RC24106 (W0611-050) November 10, 2006
Computer Science

IBM Research Report

SLA Based Resource Allocation Policies in Autonomic Environments

Danilo Ardagna
Politecnico di Milano
Dipartimento di Elettronica e Informazione

Marco Trubian
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Li Zhang
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Danilo Ardagna [*,1] Marco Trubian [2] Li Zhang [3]

SLA Based Resource Allocation Policies in Autonomic Environments

Abstract

Nowadays, large service centers provide computational capacity to many customers by sharing a pool of IT resources. The service providers and their customers negotiate utility based Service Level Agreement (SLA) to determine the costs and penalties on the base of the achieved performance level. The system is often based on a multi-tier architecture to serve requests and autonomic techniques have been implemented to manage varying workload conditions. The service provider would like to maximize the SLA revenues, while minimizing its operating costs. The system we consider is based on a centralized network dispatcher which controls the allocation of applications to servers, the request volumes at various servers and the scheduling policy at each server. The dispatcher can also decide to turn ON or OFF servers depending on the system load. This paper designs a resource allocation scheduler for such multi-tier autonomic environments so as to maximize the profits associated with multiple class SLAs. The overall problem is NP-hard. We develop heuristic solutions by implementing a local-search algorithm. Experimental results are presented to demonstrate the benefits of our approach.

Key words:
Autonomic computing, Resource Allocation, Load Balancing, Quality of Service, SLA Optimization

1 Introduction

To reduce their management cost, companies often outsource their IT infrastructure to third party service providers. Many companies, from hardware vendors to IT consulting, have set up large service centers to provide services

* Corresponding author
Email addresses: `ardagna@elet.polimi.it` (Danilo Ardagna), `trubian@dsi.unimi.it` (Marco Trubian), `zhangli@us.ibm.com` (Li Zhang).
[1] Politecnico di Milano, Dipartimento di Elettronica e Informazione
[2] Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano
[3] IBM T.J. Watson Research Center

to many customers by sharing the IT resources. This leads to the efficient use of resources and a reduction of the operating costs.
The service providers and their customers negotiate utility based Service Level Agreements (SLAs) to determine costs and penalties based on the achieved performance level. The service provider needs to manage its resources to maximize its profits. Utility based optimization approaches are commonly used for providing load balancing and for obtaining the optimal trade-off among request classes for Quality of Service levels (21).
One main issue of these systems is the high variability of the workload. For example for Internet applications, the ratio of the peak to light load is usually in the order of 300% (9). Due to such large variations in loads, it is difficult to estimate workload requirements in advance, and planning the capacity for the worst-case is either infeasible or extremely inefficient. In order to handle workload variations, many service centers have started employing self-managing autonomic techniques (3; 16; 18; 19).

Autonomic systems maintain and adjust their operations in the face of changing components, demands or external conditions and dynamically allocate resources to applications of different customers on the base of short-term demand estimates. The goal is to meet the application requirements while adapting IT architecture to workload variations (12). In such systems, a network dispatcher manages autonomic components (e.g. storage systems, physical servers, software elements) and dynamically determines the best use of resources on the base of a short term load prediction (see Figure 1). Usually, heterogeneous clusters of servers are considered and many technical solutions, e.g. grid and Web services (12; 11), or system virtualization (5), have been proposed to support application migration and resource management.

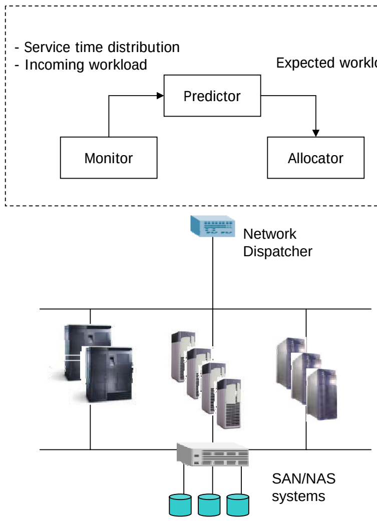

Fig. 1. Autonomic System Infrastructure

Independent on the technology used for the implementation, the main com-

ponents of the network dispatcher (7; 20) are a monitor, a predictor and a resource allocator. The system monitor measures the workload and performance metrics of each application, identifies requests from different customers and estimates requests service times. The predictor forecasts future system load conditions from load history and the allocator determines the best system configuration and applications to servers assignment. Usually, each request requires the execution of several applications allocated on multiple physical servers.
This paper focuses on the design of a resource allocator for autonomic multi-tier environments. The goal is to maximize the revenue while balancing the cost of using the resources. The cost includes the energy costs (9) and software and hardware cost of resources allocated on demand (25; 13). The overall profit (utility) includes the revenues and penalties incurred when Quality of Service guarantees are satisfied or violated. The resource allocator can establish: (i) the set of servers to be turned ON depending on the system load, (ii) the application tiers to servers assignment, (iii) the request volumes at various servers and (iv) the scheduling policy at each server.
We model the problem as a mixed integer nonlinear programming problem and develop heuristic solutions based on a local-search approach. The neighborhood exploration is based on a fixed-point iteration (FPI) technique, which iteratively solves a scheduling and a load balancing problem by implementing a gradient method. Our resource allocator considers two time scales: optimum load balancing and scheduling are determined in real-time and applied every few minutes, while application allocation and servers status are evaluated with a greater time scale, i.e. every half an hour, in order to reduce system re-configuration overhead.
Differently from the previous literature, our model considers jointly a broader set of control variables and uses mathematical programming based techniques to obtain a solution of the problem. Experimental results are presented to show the benefits of our approach.

