Throughput maximization with multiclass workloads and resource constraints

Davide Cerotti[1], Marco Gribaudo[1], Ingolf Krüger[2], Pietro Piazzolla[1], Filippo Saracini[2], and Giuseppe Serazzi[1]

[1] Dip. di Elettronica e Informazione, Politecnico di Milano, via Ponzio 34/5, 20133 Milano, Italy.
`[name.surname]@polimi.it`
[2] Department of Computer Science and Engineering University of California San Diego, La Jolla, CA 92023-0404, USA
`[fseracini,ikrueger]@ucsd.edu`

Abstract. In this paper we study the impact of different types of constraints on the maximum throughput that a system can handle. In particular, we focus on constraints limiting the use of resources and/or the allowed response time. The problem is made even more difficult by the pronounced diversity in resource requirements of the different applications in execution, i.e., by the multiclass characteristic of the workloads. The proposed approach allows to determine the maximum load of the different classes, while still satisfying the considered performance objectives. An experimental validation of the described technique through the study of a realistic e-commerce application is presented.

1 Introduction

Over the last few years, the growth of available physical resources was a very evident phenomenon thanks to the widespread diffusion of cloud computing. Concurrently, the capacity requirements of the new applications has also increased significantly. Modern computing infrastructures are characterized by a huge amount of resources with heterogeneous capacities (e.g. [13, 22]) that are shared among several applications with very different requirements. Such features have made the allocation of resources a very critical problem because the capacity required to sustain the flow of requests may not be always available. The servers performance remains a crucial component of many computing infrastructures. In order to address this problem in the case of shared systems, different types of constraints are imposed to the resources deployed to the various applications.
In this paper we study the effects of a variety of resource and time based constraints on a performance objective function. The constraints at the resource level are based on the utilization, and on the maximum number of jobs in the system. As time based constraints we consider thresholds on residence time and on the mean system response time. The performance objective function to be maximized is the system throughput.

In this paper, peculiar properties of open multiclass queuing networks subject to different types of constraints are investigated and their applicability to some practical problems is proposed. In particular, the problem solved is the following: find the maximum throughput per class of requests that the system can sustain while satisfying the given time and/or resource based constraints.
Even if multiclass open queuing networks are well established mathematical models, the specific way in which they are used in this paper constitute a novelty. This application of known theory can provide new interesting insights and be useful to solve stream-line research problems about the allocation of resources in contexts such as cloud computing and multi-tier architectures.
The structure of the paper is the following. Section 2 analyzes the related works, and Section 3 presents a brief overview of some basic results of open queuing networks that will be used throughout the paper. Sections 4 and 5 address the identification of the maximum throughput without and with constraints. Section 6 applies the results to a realistic system, and Section 7 concludes the paper.

2 Related work

A common problem in data center management is resource allocation and provisioning in the presence of loads that can vary frequently with Internet applications. Resource over-provisioning leads to low average server utilization and high recurring utility costs. On the other hand, under-provisioning translates in a potential shortage of computing resources. Both strategies may cause serious economic losses. Provisioning decisions are usually taken by either hardware, platform or application providers, even if in many cases the responsibility of provisioning id demanded to end users (see e.g [22, 24, 23]).
Several techniques have been recently introduced to deal with the identification of the proper set of resources. Autonomic data centers, referred sometimes to as self-tuning, self-adaptive or self-aware systems (e.g.[10, 27]) try to adapt the allocated resources to the fluctuations of requests in order to meet agreed operational objectives. In [19, 20] the authors take into consideration the response time only, typically defined as the aggregated value across all the request classes. Our solution differs from these approaches as it deals also with multi-class workloads. Moreover, our approach enables the data center resource management to identify the workload mix that maximizes both the throughput and the utilization of resources under a set of constrains, not only response time. Other bottleneck identification techniques for queueing networks are considered in [6].
Optimization of a cost function has been addressed in many different ways: using combinatorial search algorithms in [4, 8, 9], linear programming in [1], game theory in [25], while in other cases the maximization of some utility functions like response time (e.g. [21]) or power consumption (e.g. [2]) is sought.
The problem of scalability has been approached allocating and deallocating dynamically the resources. Admission control schemes have been devised in order to guarantee given objectives. Usually, performance goals are achieved by rejecting some types of requests during peak periods [7, 9] or by maximizing of provider's

revenue [12, 14]. Different approaches are also based on policies that control the arrival rates of the classes of applications in order to saturate simultaneously multiple resources to maximize a given metric as in [26, 11].

3 Background

In this section, we briefly review the basic notations that will be used in the following. We consider a workload consisting of C classes of requests and a system with M resources that operate at a fixed rate. Requests cannot change class during their execution. Let D_{mc} be the global service demand of a class c customer on station m. The service demands of the system are described by the following $M \times C$ demand matrix:

$$\mathbf{D} = \begin{vmatrix} D_{11} & \dots & D_{1C} \\ \vdots & & \vdots \\ D_{M1} & \dots & D_{MC} \end{vmatrix} \tag{1}$$

