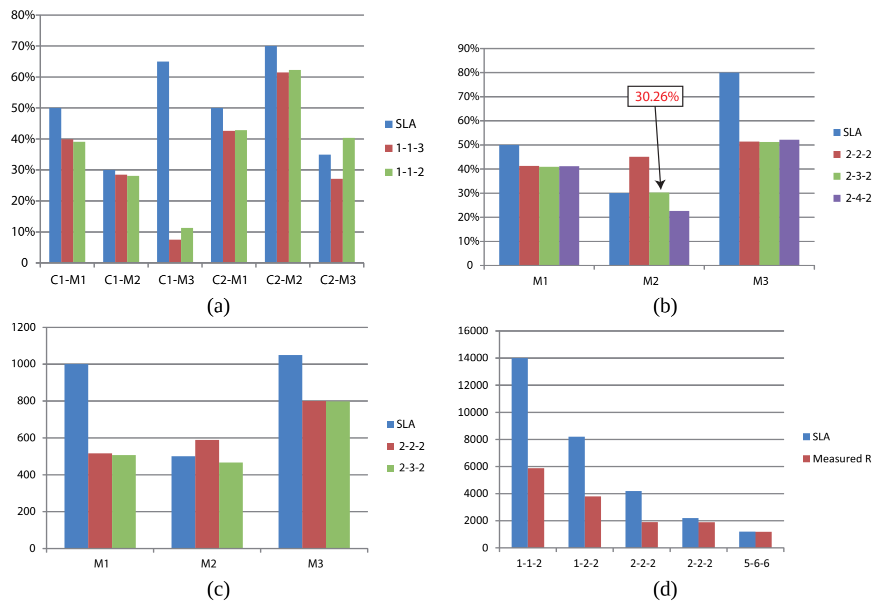
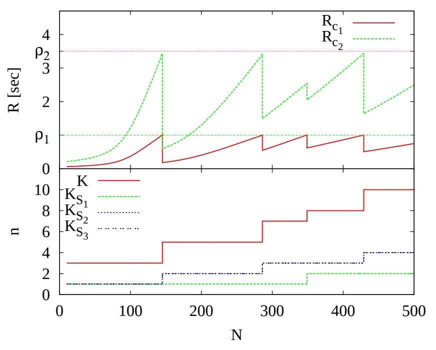


Fig. 4 Minimum number of replicas to handle a workload arrival rate $\Lambda = 0.0028$ *jobs/msec* and population mix $\boldsymbol{\beta} = |0.36, 0.64|$ with the following constraints on the: (a) utilization of a server *m* by a class *c*; (b) total utilization of a server *m*; (c) response time of server *m*; (d) system response time. The last two are expressed in *msec*.

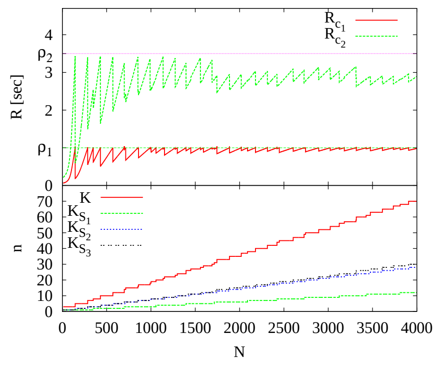
6.2 Considering a closed model

Let us focus on the analysis of the model described in the running example of Section 5. We start considering the PC thresholds about the per-class response time $|\gamma_c| = |1 \;\; 3.5|$ and running the algorithm for different population mixes β. Figure 5 a-c), shows the effect of the replication, for a population size varying from 1 to 500 respectively for $\beta = 0.3$, $\beta = 0.5$ and $\beta = 0.7$. The upper part of the figure shows the average response time for both the classes c_1 and c_2, together with their respective PCs.