The remainder of the paper is organized as follows. Section 2 describes other literature approaches. Section 3 introduces the overall system model. The optimization problem formulation is presented in Section 4. The local-search approach is presented in Section 5. The experimental results in Section 6 demonstrate the quality and efficiency of our solutions. Conclusions are drawn in Section 7.

2 Related Work

Recently, the problem of maximization of SLA revenues in shared service center environments implementing autonomic self-managing techniques has attracted vast attention by the research community. The SLA maximization problem which considers (i) the set of servers to be turned ON depending on the system load, (ii) the application tiers to servers assignment, (iii) the request volumes at various servers and (iv) the scheduling policy at each server as joint control variables, as faced in this paper, however has not yet been considered.

In (18; 19) authors address the problem of handling service centers resources in overload conditions while maximizing SLA revenues. Anyway, applications are assigned to dedicated servers and the load balancing problem is not addressed. Authors in (25) consider the optimization of a multi-tier system where the system workload is evenly shared among servers and the processor sharing scheduling policy is applied, i.e. control variables (iii) and (iv) are not taken into account. Furthermore, a single class model is considered and the problem is solved by enumeration.

When the scheduling and load balancing are used as control variables, the optimization of a single tier is usually addressed, mainly the Web server tier and the servers are always ON, hence decisions (i) and (ii) are not considered (23; 17; 8; 16). In particular, in (23) the problem of minimization of system response times and maximization of throughput is analyzed. The work proposes a *static* algorithm which assigns Web sites to overlapping servers executed once a week, on long term predictions basis, while a *dynamic* algorithm implements a real time dispatcher and assigns incoming requests to servers considering short term load forecasts.

In (17) continuous utility functions are introduced, the load is evenly shared among servers and the problem of maximization of SLA is formulated as a scheduling problem. The effectiveness of the overall approach is verified by simulation. The authors in (8) faced the dual problem of minimizing customers' discontent function considering an online estimate of service time requirements and their response times. The optimal Generalized Processor Sharing (GPS) scheduling policy (28) is identified by using Lagrange techniques. In (16), the authors proposed an analytical formulation of the problem to maximize the multi-class SLA in heterogeneous Web clusters, considering the tail distribution of the request response times. The control variables are the scheduling parameters at each cluster and the frequency of requests assigned to different clusters. The load balancing problem is addressed among different clusters but the load is evenly shared among serves of the same cluster.

Finally, in a previous paper (26), we extended (16) work by considering stepwise utility functions and the number of servers to be switched ON as a control variable. Anyway the pricing schema considered the response time of each request at a single tier. In multi tier system, the flexibility provided by current technology can be exploited to implement the most convenient load sharing among multiple machines, if the overall resource allocation problem is addressed.

3 The System Model

The system under study is a distributed computer system consisting of M heterogeneous physical servers. There are totally K classes of request streams. Each class $k \in K$ request can be served by a set of server applications (application tiers in the following) according to the client/server paradigm. For simplicity assume that each class k request is associated with a single customer. The architecture comprises a requests dispatcher in front of physical servers that estabilishes the allocation of application tiers to physical servers, the scheduling policy, and the load balancing of incoming requests to each physical server. The service discipline under consideration is the Generalized Processor Sharing (GPS) class (28). The controller can also turn OFF and ON physical server in order to reduce the overall cost.

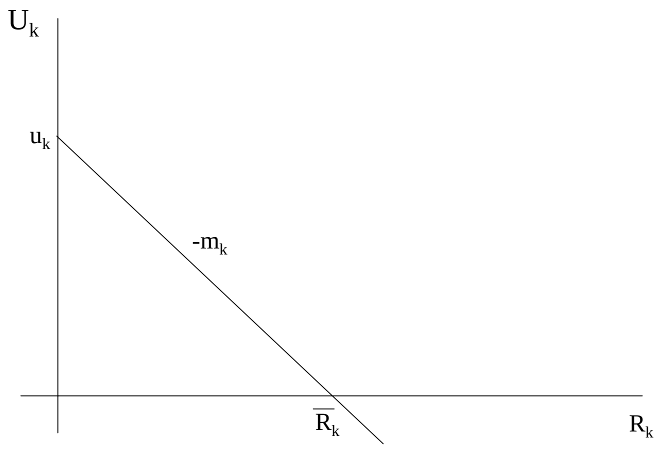

Fig. 2. Utility function used to evaluate per request revenues in terms of average requests response times R_k

For each class $k \in K$, a linear utility function is defined to specify the per request revenue (or penalty) incurred when the average end-to-end response time R_k, from multiple application tiers, assumes a given value. Figure 2 shows, as an example, the plot of an utility function. $-m_k$ indicates the slope of the utility function ($m_k = u_k/\overline{R}_k > 0$) and $\overline{R}_k$ is the threshold that identifies the revenue/penalty region (i.e., if $R_k > \overline{R}_k$ the SLA is violated and penalties are incurred). Linear utility functions are currently proposed in the literature (see for example (6; 18; 19)), anyway our approach can be extended in order to consider a broad family of utility functions. We only assume that the utility function is monotonically non-increasing, continuous and differentiable. Monotonic non-increasing utility functions are very realistic since the better the achieved performance by end users, the higher are the revenues gained per request by the Service Provider.