Class c requests enter the system at a Poisson rate λ_c. We collect the workload intensities of all the classes in a vector $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_C)$. The *overall arrival rate*, i.e., the *global load* of the system, is given by $\Lambda = \sum_c \lambda_c$. Let $\boldsymbol{\beta} = (\beta_1, \dots, \beta_C)$ be the *population mix* vector, where β_c is the fraction of arriving requests that belong to class c. The following relations between Λ, $\boldsymbol{\lambda}$, $\boldsymbol{\beta}$ exists:

$$\begin{aligned} \boldsymbol{\lambda} = \Lambda\boldsymbol{\beta}, \lambda_c = \Lambda\beta_c, \sum_c \lambda_c = \Lambda, \\ \boldsymbol{\beta} = \frac{\boldsymbol{\lambda}}{\Lambda}, \ \ \beta_c = \frac{\lambda_c}{\Lambda}, \ \ \sum_c \beta_c = 1 \end{aligned} \tag{2}$$

We define the *population mix scaled demand* for resource m D_m as: $D_m(\boldsymbol{\beta})$

$$D_m(\boldsymbol{\beta}) = \sum_c \beta_c D_{mc} \tag{3}$$

$D_m(\boldsymbol{\beta})$ represents the mean service demand generated on resource m by a given population mix $\boldsymbol{\beta}$.
We use the subscript 'mc' to denote an index computed for class c at resource m (U_{mc}, R_{mc} and Q_{mc}). We also denote aggregated metrics related to class c by the index $\star c$: $U_{\star c}$, $R_{\star c}$ and $Q_{\star c}$. The metrics at the resource level are denoted by the index of the resource: R_m and Q_m.

4 Maximization of unconstrained systems

With a multiclass workload, the maximum throughput that a system can handle is a function of the fraction of the jobs of the different classes in concurrent execution, i.e., is *mix dependent*. Indeed, for a given workload intensity there are mixes in correspondence to which the system perform better than with other, i.e., that provide the maximum throughput and the minimum response time. We are considering open systems, and it is known that as the arrival rate of customers of the various classes increase, the number of customers in the system, and thus the response time, tend to grow *without bounds*, (i.e. the system *saturate*). In particular, with a given population mix $\boldsymbol{\beta}$, the system is not in saturation if the utilization of each resource is strictly less than 1, that is: $\Lambda D_m(\boldsymbol{\beta}) < 1, \quad \forall m : 1 \leq m \leq M$. This ensures that the system is stable, and it can be used to determine the maximum arrival rates that it can handle. For a given population mix $\boldsymbol{\beta}$, the maximum possible arrival rate $\hat{\Lambda}(\boldsymbol{\beta})$ can be determined by inverting the stability condition:

$$\hat{\Lambda}(\boldsymbol{\beta}) = \frac{1}{\max_m \{D_m(\boldsymbol{\beta})\}}. \tag{4}$$

It can then be interesting to determine the population mix $\boldsymbol{\beta}$ for which the system can experience the maximum throughput $\hat{\Lambda}(\boldsymbol{\beta})$. Since the utilization of the resources are linear functions, the population mix $\boldsymbol{\beta}^*$ corresponding to the maximum throughput can be obtained solving the following Linear Programming Problem (LPP):

$$\begin{array}{llll} \textbf{Variables:} & & \lambda_c, & 1 \leq c \leq C \\ \textbf{Objective:} & \text{maximize} & \sum_c \lambda_c & \\ \textbf{Constraints:} & \sum_d \lambda_d D_{md} \leq 1, & & 1 \leq m \leq M \\ & & \lambda_c \geq 0, & 1 \leq c \leq C \end{array} \tag{5}$$

The interpretation of the LPP of Eq. 5 is the following: the objective function corresponds to the total arrival rate, expressed as the sum of the arrival rates of the individual classes ($\Lambda = \sum_c \lambda_c$). Constraints ensure that arrival rates are positive ($\lambda_c \geq 0$), and that the utilization of each resource is less than 1 ($\sum_c \lambda_d D_{md} \leq 1$). If we call $\lambda_c^*, \quad 1 \leq c \leq C$ the optimizer, then the maximum allowed throughput corresponds to the value assumed by the objective function $\Lambda^* = \sum_c \lambda_c^*$. The optimal population mix can then be computed as $\boldsymbol{\beta}^* = \{\beta_c^*\}$ with $\beta_c^* = \lambda_c^* / \Lambda^*$.

The set of the population mixes that can achieve the maximum throughput depends on the set of the solution of the linear program. If the solution of the LPP of Eq.5 is a single point, then the maximum throughput can be obtained only for single population mix $\boldsymbol{\beta}^*$. If the solution of the LPP in Eq.5 is a segment or a convex polyhedron, then there exists a set of population mixes corresponding to the maximum overall throughput Λ^*.

We now see the application of the technique through the description of an e-commerce system consisting of four resources (*Management*, *CMS System*, *Inventory* and *Shipping*) and three classes of customers (*Intranet*, *On-line Purchase*, *In Store*). The service demands of the three classes are shown in Table 1. The *Management* resource is the storage server for the financial, transactions and customers data. The different contents of the website (mainly products catalog) are managed by the *CMS system*, while the administration of the catalog and the warehouse inventory involve the *Inventory* server. A dedicated resource is provided to handle the *Shipping* of purchased items. The back-end operations, including tasks related to the organization of the web site, the products catalog update, and the customers data update, are executed by the requests of the *Intranet* class. *On-Line Purchase* represents the process of buying one or more products through the web in a single transaction, while *In Store purchase* transactions take place off-line, and usually are started by a customer entering a physical store. During their execution the requests of each class visit all the four servers. Fig.1 shows the maximum throughput $\hat{\Lambda}(\boldsymbol{\beta})$ for all the possible