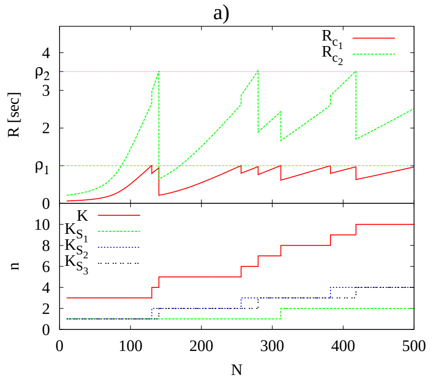
The bottom part shows the number of replicas per server, and the total number of instances (that is, the total cost) required to run the system with the required PCs. As expected, whenever one of the two classes reaches the threshold, the most heavily loaded server is replicated (new resource replicas for the considered server is added). The addition of new instances creates a discontinuity in the response time: in particular, it reduces the response time for the class whose PC was violated. For the other classes however, two possible behaviors can be observed: it can either decrease (e.g. $N \approx 140$ for $\beta = 0.5$), or increase (e.g.



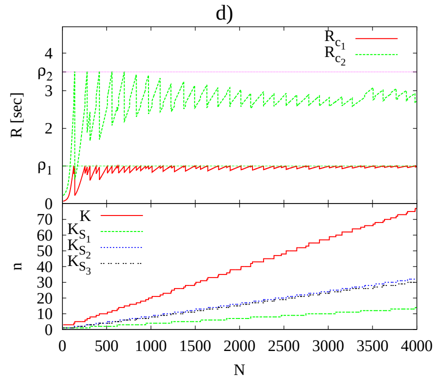
a)



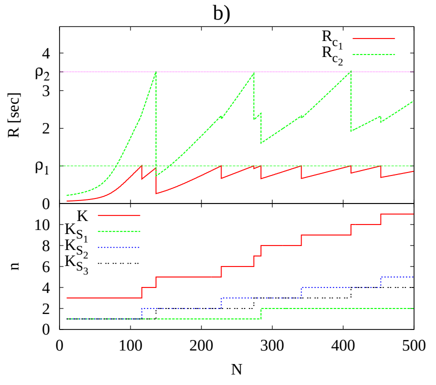
d)



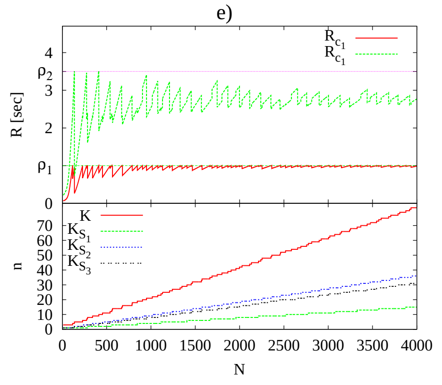
b)



e)



c)



f)

Fig. 5 Mean System Response Time per classes (c_1, c_2); ρ_{c_1} = 1 sec; ρ_{c_2} = 3.5 sec; for different population mixes β and maximum population size N: a) $\beta = 0.3; N = 500$, b) $\beta = 0.5; N = 500$, c) $\beta = 0.7; N = 500$, d) $\beta = 0.3; N = 4000$, e) $\beta = 0.5; N = 4000$, f) $\beta = 0.7; N = 4000$.