The overall system is modeled by a queueing network composed of a set of

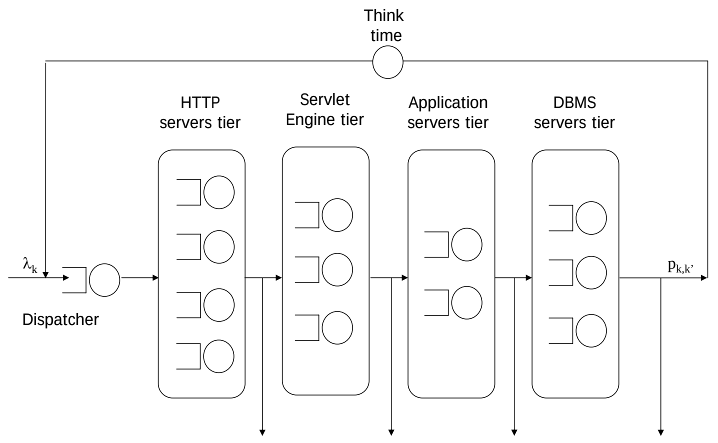

Fig. 3. Queueing network performance model of the autonomic system

multi-class single-server queues and a multi-class infinite-server queue. The first layers of queues represent the collection of physical servers supporting requests execution. The infinite-server queues represent the client-based delays, or think time, between the server completion of one request and the arrival of the subsequent request within a session (see Figure 3).
User sessions begin with a class k request arriving to the service center from an exogenous source with rate λ_k. The analysis of actual e-commerce site traces (see for example (22)) has shown that the Internet workload follows a Poisson distribution, hence we assume that the exogenous arrival streams are Poisson processes. Upon completion the request either returns to the system as a class k' request with probability $p_{k,k'}$ or it completes with probability $1 - \sum_{l=1}^{K} p_{k,l}$. Let Λ_k denote the aggregate rate of arrivals for class k requests, $\Lambda_k = \sum_{k'=1}^{K} \Lambda_{k'} p_{k',k} + \lambda_k$.

The service center can be characterized by the following set of parameters:

K	:=	set of request classes;
N_k	:=	number of application tiers involved in the execution of class k requests;
M	:=	number of physical servers at the service center;
C_i	:=	capacity of physical server i;
c_i	:=	time unit cost for physical server i ON;
$A_{i,k,j}$	:=	1 if physical server i can support the execution of application tier j for class k request, 0 otherwise;
$\mu_{k,j}$	:=	maximum service rate of a capacity 1 physical server for executing processes at tier j for class k requests.

Note that, different request classes require different number of application tiers N_k to be executed. For example, a request for a static Web page is executed by a Web server, while the request of dynamic Web page involves multiple tiers, from the Web server to the DBMS tier (see Figure 3). The routing matrix $[A_{i,k,j}]$ is usually obtained as a result of an optimization problem (23). It is used to assign private physical servers to different cutomers e.g. for dedicated e-commerce transaction servers. It can also limit the number of different Web sites assigned to physical servers due to caching issues (23). Thus, the routing matrix is used to limit the feasible assignments of application tiers to physical servers. Note that, $c_i \propto C_i$, if power is the main cost associated with turning ON a physical server.

The decision variables of our model are the followings:

x_i	:=	1 if physical server i is ON, 0 otherwise;
$z_{i,k,j}$	:=	1 if the application tier j for class k requests is assigned to physical server i, 0 otherwise.
$\lambda_{i,k,j}$	:=	rate of execution for class k requests at application tier j on physical server i;
$\phi_{i,k,j}$	:=	GPS parameter at physical server i for executing application tier j for class k requests;

The analysis of multi-class queueing system is notoriously difficult. We use the GPS bounding technique in (28) to approximate the queueing system. Under GPS, the physical server capacity devoted to class k requests for application tier j at time t (if any) is $C_i\phi_{i,k,j}/\sum_{k'\in\mathcal{K}(t)}\sum_{j=1}^{N_{k'}}\phi_{i,j,k'}$, where $\mathcal{K}(t)$ is the set of classes with waiting requests on physical server i at time t. Requests at different application tiers within each class and on every physical server are executed either in a First-Come First-Served (FCFS) or a Processor Sharing (PS) manner. Under FCFS, we assume that the service time for class k requests at physical server i has an exponential distribution with mean $(C_i\mu_{k,j})^{-1}$, whereas, under PS, service time of class k requests at physical server i follows a general distribution with mean $(C_i\mu_{k,j})^{-1}$, including heavy-tail distributions of Web servers. In the approximation, each multi-class single-server queue associated with an application tier is decomposed into multiple independent single-class single-server queues with capacity greater than or equal to $C_i\phi_{i,k,j}$. Authors in (28) have shown that under the hypothesis that $\lambda_{i,k,j} < C_i\phi_{i,k,j}$ and the external arrivals are exponentially boundend processes (and this hypothesis holds for Poisson arrivals, (24; 10)), in a network of arbitrary topology the performance metrics can be evaluated at any node of the queue network independently on the requests route. The response times evaluated in the isolated per-class queues are upper bounds on the corresponding measures in the original system. Under these hypothesis $R_{i,k,j}$, i.e., the average response time for the execution of the process at application tier j of class k requests at physical server i can be approximated by $R_{i,k,j} = \frac{1}{C_i\mu_{k,j}\phi_{i,k,j} - \lambda_{i,k,j}}$.