D		Class 1 ($ms.$)	Class 2 ($ms.$)	Class 3 ($ms.$)
		Intranet	**On-Line Purchase**	**In Store**
Resource 1	**Management**	80	65	60
Resource 2	**CMS system**	30	130	80
Resource 3	**Inventory**	45	30	135
Resource 4	**Shipping**	65	115	45

Table 1. Service demands **D** of the three classes of requests at the three resources of the e-commerce model.

mixes $\boldsymbol{\beta} = |\beta_1, \beta_2, (1-\beta_1-\beta_2)|$ of the requests of the three classes. The mixes of requests of each surface of the polyhedron shown correspond to a bottleneck on a different resource (indicated in the figure by its own label). The edges at the intersection of two surfaces are the *common saturation sectors*, and represent the mixes whose execution generate more two bottlenecks, i.e., that saturate concurrently two resources [3]. It is clear from the figure that there exists a point, corresponding to the population mix $\boldsymbol{\beta}^*$, for which the throughput is maximized (i.e. the top of the pyramid like polyhedron). In this point, more than two resources saturate concurrently.
Solving the LPP of Eq. 5 to determine the maximum system throughput with respect to the mixes $\boldsymbol{\beta}$, we obtain:

$$\boldsymbol{\beta}^* = |0.4512\ 0.2307,\ 0.3181| \quad , \quad \Lambda^* = 14.25\ jobs/s \tag{6}$$

It is interesting to point out that the maximum system throughput is obtained with $\beta_1 = 0.4512$, that is, when 45% of the requests in execution are of the *Intranet* class. This is not surprising, since the global service demand of the *Intranet* requests, i.e., the sum of its service demands over all the resource, is smaller with respect to the ones of the other two classes.

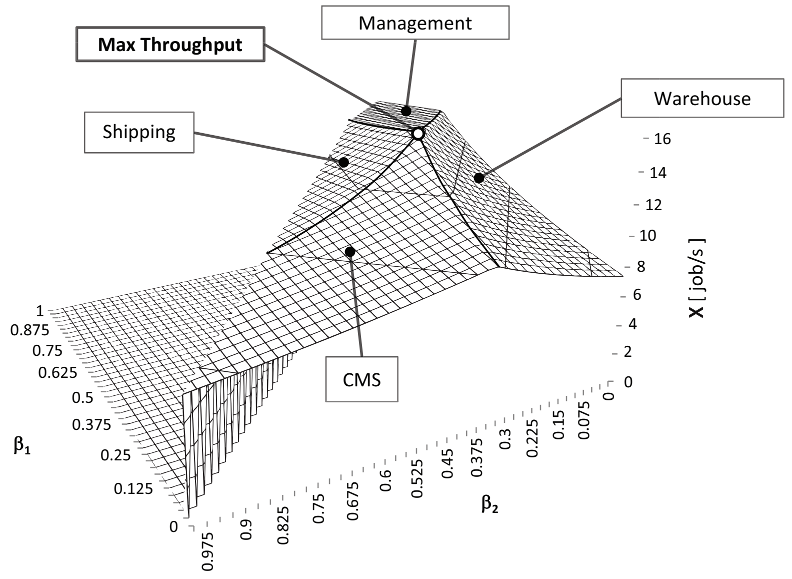


Fig. 1. Maximum system throughput vs the population mixes $\boldsymbol{\beta}$ in a three class, four resources system.

5 Performance constraints

We then consider the case in which constraints are imposed to some performance metrics of the system. Typical constrains are, for example, thresholds to the maximum mean response time, to the utilization, or upper bounds for the queue lengths of a resource. Constraints may reduce the maximum allowed arrival rate Λ^* to values that are smaller than $\hat{\Lambda}(\boldsymbol{\beta})$ given by Eq.4. They can also change the population mixes $\boldsymbol{\beta}^*$ for which the maximum throughput Λ^* can be achieved. To address this issue, we divide the performance constraints in two categories: *system-level* and *resource-level*. *System-level* constraints concern either the whole system or a specific class c. For all possible $\boldsymbol{\beta}$, as the total arrival rate Λ approaches $\hat{\Lambda}(\boldsymbol{\beta})$, both the queue length and the response time tend to the infinity:

$$\begin{aligned} \lim_{\Lambda\to\hat{\Lambda}(\boldsymbol{\beta})} R_{\star c} = \infty, \quad \lim_{\Lambda\to\hat{\Lambda}(\boldsymbol{\beta})} R_S = \infty, \\ \lim_{\Lambda\to\hat{\Lambda}(\boldsymbol{\beta})} Q_{\star c} = \infty, \quad \lim_{\Lambda\to\hat{\Lambda}(\boldsymbol{\beta})} Q_S = \infty. \end{aligned} \tag{7}$$

Resource-level constraints refer to a station m, either for a specific class c or for the aggregate of all the classes. In this case the utilization, the mean queue length and the response time can have very different behaviors depending on the considered population mix $\boldsymbol{\beta}$. Let us call $\boldsymbol{d}(\boldsymbol{\beta}) = \{m : 1 \le m \le M \wedge D_m(\boldsymbol{\beta}) = \max_l \{D_l(\boldsymbol{\beta})\}$, the set of resources that are bottleneck with the population mix