$N \approx 130$ for $\beta = 0.5$). The decrease is quite intuitive: the replication of the bottleneck server has benefit not only on the class whose response time was over the PC, but also for the other one. The increase might seem strange at first, but it also has a physical explanation: when the bottleneck server is replicated, the throughput of all the replicas together is increased. For this reason, a non-replicated server, may see an increase in the number of its arrivals, creating thus an higher mean service time. Moreover there are cases where two servers become bottleneck almost at the same time (e.g. $N \approx 145$ for $\beta = 0.3$). In this case, in order to obtain the required PC, two servers (in particular both S_2 and S_3) have to be replicated at the same time. To summarize, let us follow the evolution of the number of replicas required to obtain the given PC for $\beta = 0.5$. The system is able to handle the requests with the wanted performance for a total population of about $N \approx 130$. Then the PCs for the c_1 class is violated, and the server S_2 has to be replicated. This causes an increase in the response time to c_2 class, since now S_3 becomes more loaded. After a few more request, at $N \approx 140$, the response time for the c_2 class reaches its threshold. The server S_3 has now to be replicated, creating a large improvement in the response time for both classes. The system in this new configuration 1-2-2 is able to handle a population of around $N \approx 260$, where S_2 becomes the bottleneck again and violates the PC on the c_2 class. The server S_1 becomes a bottleneck only for $N \approx 315$, where to respect the PCs in configuration 2-3-3, 8 replicas are required. For $N = 500$, in configuration 2-4-4 ten machines are required to maintain the PCs. It is interesting to see how the replication pattern increases for larger populations. Figure 5 d-f), shows the same settings for β, this time up to $N = 4000$. Several insights can be argued examining these plots: first the growth in the number of replicas is linear with respect to the population size. In this case the replicas of the S_2 and S_3 servers have more or less the same slope (with the S_3 a little bit higher for $\beta = 0.3$, and lower for $\beta = 0.5, 0.7$). This can be justified by the fact that demands of both servers presented in Table 31 are similar, and higher with respect to server S_1. Second, we can see that total number of replicas necessary to handle a given population strongly depends on the population mix β. In this case 70 replicas are required with $N = 4000$ for $\beta = 0.3$, and 77 for $\beta = 0.7$, that is 10% more. Finally, it is very interesting to note that for large populations, only one of the PCs become dominant. As it can be seen, when the number of replicas becomes large, the effect of the replication decreases: the jump in the response time becomes smaller and smaller. One of the two response time (the c_2 in this case), becomes almost constantly equal to the threshold. The other, tends to be smaller, and to have a different asymptotic behavior. Despite an higher jump that destroys the trend around $N \approx 3500$ for all the considered β, the response time of the c_2 class seems to stabilize around 2.8 sec., almost 25% less than the required PC.

It is thus interesting to test the sensitivity of the proposed technique with respect to the thresholds. With a fixed threshold of 1 sec. for the c_1 class, we vary the threshold of the c_2 class from 2 sec. to 3.5 sec., and study the total number of replicas necessary to satisfy the PCs with a population of $N = 3000$, for two opposite mixes: $\beta = 0.3$ and $\beta = 0.7$. The results in Figure 6 confirm the importance of the population mix showing a difference of around 15 replicas between the two considered values of β. It is interesting to see that the change in the threshold seems to affect both population mixes in the same way, causing an increase of around $8 \sim 10$ machines. On the other hand the curve becomes flat for large thresholds, confirming the fact that in this case the PC on the c_2 class is masked out by the constraint on the c_1 class.

In the previous experiments, we have seen that the considered replication strategy (that is, replicating the most used server when an increase in the population determines the violation of an PC) allows the system to deal with a growing number of requests, while respecting the given performance constraints. Now we want to investigate whether the proposed policy is optimal or not. A rigorous prove, based on queuing theory results would be advisable, but it is outside the scope of this paper. We tackled the problem by showing that at least in the considered case the replication pattern proposed by the algorithm is always the best possible. Of course this cannot give the confidence that can be derived by a rigorous proof, but at least it can show that the proposed strategy can find the optimal solution in the considered cases. In particular, for each proposed configuration, we have computed all the possible

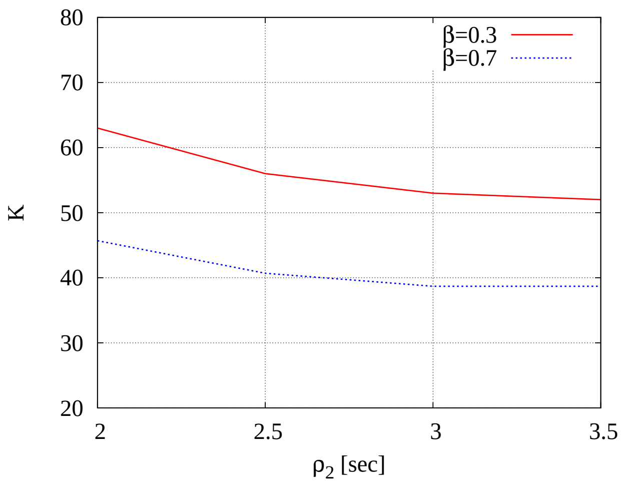


Fig. 6 Number of replicas needed to satisfy different value ρ_{c_2} of the PC with population size $N = 3000$.