Note that, this approximation is asymptotically correct for high workloads, since at a high load if a requests class have been assigned to a physical server at least one request will be waiting for the execution and hence $C_i\phi_{i,k,j}$ is the capacity devoted for every request streams execution without any approximation. We adopt analytical models in order to obtain an indication of system performance and response time as the authors in (16; 29). In Section 6, resource allocation results will be validated by simulations considering also heavy-tail distributions for service time.

The average response time for class k requests is the sum of the average response times at each application tier computed over all physical servers, and is given by $R_k = \frac{1}{\Lambda_k}\left(\sum_{i=1}^{M}\sum_{j=1}^{N_k}\lambda_{i,k,j}R_{i,k,j}\right)$.

Our objective is to maximize the difference between revenues from SLAs and the costs associated with physical servers ON in the inter-scheduler time period which can be expressed as $\sum_{k=1}^{K}\Lambda_k(-m_kR_k + u_k) - \sum_{i=1}^{M}c_ix_i$, which, after substituting R_k, becomes:

$$\sum_{k=1}^{K}\left(-m_k\sum_{i=1}^{M}\sum_{j=1}^{N_k}\frac{\lambda_{i,k,j}}{C_i\mu_{k,j}\phi_{i,k,j} - \lambda_{i,k,j}}\right) + \sum_{k=1}^{K}u_k\Lambda_k - \sum_{i=1}^{M}c_ix_i$$

4 Optimization Problem

The overall optimization problem can be formulated as:

P1) $$\max_{x_i,\lambda_{i,k,j},\phi_{i,k,j}} f(\mathbf{x},\lambda,\phi) = \sum_{k=1}^{K}\left(-m_k\sum_{i=1}^{M}\sum_{j=1}^{N_k}\frac{\lambda_{i,k,j}}{C_i\mu_{k,j}\phi_{i,k,j} - \lambda_{i,k,j}}\right) - \sum_{i=1}^{M}c_ix_i$$

such that

$$\sum_{i=1}^{M}\lambda_{i,k,j} = \Lambda_k;\ \forall k, j \tag{1}$$

$$\sum_{k=1}^{K}\sum_{j=1}^{N_k}\phi_{i,k,j} \le 1;\ \forall i \tag{2}$$

$$\sum_{(k,j)\in\mathcal{B}_l} z_{i,k,j} \le 1;\ \forall i, l \tag{3}$$

$$z_{i,k,j} \le A_{i,k,j}x_i;\ \forall i, k, j \tag{4}$$

$$\lambda_{i,k,j} \le \Lambda_k z_{i,k,j};\ \forall i, k, j \tag{5}$$

$$\begin{aligned} \lambda_{i,k,j} &< C_i \mu_{k,j} \phi_{i,k,j};\ \forall i,k,j \qquad (6)\\ \lambda_{i,k,j}, \phi_{i,k,j} &\geq 0;\ \forall i,k,j \\ x_i, z_{i,k,j} &\in \{0,1\}\ \forall i,k,j \end{aligned}$$

In the above objective function we have omitted the term $\sum_{k=1}^{K} u_k \Lambda_k$ since it does not depend on the decision variables. Constraint family (1) entails that the traffic assigned to individual physical servers, and for every application tier, equals the overall load predicted for class k requests. Constraint family (2) expresses the bounds for GPS scheduling parameters. Constraint family (3) is introduced in order to assign distinct physical servers to subset of applications, where $\mathcal{B}_l$ is a subset of the indexes in $N_k \times K$. For example, a servlet engine can be executed with a Web server or an application server instance, vice versa application and DBMS servers are usually allocated to individual phisycal servers (i.e., eventually supporting multiple application or DBMS instances) for management and security reasons. The constraint family (4) allows assigning application tiers to physical severs according to $A_{i,k,j}$ and only if servers are ON. Constraint family (5) allows executing request k at server i only if the application tier j has been assigned to server i. Note that, for a given request class k, the overall load Λ_k is the same at every application tier. Finally, constraints family (6) guarantees that resources are not saturated.

Observation 1. Problem P1) is a mixed integer nonlinear programming problem. Even if the set of server ON is fixed, i.e. the value of variables x_i has been determined, the joint scheduling and load balancing problem is difficult since the objective function is neither concave nor convex. In fact, it is possible to prove by diagonalization techniques, that the eigenvalues of the Hessian of the cost function are mixed in signs (see (4)).