$\boldsymbol{\beta}$, and let us call $\hat{u}_m$, $\hat{b}_m$, $\hat{q}_m$, ($\hat{u}_{mc}$, $\hat{b}_{mc}$, $\hat{q}_{mc}$) the maximum utilization, mean response time and mean queue length that can be obtained for a resource m (respectively for class c requests at resource m). Let us focus on resource utilization first. If a resource m is a bottleneck (i.e. $m \in \boldsymbol{d}(\boldsymbol{\beta})$), then as the system reaches its instability point (i.e. $\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})$), the resource must be completely saturated. In other words, we have that:

$$\hat{u}_m = \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} U_m = 1, \qquad \forall_m \in \boldsymbol{d}(\boldsymbol{\beta}). \tag{8}$$

However, if we focus on the utilization of a given class c at a bottleneck station m, it will tend to the fraction of jobs of that particular class present at the considered resource. This fraction must be proportional to the arrival rate and to the demand of class c at resource m. This can expressed as:

$$\hat{u}_{mc} = \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} U_{mc} = \frac{\beta_c D_{mc}}{D_m(\boldsymbol{\beta})}, \qquad \forall_r \in \boldsymbol{d}(\boldsymbol{\beta}). \tag{9}$$

The mean response time of the mean queue length at a bottleneck resource m tends to infinity since the system is not able to handle all the incoming requests. That is:

$$\begin{gathered} \hat{b}_{mc} = \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} R_{mc} = \infty, \hat{b}_m = \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} R_m = \infty, \\ \hat{q}_{mc} = \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} Q_{mc} = \infty, \hat{q}_m = \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} Q_m = \infty, \\ \forall_m \in \boldsymbol{d}(\boldsymbol{\beta}). \end{gathered} \tag{10}$$

Instead if we consider a resource k that is not a bottleneck (i.e, $k \notin \boldsymbol{d}(\boldsymbol{\beta})$), the limiting value of the considered performance index is finite, and it can be computed with standard queueing network formulas using an arrival rate $\hat{\Lambda}(\boldsymbol{\beta})$. Let us call such limiting values:

$$\begin{aligned} \hat{u}_{kc} &= \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} U_{kc} = U_{kc}(\hat{\Lambda}(\boldsymbol{\beta})), & \hat{u}_k &= \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} U_k = U_k(\hat{\Lambda}(\boldsymbol{\beta})), \\ \hat{b}_{kc} &= \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} R_{kc} = R_{kc}(\hat{\Lambda}(\boldsymbol{\beta})), & \hat{b}_k &= \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} R_k = R_k(\hat{\Lambda}(\boldsymbol{\beta})), \\ \hat{q}_{kc} &= \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} Q_{kc} = Q_{kc}(\hat{\Lambda}(\boldsymbol{\beta})), & \hat{q}_k &= \lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} Q_k = Q_k(\hat{\Lambda}(\boldsymbol{\beta})), \\ &\forall_k \notin \boldsymbol{d}(\boldsymbol{\beta}). \end{aligned} \tag{11}$$

5.1 Resource-level constraints

Constraints on resources (either per class, or aggregate) can be computed with closed form expressions. Let us denote with u_{mc}, b_{mc}, q_{mc}, the maximum utilization, the response time, and the mean queue length allowed for a class c job at resource m, and with u_m, b_m and q_m the corresponding thresholds at resource m regardless of the class.

Constraints are automatically satisfied if they are based on thresholds that are greater than the maximum values determined in Eq.11: that is if either

$u_{mc} > \hat{u}_{mc}$, $b_{mc} > \hat{b}_{mc}$, $q_{mc} > \hat{q}_{mc}$, $u_m > \hat{u}_m$, $b_m > \hat{b}_m$ or $q_m > \hat{q}_m$.
Let us consider a non-trivial case, and let us initially focus on the response time of a class c job at resource m. The constraint is not trivial if $b_{mc} < \hat{b}_{mc}$. In this case we have that:

$$\frac{D_{mc}}{1 - \Lambda D_m(\boldsymbol{\beta})} < b_{mc}. \tag{12}$$

Since we consider that the system must be stable, we know that $U_m < 1 \quad \forall r$, which implies that $1 - \Lambda D_m(\boldsymbol{\beta}) > 0$ (since we have that $U_m = \Lambda D_m(\boldsymbol{\beta})$). We can then multiply both sides of the equation by the denominator obtaining:

$$D_{mc} < b_{mc} \left(1 - \Lambda D_m(\boldsymbol{\beta})\right). \tag{13}$$

We can thus derive Λ from Eq. 13:

$$\Lambda < \Lambda^* = \frac{b_{mc} - D_{mc}}{b_{mc} D_m(\boldsymbol{\beta})}, \tag{14}$$

that expresses the maximum total arrival rate Λ^* allowed for the population mix $\boldsymbol{\beta}$ to ensure that the mean response time is $R_{mc} < b_{mc}$.
The expressions for the maximum arrival rate that satisfies the other type of resource level constraints can be derived with similar computations, and they are shown in Table 2.