configurations with one less replica. For example, if the algorithm determined that the configuration with the minimum number of replicas required to match the PCs is 3-3-2, we tried the three configurations: 2-3-2, 3-2-2 and 3-3-1. We then computed the minimum response time obtainable (for each class) between the three configurations with one less replicas: this corresponds to a lower bound to the response time that can be achievable with less replicas than computed. We then compared this minimum with the PCs, and verified that for all the considered cases, at least for one of the classes it was greater than the PC threshold. Figures 7 and 8 show the minimum response time for one less replica (R_{c_1}, R_{c_2}), and the response time computed by the proposed algorithm $(R^*_{c_1}, R^*_{c_2})$ for both the c_2 and c_1 class, as function of the different population mixes β, for $N = 1000$ and $N = 2000$. In both cases, the minimum response time with one less replica for the c_1 class is always greater than the PC threshold. In some cases however the difference is minimal (e.g. for $\beta = 0.4$), and thus it cannot be fully appreciated by the graph. It is interesting to see that for $N = 1000$ and $\beta = 0.1$, also the minimum response time for the c_2 class would have been greater than its threshold.

7 Conclusions

In this paper we have considered the topic of consolidation and replication from an end-user point of view. In particular for open workloads, we have proposed analytical equations to predict the effect of consolidation, and to appropriately dimension a system, in terms of replication of service, to match a given set of performance objectives. For closed workloads, we have developed an iterative algorithm to fulfill PCs based on the principle of provisioning resources only when needed. We have presented a set of experiments to investigate the effectiveness of both approaches.

Future works will address more complex performance objectives, and will consider more complex types of resources, to better capture the internal parallelization characteristics related to multi-core and multi-threaded resources.

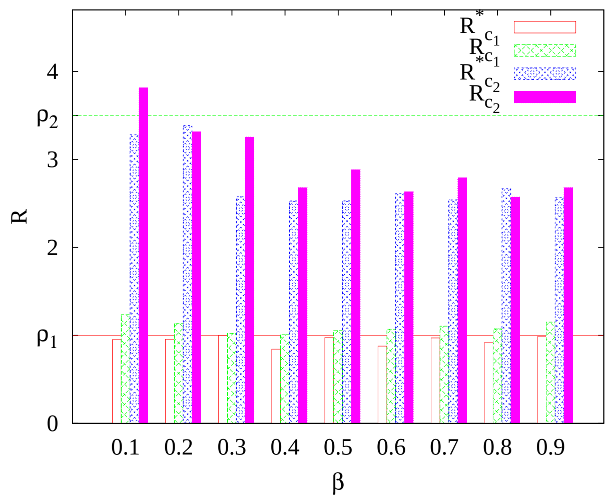


Fig. 7 Comparing optimal solution with one less replica $N = 1000$.

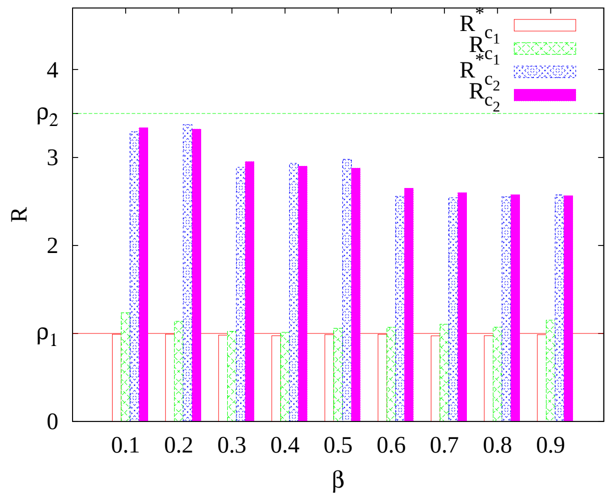


Fig. 8 Comparing optimal solution with one less replica $N = 2000$.