Observation 2. Constraint family (3) defines implication constraints. In a preprocessing phase, implication (or logical) constraints can be strengthened by dedicated constraint programming tools or by standard integer programming tools to obtain stronger formulations. E.g., let us suppose that a Web server (tier 1) can share a machine only with a servlet engine instance (tier 2), while the application server (tier 3) and the DBMS server (tier 4) are allocated to individual physical server. The servlet engine instance can share servers also with the application server. Such a situation can be modelled by the following set of equations: $z_{i,k,1} + z_{i,k,3} \leq 1$; $z_{i,k,1} + z_{i,k,4} \leq 1$; $z_{i,k,2} + z_{i,k,4} \leq 1$ and $z_{i,k,3} + z_{i,k,4} \leq 1$. Adding the first, second and fourth constraint we obtain the inequality $2z_{i,k,1} + 2z_{i,k,3} + 2z_{i,k,4} \leq 3$, which is equivalent to $z_{i,k,1} + z_{i,k,3} + z_{i,k,4} \leq 1$ since z-s are binary. This last equations entails that only one process among Web, Application and DBMS servers can be active on the same machine.

5 Optimization Technique

As it will be discussed in the experimental results section, the given problem can be solved by nonlinear commercial tools only for small size instances. For all instances of interest, an heuristic approach has to be considered. We resolve the problem in four steps:

(1) we estimate $\Gamma_{k,j}$ the value of the service center capacity to be provided to each application tier;
(2) we build a feasible solution which identifies an initial set of servers ON (by setting x_i variables), assigns application tiers to physical servers (by setting $z_{i,k,j}$ variables), and identifies an initial scheduling and load balancing for system requests (by setting $\phi_{i,k,j}$ and $\lambda_{i,k,j}$ variables). $\Gamma_{k,j}$ are used to initialize the $\phi_{i,k,j}$ values;
(3) a fixed point iteration (FPI) based on the sub-gradient method is then applied in order to improve ϕ-s and λ-s. This technique iteratively identifies the optimum value of a set of variables (λ-s or ϕ-s), while the value of the other one (alternatively ϕ-s or λ-s) is hold fixed;
(4) the obtained solution is finally enhanced with a local-search algorithm which turns on and off servers, modifies the assignment of application tiers to physical servers and update the scheduling (ϕ-s) and load balancing (λ-s).

The optimization problem P1) (steps (1)-(4)) is solved periodically. The time period T_{long} is in the order of magnitude of several minutes, e.g. 15-30 minutes which are mainly due by step (4). The optimum scheduling and load balancing (step (3) above) are evaluated in a shorter time scale $T_{short} \ll T_{long}$ and are applied every few minutes (e.g., 5 minutes). The two optimization problems (P1) and the joint optimum scheduling and load balancing) are solved by considering the workload prediction for the next control time interval (T_{long} and T_{short}, respectively) and their solutions are applied only if the variation of the objective function value which can be obtained by applying the new system configuration is greater than a given threshold. As proposed in (2), in this way system re-configuration overhead is reduced and the system will be stable, i.e. the system will not oscillate between two equilibrium points if the incoming workload will change slightly during the time control horizon.

5.1 Application Tiers Capacity Estimation

In order to estimate the value of the service center capacity to be provided to each application tier, we model the service center as a single physical server of capacity U. We consider the problem of minimizing the weighted average of response times and the cost associated with resource use as a function of $\Gamma_{k,j}$, the computing capacity assigned to the execution of request k at application tier j. Because the utility function is linear, the minimization of the weighted average of response times is equivalent to the maximization of SLA revenues. In this way we consider the trade-off between the SLA revenues and the costs of use of resources assigned to each application tier.

The optimization problem can be formalized as:

P2) $$\min_{\Gamma_{k,j}} \frac{1}{\Lambda'} \sum_{k=1}^{K} \sum_{j=1}^{N_k} \frac{m_k \Lambda_k}{\Gamma_{k,j} - \Lambda_k} + \mathcal{C} \sum_{k=1}^{K} \sum_{j=1}^{N_k} \Gamma_{k,j}$$

such that

$$\sum_{k=1}^{K} \sum_{j=1}^{N_k} \Gamma_{k,j} \leq U \tag{7}$$
$$\Lambda_k < \Gamma_{k,j} \qquad \forall k, j$$

The decision variables are $\Gamma_{k,j}$. $\mathcal{C} = \frac{1}{M} \sum_{i=1}^{M} \frac{c_i}{C_i}$ is the mean cost per unit resource use, U is the overall capacity of the service center which is estimated as $U = \frac{\sum_{i=1}^{M} C_i \sum_{k=1}^{K} \Lambda_k N_k}{\sum_{k=1}^{K} \Lambda_k (\sum_{j=1}^{N_k} 1/\mu_{k,j})}$ by applying a single class model as proposed in (14), and $\Lambda' = \sum_{k=1}^{K} \Lambda_k$ is the overall service center workload.