	Utilization	Response time	Mean queue length
Class c, res. m	$\Lambda < \frac{u_{mc}}{\beta_c D_{mc}}$	$\Lambda < \frac{b_{mc} - D_{mc}}{b_{mc} D_m(\boldsymbol{\beta})}$	$\Lambda < \frac{q_{mc}}{q_{mc} D_m(\boldsymbol{\beta}) + \beta_c D_{mc}}$
Res. m	$\Lambda < \frac{u_m}{D_m(\boldsymbol{\beta})}$	$\Lambda < \frac{b_m - D_m(\boldsymbol{\beta})}{b_m D_m(\boldsymbol{\beta})}$	$\Lambda < \frac{q_m}{D_m(\boldsymbol{\beta})\,(q_m + 1)}$

Table 2. Maximum throughput to assure resource-level constraints as function of the control parameters

5.2 System-level constraints

The maximum throughput for system-level constraints does not have a closed form expression, and it must be computed numerically. However, it is easy to show that the considered indices are monotone with respect to Λ, and have a closed form expression for their first derivative. These features allows the use of efficient numerical solution techniques that can obtain the maximum throughput with an iterative procedure. In particular, the Newton-Raphson method [17] is able to converge to the solution in very few iterations.
The tchnique requires an initial guess x_0 for the unknown that has to be computed. This initial solution should be greater than the actual value, in order to keep the guess decreasing (and thus always included in the stability region of the model). Let us focus on limiting the mean number of jobs in the system below a threshold q_s. From Eq.7 we know that $\lim_{\Lambda \to \hat{\Lambda}(\boldsymbol{\beta})} Q_S = \infty$, which means that it is always possible to find an initial guess greater than the threshold in an interval

close to x_0. We can then set $x_0 = \hat{\Lambda}(\boldsymbol{\beta}) - \delta$, where $\delta > 0$ is a small number such that $Q_S(x_0) > q_s$ (i.e the corresponding performance index computed in x_0 is greater than the required threshold). This parameter δ can be efficiently computed with an exponential scaling step. The procedure to compute the maximum throughput satisfying $Q_S < q_S$, is the following:

```
$\delta = 1$;
**do** {
    $\delta = \delta \cdot \delta_0$;
    $x = \hat{\Lambda}(\boldsymbol{\beta}) \cdot (1 - \delta)$;
} **while** ($Q_S(x) < q_S$);
**while** ($\left|\frac{Q_S(x)-q_S}{q_S}\right| > \epsilon$);
    $x = x - \frac{Q_S(x)-q_S}{Q'_S(x)}$;
}
$\Lambda = x$;
```
(15)

where ϵ is a term that represents the relative precision of the solution, δ_0 is a constant $0 < \delta_0 < 1$ corresponding to the exponential scaling factor and $Q'_S(x)$ is the first derivative of the mean number of requests in the system (as defined in Table 3). In the experiments presented in this paper we used $\delta_0 = 0.1$ and $\epsilon = 10^{-6}$.

	Response time	Mean queue length
Class c	$\frac{\partial R_{\star c}}{\partial \Lambda} = \sum_m \frac{D_{mc} D_m(\boldsymbol{\beta})}{(1 - \Lambda D_m(\boldsymbol{\beta}))^2}$	$\frac{\partial Q_{\star c}}{\partial \Lambda} = \beta_c \sum_m \frac{D_{mc}}{(1 - \Lambda D_m(\boldsymbol{\beta}))^2}$
System	$\frac{\partial R_S}{\partial \Lambda} = \sum_m \frac{D_m(\boldsymbol{\beta})^2}{(1 - \Lambda D_m(\boldsymbol{\beta}))^2}$	$\frac{\partial Q_S}{\partial \Lambda} = \sum_m \frac{D_m(\boldsymbol{\beta})}{(1 - \Lambda D_m(\boldsymbol{\beta}))^2}$

Table 3. First derivative of system-level indices

5.3 Constrained Maximum Throughput

Also for constrained systems, we can determine the population mix for which the maximum arrival rate could be achieved without violating the given performance constraints. This can be done by solving an Optimization Problem (OP), similar to the one proposed in Section 4, where extra constraints are added to account for the desired performance requirements:

$$\begin{array}{lll} \textbf{Variables:} & \lambda_c, & 1 \le c \le C \\ \textbf{Objective:} & \text{maximize } \sum_c \lambda_c & \\ \textbf{Constraints:} & \sum_d \lambda_d D_{md} \le 1, & 1 \le m \le M \\ & \lambda_c \ge 0, & 1 \le c \le C \\ & \textit{Performance constraints}_1 & \\ & \vdots & \\ & \textit{Performance constraints}_n & \end{array} \tag{16}$$

	Utilization	Response time	Mean queue length
Class c, res. m	$\lambda_c D_{mc} \leq u_{mc}$	$b_{mc}\left(1-\sum_d \lambda_d D_{md}\right) \geq D_{mc}$	$q_{mc}\left(1-\sum_d \lambda_d D_{md}\right) \geq \lambda_c D_{mc}$
Res. m	$\sum_d \lambda_d D_{md} \leq u_m$	$b_{mc}\left(1-\sum_d \lambda_d D_{md}\right)\sum_d \lambda_d \geq \sum_d \lambda_d D_{md}$	$q_m\left(1-\sum_d \lambda_d D_{md}\right) \geq \sum_d \lambda_d D_{md}$
Class c	//	$\sum_m \frac{D_{mc}}{1-\sum_d \lambda_d D_{md}} \leq b_c$	$\lambda_c \sum_m \frac{D_{mc}}{1-\sum_d \lambda_d D_{md}} \leq q_c$
System	//	$\sum_m \frac{\sum_d \lambda_d D_{md}}{1-\sum_d \lambda_d D_{md}} \leq b_S\left(\sum_d \lambda_d\right)$	$\sum_m \frac{\sum_d \lambda_d D_{md}}{1-\sum_d \lambda_d D_{md}} \leq q_S$

Table 4. Constraints. Equations in gray cells are of non-linear constraints.