Acknowledgements This work has been partially supported by the ”AWS in Education research grant” from Amazon, and by the ”ForgeSDK” project sponsored by Reply S.R.L.

References

Balbo G, Serazzi G (1996) Asymptotic analysis of multiclass closed queueing networks: Common bottlenecks. Performance Evaluation 26(1):51–72

Benevenuto F, Fernandes C, Santos M, Almeida VAF, Almeida JM, Janakiraman GJ, Santos JR (2006) Performance models for virtualized applications. In: Min G, Martino BD, Yang LT, Guo M, Rnger G (eds) ISPA Workshops, Springer, Lecture Notes in Computer Science, vol 4331, pp 427–439

Bennani M, Menascé D (2005) Resource allocation for autonomic data centers using analytic performance models. In: Autonomic Computing, 2005. ICAC 2005., pp 229 –240, DOI 10.1109/ICAC.2005.50

Bertoli M, Casale G, Serazzi G (2009) JMT: Performance engineering tools for system modeling. SIGMETRICS Perform Eval Rev 36(4):10–15, DOI 10.1145/1530873.1530877

Bobroff N, Kochut A, Beaty K (2007) Dynamic placement of virtual machines for managing sla violations. In: Integrated Network Management, 2007. IM ’07. 10th IFIP/IEEE International Symposium on, pp 119 –128, DOI 10.1109/INM.2007.374776

Bushehrian O (2011) The application of fsp models in automatic optimization of software deployment. In: ASMTA, pp 43–54

Curino C, Jones EP, Madden S, Balakrishnan H (2011) Workload-aware database monitoring and consolidation. In: Proceedings of the 2011 international conference on Management of data, ACM, New York, NY, USA, SIGMOD ’11, pp 313–324

Ganapathi A, Chen Y, Fox A, Katz R, Patterson D (2010) Statistics-driven workload modeling for the cloud. In: Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on, pp 87 –92, DOI 10.1109/ICDEW.2010.5452742

Gribaudo M, Piazzolla P, Serazzi G (2012) Consolidation and replication of vms matching performance objectives. In: ASMTA, pp 106–120, DOI 10.1007/978-3-642-30782-9_8

Jackson JR (1963) Jobshop-like queueing systems. Management Science 10(1):pp. 131–142

Khanna G, Beaty KA, Kar G, Kochut A (2006) Application performance management in virtualized server environments. In: NOMS, pp 373–381

Kokkinos P, Christodoulopoulos K, Kretsis A, Varvarigos E (2008) Data consolidation: A task scheduling and data migration technique for grid networks. In: Proceedings of the 8th IEEE Int. Symposium on Cluster Computing and the Grid, IEEE Computer Society, Washington, DC, USA, pp 722–727

Lazowska ED, Zahorjan J, Graham GS, Sevcik KC (1984) Quantitative System Performance. Prentice-Hall

Menascé DA (2005) Virtualization: Concepts, applications, and performance modeling

Menascé DA, Bennani MN (2006) Autonomic virtualized environments. In: Proceedings of the International Conference on Autonomic and Autonomous Systems, IEEE Computer Society, Washington, DC, USA, ICAS ’06, pp 28–

Ongaro D, Cox AL, Rixner S (2008) Scheduling i/o in virtual machine monitors. In: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, ACM, New York, NY, USA, VEE ’08, pp 1–10

Padala P, Shin KG, Zhu X, Uysal M, Wang Z, Singhal S, Merchant A, Salem K (2007) Adaptive control of virtualized resources in utility computing environments. In: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, ACM, New York, NY, USA, EuroSys ’07, pp 289–302

VirtualBox (2013) http://www.virtualbox.org

VMware (2013) http://www.vmware.com

Watson BJ, Marwah M, Gmach D, Chen Y, Arlitt M, Wang Z (2010) Probabilistic performance modeling of virtualized resource allocation. In: Proceedings of the 7th international conference on Autonomic computing, ACM, New York, NY, USA, ICAC ’10, pp 99–108