Equation (7) entails that the capacity assigned to application tiers is lower than the overall capacity of the service center. Since the cost function is continuous and convex (the Hessian is a diagonal matrix with elements $\frac{2 m_k \Lambda_k}{(\Gamma_{k,j} - \Lambda_k)^3} > 0$) the global optimum can be found in polynomial time. In (4), we have determined closed formulas to evaluate $\Gamma_{k,j}$ by appying Karush-Kuhn-Tucker conditions, and we have obtained an ineresting inequality that relates the overall load and the cost of physical servers to the overall capacity of the service center: $\mathcal{C} > \frac{1}{\Lambda'} \left(\frac{\sum_{k=1}^{K} N_k \sqrt{m_k \Lambda_k}}{U - \sum_{k=1}^{K} N_k \Lambda_k} \right)^2$. If this inequalities holds, then SLA profits cannot counter balance the cost of use of service center resources and some physical servers should be turned OFF. With this technique $\Gamma_{k,j}$ can be computed by performing $O(K)$ operations, for further details see (4).

5.2 Application Tiers to Servers Assignment

We now consider the problem of the assignment of application tiers to physical servers. The problem is NP-hard. This can be proved by a reduction from the Capacitated Facility Location Problem (see (4)). We have implemented a greedy algorithm for obtaining an initial solution which will be eventually enhanced by the local-search algorithm in the final step.

We first sort the physical servers according to the non decreasing cost-over-capacity ratio and the application tiers according to the non increasing number of different sets $\mathcal{B}_l$ they belong to. Then, each application tier is assigned to physical servers in the given order until the corresponding capacity requirement $\Gamma_{k,j}$ is satisfied, while respecting the constraints family (3) in problem P1). When a single physical server cannot satisfy an application tier's demand, then the application tier is split into multiple physical servers. Let us denote with $y_{i,k,j}$ the values returned by the algorithm. They define, for each combination of the i, j and k indices, the capacity of physical server i devoted to the execution of application tier j of request class k. $y_{i,k,j} > 0$ implies $z_{i,k,j} = 1$, $x_i = 1$, i.e. server i is turned ON, and $\sum_i y_{i,k,j} = \Gamma_{k,j}$. The CFLP problem is depicted in Figure 4 where application tiers and servers are modeled as a bipartite graph. The overall complexity is $O(ML + L\log L + M\log M)$, where $L = \sum_{k=1}^{K} N_k$, under the worst case hypothesis that at each iteration for every application tier the last physical server of the set has to be turned ON and physical servers are never saturated (which always requires to consider physical server compatibility with the current application tier $\Gamma_{k,j}$ to be allocated).

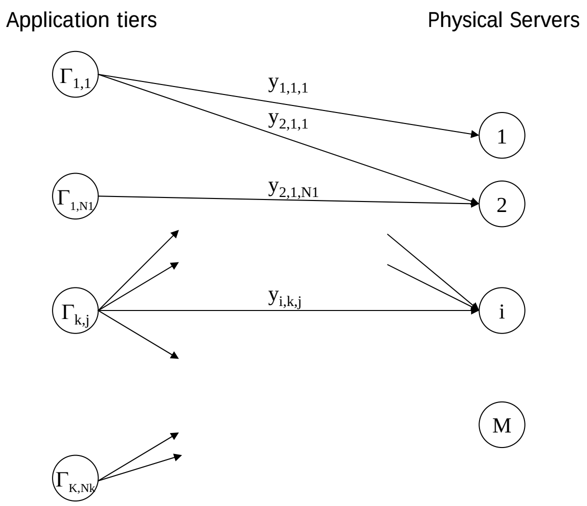

Fig. 4. CFLP formalization of the application tiers to physical servers allocation problem. Nodes on the left represent the application tiers capacity that have to be provided by the servers represented as nodes on the right

5.3 The Load Balancing and Scheduling Problems

Once the application tiers are assigned to physical servers, the scheduling policy at each physical server and the load balancing policy have to be identified. Let $\mathcal{I} = \{i | x_i = 1\}$ denote the set of physical servers ON and let $\overline{z}_{i,k,j} = 1$

if $y_{i,k,j} > 0$, as determined by the solution of the previous sub-problems. The joint scheduling and load balancing problem can be modeled as follows:

P3) $$\min_{\phi_{i,k,j},\lambda_{i,k,j}} \sum_{k=1}^{K} m_k \sum_{i\in\mathcal{I}} \sum_{j=1}^{N_k} \frac{\lambda_{i,k,j}}{C_i\mu_{k,j}\phi_{i,k,j}-\lambda_{i,k,j}}$$

such that

$$\begin{aligned}
\sum_{i\in\mathcal{I}} \lambda_{i,k,j} &= \Lambda_k;\ \forall k,j \\
\sum_{k=1}^{K}\sum_{j=1}^{N_k} \phi_{i,k,j} &\le 1;\ \forall i\in\mathcal{I} \\
\lambda_{i,k,j} &\le \Lambda_k \overline{z}_{i,k,j};\ \forall i\in\mathcal{I}, \forall k,j \\
\lambda_{i,k,j} &< C_i\mu_{k,j}\phi_{i,k,j};\ \forall i\in\mathcal{I}, \forall k,j \\
\lambda_{i,k,j}, \phi_{i,k,j} &\ge 0;\ \forall i\in\mathcal{I}, \forall k,j
\end{aligned}$$

where the decision variables are $\lambda_{i,k,j}$ and $\phi_{i,k,j}$. Note that the goal is to minimize the weighted average response time of request classes. As discussed in Section 5, P3) is solved periodically with time period T_{short}. We applied a FPI technique to obtain a solution. This approach iteratively identifies the optimum value of a set of variables (λ-s or ϕ-s), while the value of the other one (alternatively ϕ-s or λ-s) is hold fixed.