Constraint expressions are given in Table 4. For five of the performance indexes shown in Table 4, (the one with the white background), the corresponding constraints are linear in λ_c. This means that when only such constraints are present, Eq. 16 is a LPP which can be efficiently and accurately solved using the simplex algorithm. In presence of the other constraints (the one with gray background in Table 4), non linear optimization techniques should be employed. However, since all the functions are convex, the problem can still be solved using simple techniques such as Successive Quadratic Programming (SQP) [15].

As an example, we can use the result to investigate the effect of combining several different constraints on the total throughput of the system described in Section 4.In particular, we set the following requirements:

S1: The utilizations of the (**S1a**) *Inventory* and (**S1b**) *Shipping* resources should be less than 70%.

S2: The requests of the *Intranet* class should be executed with a mean transfer time less than 650 ms.

S3: We mush have $\beta_3 = 0.2$.

The maximum throughput with the given set of constraints corresponds to the solution of the following OP:

$$
\begin{array}{llll}
\textbf{Variables:} & \lambda_c,\ 1 \leq c \leq C & & \\
\textbf{Objective:} & \text{maximize} \sum_c \lambda_c & & \\
\textbf{Constraints:} & & & \\
\textit{1)} & u_{r_3} - \sum_d \lambda_d D_{3d} \geq 0 & \textit{2)}\ u_{r_4} - \sum_d \lambda_d D_{4d} \geq 0 & \\
\textit{3)}\ b_1 - \sum_m \dfrac{D_{m1}}{1 - \sum_d \lambda_d D_{md}} \geq 0 & & \textit{4)}\ \lambda_3 - \beta_3(\sum_d \lambda_d) \geq 0 & \\
\textit{5)} & 1 - \sum_d \lambda_d D_{md} \geq 0, & 1 \leq m \leq M & \\
\textit{6)} & \lambda_c \geq 0, & 1 \leq c \leq C &
\end{array}
\qquad (17)
$$

The constraints in Eq. 17 are derived from those in Table 4 and expressed as inequalities greater than or equal to 0 to conform with OP conventions. Constant parameters u_{r_3} and u_{r_4} of the *1)* and *2)* constraints (which corresponds to constraints **S1a** and **S1b**) are both set to 0.7. In constraint *3)* (corresponding to **S2**) the Response time requirement of class 1 is set to $b_1 = 650$ ms, while constraint *4)* is corresponds to constraint **S3**. It is interesting to note that among all the constraints in Eq.17, *3)* (**S2**) is the only one non-linear. If we exclude it from the OP, the Eq. 17 becomes a LPP which can be solved very efficiently with the simplex algorithm. In this case, the maximum Λ is 9.1864 job/s. with $\beta_1 = 0.38857$, $\beta_2 = 0.41143$ and $\beta_3 = 0.20006$. If we consider the complete set of constraints, we can use the SQP technique non-linear optimization technique, to find a maximum $\Lambda = 8.7508$ jobs/s. with $\beta_1 = 0.35245$, $\beta_2 = 0.44755$ and $\beta_3 = 0.2$. Results were computed using GNU Octave [16] on a standard laptop PC in few seconds.

5.4 Computational complexity

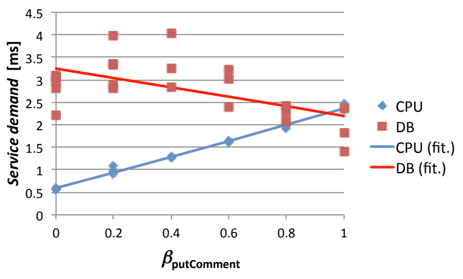
The proposed performance bounds can be computed very efficiently. For a considered population mix $\boldsymbol{\beta}$, the population mix scaled demand $D_m(\boldsymbol{\beta})$, for all the resources $1 \leq m \leq M$, can be computed with time complexity $O(M \cdot C)$, and storage complexity $O(M)$. $D_m(\boldsymbol{\beta})$ can then be used to compute the population mix maximum throughput $\hat{\Lambda}(\boldsymbol{\beta})$ with complexity $O(M)$.
Knowing the population mix scaled demand, resource-level constraints can be obtained in $O(1)$ time, since they do not include any iterative procedure and can be computed with closed form expressions.
System-level constraints are more complex since they must be computed iteratively. However, thanks to the Newton-Raphson, usually less than ten iterations are enough to reach a solution within the desired precision. During each iteration, both the value of the performance index and of its derivative must be computed: the complexity of both operation is $O(M)$ since these expressions iterate over all the system resources.

The most time-consuming operation is thus the computation of $D_m(\boldsymbol{\beta})$, which however could be cached and reused if several constraints have to be computed (i.e., a set of J constraints must be satisfied at the same time). The final computational complexity is $O(M \cdot \max\{C, J\})$, which gives the possibility to explore very large parameter spaces in term of different $\boldsymbol{\beta}$, even when considering a large number of classes, resources and performance objectives.
The convergency to the optimal solution of the LPP to determine either the unconstrained maximum throughput, or to consider linear constraint is also not an issue. The LPPs have always C variables and $M + \#LK + C$ (LP) constraints, where $\#LK$ is the total number of linear constraints. If the problem includes also $\#LK$ non-linear constraints, then the OP has a total of $M + \#LK + \#NLK + C$ (OP) constraints, which can still be handled by todays commodity hardware, provided that the total number of classes or resources is not extremely high.

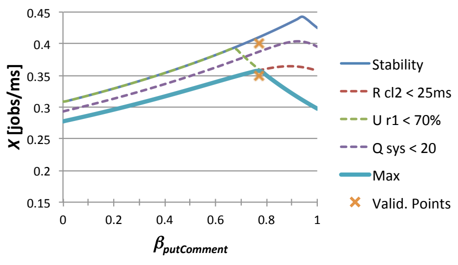
6 Experimental results

To show the applicability of the proposed technique in a real scenario, we applied it to study the maximum throughput of the RUBiS [5] benchmark application. RUBiS is a prototype of an auction site that mimics eBay, which is available in three different technologies: Java servlets, PHP, and Enterprise Java Bean. We used the servlet version of the benchmark, which is organized as a three-tier architecture using standard HTML, Java Servlet, and SQL technology. RUBiS comes with Apache server as the web server, JBoss as application server and MySQL as database. Each tier was deployed on a different physical machine equipped with a single core Intel Xeon processor running at 2.66 GHz with Ubuntu 12.04 LTS; the client emulator and the load balancer were deployed on servers running Microsoft Windows Server 2008 R2. A dedicated gigabit LAN provided the network functionality.
Since servlets' execution times are mostly related with the queries sent to the database, and with the amount of data returned, we focused our study on two servlets with fixed queries: *ViewBidHistory* and *PutComment*. Even though RUBiS is composed of a larger set of servlets, we selected these two as representatives of two types of requests: the application server intensive and the database intensive type, respectively. We consider this not to be an over-simplification; on the contrary, it is quite common in other related works as well, where techniques such as K-means clustering are commonly used to group requests into fewer clusters with similar profiles e.g. [18]. The technique proposed in Sec. 4 and Sec. 5 requires the determination of the demands of the considered classes at the resources that compose the system. In order to estimate such parameters, we studied the system by performing a set of test workloads. In particular, we loaded our client emulator with $\Lambda \in \{150, 200, 250, 350, 400\}$ job/s. and $\beta_1 \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. Not all the configurations were stable: in particular we experienced a large number of requests being dropped for $\Lambda = 350$ job/s. and $\beta_1 = 0$, and for $\Lambda = 400$ job/s. and $\beta_1 \leq 0.6$. We then discarded the ones for which we experienced drops in the system. We used the JBoss' and database's

mean service times to determine the demands that better describes our system through a simple fitting procedure: we minimized the difference between the service time expected by the model and the one measured on the real system. Figure 2a shows both the model and the measures: as it can be seen, we have very small errors for the JBoss component, while the DB experiences a larger deviation. Table 5a shows the estimated service times.



a.



b.

Fig. 2. (a.) Estimated vs. measured mean service demands and (b.) Maximum throughput of the considered RUBiS application.

D	*PutComment*	*viewBidHistory*
JBoss	0.5855	2.3532
DB	3.2463	2.1959

a.

Λ [job/sec]	$R_{ViewBidHistory}$ [ms.]	U_{JBoss}	Q_{sys}
350	11.0	55%	11.9
400	55.0	74%	373.6

b.

Table 5. (a.) Estimated demands of the considered RUBiS servlets (in msec) and (b.) Performance indices of the RUBiS application at $\boldsymbol{\beta}^*$.

By applying the values from Table 5a to Eq. 5, we determined that the systems is capable of offering a maximum throughput $\Lambda^* = 443.5$ jobs/s for a population mix of $\boldsymbol{\beta}^* = |0.0558,\ 0.9442|$. In particular, in Fig. 2b we can see that the system should be stable for all the possible population mixes for $\Lambda < 308$ jobs/s, while it will present instabilities for some $\boldsymbol{\beta}$ for $\Lambda > 308$. These results were confirmed by measurements, where we found the system being unstable at $\Lambda = 350$ job/s. for $\beta_{ViewBidHistory} = 1$, and at $\Lambda = 400$ job/s. for $\beta_{ViewBidHistory} \leq 0.4$.
We then adds three constraints to the system:

S2: The requests of the *ViewBidHisotry* class should be executed with a mean transfer time less than 25 ms.

S2: The utilizations of the *JBoss* resources should be less than 70%.

S3: The mean number of jobs in the system should be not greater than 20.

Figure 2b shows the effect of the constraints to the maximum throughput for various values of the class mix β. Using the optimization procedure described in Sec. 5.3, we can determine that the maximum throughput is $\Lambda^* = 358.34$ jobs/s. for a population mix of $\boldsymbol{\beta}^* = |0.2262,\ 0.7738|$. To check such requirements, we measured the system at $\boldsymbol{\beta}^* = |0.2262,\ 0.7738|$ for $\Lambda_{low} = 350$ job/s. and $\Lambda_{high} = 400$ job/s. The system was stable for both arrival rates. The results are shown in Table 5b: as it can be seen, all the constraints are met at Λ_{low}, but they are all violated for Λ_{high}, even if the system is still stable.

7 Conclusions

In this paper we described a technique to identify the maximum throughput that a system can provide, given an SLA for each class of applications. Limiting values of several performance metrics were considered, i.e., response times, utilizations, and queue lengths, and the maximum throughput for each class was computed. We demonstrated that the predicted values can be obtained in a real environment through experiments executed on a commonly adopted benchmark that simulates an e-commerce web site.
The future work will be concentrated on policies for the performance control of each node of a large set of interconnected systems in order to maximize the throughput of the global network.