If the scheduling policy at every physical server is fixed, i.e. the $\phi_{i,k,j}$ variables are fixed to the values $\overline{\phi}_{i,k,j}$, then the problem is separable and $\sum_{k=1}^{K} N_k$ load balancing sub-problems (one for every application tier of every request class) can be solved independently. As the objective function is convex (the Hessian is given by $Diag\left(\frac{2m_kC_i\mu_{k,j}\overline{\phi}_{i,k,j}}{(C_i\mu_{k,j}\overline{\phi}_{i,k,j}-\lambda_{i,k,j})^3}\right)$ and eigenvalues are positives), the optimal solution of each sub-problem can be identified. An initial solution for ϕ-s parameters can be obtained from the solution of the problem in section 5.2 by setting $\overline{\phi}_{i,k,j} = \frac{y_{i,k,j}}{C_i\mu_{k,j}}$.

Vice versa, if the load balancing is fixed, i.e. $\lambda_{i,k,j}$ variables are fixed to the values $\overline{\lambda}_{i,k,j}$, then the problem is separable and M scheduling sub-problems (one for each physical server) can be solved independently. Again, the objective function is convex (the Hessian is given by $Diag\left(\frac{2m_kC_i^2\mu_{k,j}2\overline{\lambda}_{i,k,j}}{(C_i\mu_{k,j}\phi_{i,k,j}-\overline{\lambda}_{i,k,j})^3}\right)$ and eigenvalues are positives), the optimal solution of each sub-problem can be identified.

The FPI iteratively solve the load balancing and scheduling problems. Although we can not guarantee that the procedure converges to a global optimum, we can state that the procedure will always converge. In fact, since the optimal solutions of each sub-problem can be identified, at each step, the current λ-s and ϕ-s assignment is improved and the algorithm will find a local optimal solution.

The optimal solution of each of the two sub-problems can be obtained by applying KKT conditions (see (4)). Here, we present a faster iterative solution based on the gradient method.

The solution of the load balancing problem (where a request of class k and its corresponding application tier j are fixed) is obtained by starting from a feasible solution s and performing the optimal re-allocation of request load between only two physical servers, say l and m, according to the gradient $\mathbf{g}$ of the objective function $f_1(\lambda_{l,k,j}, \lambda_{m,k,j}) = \frac{\lambda_{l,k,j}}{C_l \mu_{k,j} \overline{\phi}_{l,k,j} - \lambda_{l,k,j}} + \frac{\lambda_{m,k,j}}{C_m \mu_{k,j} \overline{\phi}_{m,k,j} - \lambda_{m,k,j}}$. Since we consider the minimization of the weighted average response times, we can improve the objective function value of the solution s by optimally balancing the load between the two physical servers with the maximum and minimum component in $\mathbf{g}$, respectively. Note that, the optimal solution of this balancing problem can be found by expressing the load at the two physical servers as a function of a single variable and by solving a second degree equation. In more detail, if $\overline{\Lambda} = \overline{\lambda}_{l,k,j} + \overline{\lambda}_{m,k,j}$ indicates the current load assignment to physical server m and l, the new assignment is obtained by solving the following problem:

$$\text{P4)} \qquad \min_{\lambda_{l,k,j}, \lambda_{m,k,j}} \frac{\lambda_{l,k,j}}{C_l \mu_{k,j} \overline{\phi}_{l,k,j} - \lambda_{l,k,j}} + \frac{\lambda_{m,k,j}}{C_m \mu_{k,j} \overline{\phi}_{m,k,j} - \lambda_{m,k,j}}$$

$$0 \le \lambda_l < C_l \mu_{k,j} \overline{\phi}_{l,k,j}; \tag{8}$$

$$0 \le \lambda_m < C_m \mu_{k,j} \overline{\phi}_{m,k,j}; \tag{9}$$

$$\lambda_{l,k,j} + \lambda_{m,k,j} = \overline{\Lambda} \tag{10}$$

Problem P4) can be solved by minimizing the convex function of a single variable $f_1(\lambda_{l,k,j}, \overline{\Lambda} - \lambda_{l,k,j})$, i.e. by solving the second degree equation $\frac{df_1}{d\lambda_{l,k,j}} = 0$. Note that, if the solution of the first order derivative is not feasible, then the optimal solution is either $\lambda_{l,k,j} = \overline{\Lambda}$ and $\lambda_{m,k,j} = 0$, or $\lambda_{l,k,j} = 0$ and $\lambda_{m,k,j} = \overline{\Lambda}$.

The algorithm stops when the improvement in the objective function value is less than 1% in two consecutive iterations. Using ad hoc data structure the algorithm can run with $O(M + N_g \ln M)$ complexity, where N_g denotes the number of fixed point iterations (see (4)).

Likewise, the solution of the scheduling problem at a physical server i is obtained by starting from a feasible solution s and evaluating the request scheduling parameters only for the classes which corresponds to the maximum (k_1, j_1)

and to the minimum (k_2, j_2) of the gradient of the objective function $\sum_{k=1}^{K}\sum_{j=1}^{N_k} m_k \frac{\overline{\lambda}_{i,k,j}}{C_i \mu_{k,j}\phi_{i,k,j} - \overline{\lambda}_{i,k,j}}$.