References

1. Anselmi, J., Cremonesi, P., Amaldi, E.: On the consolidation of data-centers with performance constraints. In: Architectures for Adaptive Software Systems, Lecture Notes in Computer Science, vol. 5581, pp. 163–176. Springer Berlin Heidelberg (2009)
2. Ayoub, R., Ogras, U., Gorbatov, E., Jin, Y., Kam, T., Diefenbaugh, P., Rosing, T.: Os-level power minimization under tight performance constraints in general purpose systems. In: Low Power Electronics and Design (ISLPED) 2011 International Symposium on. pp. 321–326 (2011)
3. Balbo, G., Serazzi, G.: Asymptotic analysis of multiclass closed queueing networks: Multiple bottlenecks. Performance Evaluation 30(3), 115–152 (1997)
4. Barford, P., Crovella, M.: Generating representative web workloads for network and server performance evaluation. SIGMETRICS Perform. Eval. Rev. 26, 151–160 (June 1998), `http://doi.acm.org/10.1145/277858.277897`
5. Benchmark, R.: http://rubis.ow2.org/
6. Casale, G., Serazzi, G.: Bottlenecks identification in multiclass queueing networks using convex polytopes. In: Proc. of IEEE MASCOTS Symposium. pp. 223–230. IEEE Press (2004)
7. Cherkasova, L., Phaal, P.: Session-based admission control: A mechanism for peak load management of commercial web sites. IEEE Trans. Comput. 51, 669–685 (June 2002), `http://dx.doi.org/10.1109/TC.2002.1009151`
8. Eager, D.L., Sevcik, K.C.: Bound hierarchies for multiple-class queuing networks. J. ACM 33, 179–206 (January 1986), `http://doi.acm.org/10.1145/4904.4992`

9. Elnikety, S., Tracey, J., Nahum, E., Zwaenepoel, W.: A method for transparent admission control and request scheduling in e-commerce web sites. In: Proceedings of the 13th WWW. pp. 276–286 (2004)
10. Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S.: Monitoring, prediction and prevention of sla violations in composite services. In: (ICWS), 2010. pp. 369 –376 (july 2010)
11. Litoiu, M.: A performance analysis method for autonomic computing systems. ACM Trans. Auton. Adapt. Syst. 2 (March 2007), `http://doi.acm.org/10.1145/1216895.1216898`
12. Liu, Z., Squillante, M.S., Wolf, J.L.: On maximizing service-level-agreement profits. In: Proceedings of EC '01. pp. 213–223. ACM, New York, NY, USA (2001), `http://doi.acm.org/10.1145/501158.501185`
13. Mars, J., Tang, L.: Whare-map: heterogeneity in "homogeneous" warehouse-scale computers. In: ISCA '13. ACM, New York, NY, USA (2013), `http://doi.acm.org/10.1145/2485922.2485975`
14. Menascé, D.A., Almeida, V.A.F., Fonseca, R., Mendes, M.A.: Business-oriented resource management policies for e-commerce servers. Perform. Eval. 42, 223–239 (October 2000), `http://dx.doi.org/10.1016/S0166-5316(00)00034-1`
15. Nocedal, J., Wright, S.J.: Numerical optimization. Springer, New York (2006), `http://site.ebrary.com/id/10228772`
16. Octave, G.: http://www.gnu.org/software/octave/
17. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical recipes in C (2nd ed.): the art of scientific computing. Cambridge University Press, New York, NY, USA (1992)
18. Singh, R., Sharma, U., Cecchet, E., Shenoy, P.: Autonomic mix-aware provisioning for non-stationary data center workloads. In: ICAC '10. p. 2130. ACM, New York, NY, USA (2010), `http://doi.acm.org/10.1145/1809049.1809053`
19. Urgaonkar, B., Chandra, A.: Dynamic provisioning of multi-tier internet applications. In: Proceedings of the Second International Conference on Automatic Computing. p. 217228. ICAC '05, IEEE Computer Society, Washington, DC, USA (2005), `http://dx.doi.org/10.1109/ICAC.2005.27`
20. Villela, D., Pradhan, P., Rubenstein, D.: Provisioning servers in the application tier for e-commerce systems. ACM Trans. Internet Technol. 7(1) (Feb 2007), `http://doi.acm.org/10.1145/1189740.1189747`
21. Walsh, W.E., Tesauro, G., Kephart, J.O., Das, R.: Utility functions in autonomic systems. In: In Proceedings of the International Conference on Autonomic Computing. pp. 70–77 (2004)
22. website, T.A.E.: http://aws.amazon.com/ec2/#instance
23. Website, T.I.S.C.: http://www.ibm.com/cloud-computing/us/en/
24. Website, T.M.A.: http://www.microsoft.com/windowsazure/
25. Xu, X., Yu, H., Cong, X.: A qos-constrained resurce allocation game in federated cloud. In: IMIS 2013. pp. 268–275 (2013)
26. Xue, J.W.J., Chester, A.P., He, L., Jarvis, S.A.: Model-driven server allocation in distributed enterprise systems. In: ABIS '09 (2009)
27. Zhou, X., Ippoliti, D.: Resource allocation optimization for quantitative service differentiation on server clusters. Journal of Parallel and Distributed Computing 68(9), 1250–1262 (Sep 2008), `http://www.sciencedirect.com/science/article/pii/S074373150800107X`