Again, the optimal solution can be found by minimizing a convex function of a single variable. The algorithm can run with $O(\sum_{k=1}^{K} N_k + N_g \ln \sum_{k=1}^{K} N_k)$ complexity (see (4)).

5.4 The Local Search Algorithm

The solution returned by the FPI technique is (possibly) improved by applying a local search algorithm, i.e., the FPI solution is the starting point of our local search.

Let S denote the set of the feasible solutions of our problem. To each $s \in S$ we associate a subset $N(s)$ of S, called neighborhood of s which contains all those solutions that can be obtained by applying four different kind of moves: turning ON a physical server, turning OFF a physical server, physical servers swapping, and re-allocation of application tiers to physical servers.

The above moves directly modify either the values of the x or the z variables. For each modification new optimal (or sub-optimal) λ-s and ϕ-s values have to be computed. If the FPI were performed at each move, then only local optimal states for the overall system would be obtained. Unfortunately, we can not run the fixed point procedure for every neighborhood candidate solution $s' \in N(s)$, because it is too time consuming even with the gradient implementation. To overcome this difficulty, we overestimate the value of the each candidate solution s' by updating the values of a restricted subset of the variables λ and ϕ. In fact, after the first execution of the FPI from the initial state obtained by the solution of problem in section 5.2, we can reasonably assume that the optimal load balancing and scheduling solutions are only perturbed by switching ON or OFF one physical server or by re-allocating applications. This hypothesis has been confirmed by computational experiments under low or medium load. When the local-search stops at a local optimum, we execute a full run of the FPI. In this way we try to escape from the local minimum by optimal updating all the λ and ϕ variables, instead of considering only a subset of them, as usually done in the neighborhood exploration. If this step modifies the λ or ϕ variables we restart the local search from the last FPI solution.

Note that, the local-search is not effective for high load. As we can expect a-priori the optimal solution uses all the physical servers available when the average utilization of the data center is greater than 50-60% (9; 26). Under high load conditions, only the optimal scheduling and load balancing policies have to be identified.

5.4.1 Turning OFF servers

All physical servers with average utilization U_i in $[\min U_i, P \cdot \min U_i]$ are candidates to be turned OFF. Here P is a constant experimentally set between 1.1 and 1.2. The load of the physical server, say $\hat{i}$, to be switched off is allocated on the remaining physical servers proportionally to their spare capacity. The spare capacity for the application tier j of class k request at physical server i is $S_{i,k,j} = C_i \mu_{k,j} \phi_{i,k,j} - \lambda_{i,k,j}$. Let be $S_{k,j} = \sum_{i \in \mathcal{I} - \{\hat{i}\}} S_{i,k,j}$ the overall spare capacity available at remaining physical servers ON for application tier j of class k. The load at physical server $\hat{i}$ is assigned to remaining physical servers according to the equation $\lambda_{i,k,j} = \lambda_{i,k,j} + \lambda_{\hat{i},k,j} \cdot \frac{S_{i,k,j}}{S_{k,j}}$. The spare capacity is zero for all physical servers that can not support application tier j for class k for constraints (3)-(5) of problem P1). The neighborhood exploration has complexity $O(M \cdot \sum_{k=1}^{K} N_k)$.

5.4.2 Turning ON servers

To alleviate the load at a bottleneck physical server, a physical server in OFF status is turned ON. All physical server whose utilization U_i is in $[P \cdot \max U_i, \max U_i]$, where P is a constant experimentally set between 0.9 and 0.95, are considered as bottleneck physical servers. The physical server turned ON applies the same scheduling policy adopted at the bottleneck physical server. For each bottleneck physical server the set of compatible physical servers in status OFF is identified. A physical server machine is compatible to a bottleneck physical server, if they are characterized by the same value of parameters $A_{i,k,j}$ limited to the set of application tiers and request classes currently executed at the bottleneck. The optimal load balancing among the two physical servers is identified by solving an instance of problem P4). The neighborhood exploration has complexity $O(M^2 \cdot \sum_{k=1}^{K} N_k)$.

5.4.3 Servers Swap

This move looks for a physical server i_1 in status ON and one i_2 in status OFF which are compatible and such that the physical server OFF has greater capacity or lower cost than the physical server ON. The load is moved from physical server i_1 to i_2, and at i_2 the same scheduling policy applied at i_1 is used, i.e., λ and ϕ variables are unchanged. Note that this move can not always be substituted by a sequence of switching ON and switching OFF moves. For example, if a single physical server i_1 is the only one supporting a class of requests k_1, then it will never be turned OFF. Furthermore, turning ON a

physical server i_2, which can support k_1, could worsen the objective function value since the enhancement of the system performance could not counterbalance the cost of i_2. Vice versa, the swap of i_1 and i_2 can be performed and the variation of the cost function can be evaluated accordingly.
Note that, since the set of swaps which are not considered must satisfy the following conditions: $C_{i_2} \leq C_{i_1}$ and $c_{i_2} \geq c_{i_1}$, only worsening moves are excluded. The neighborhood exploration has complexity $O(M^2 \cdot \sum_{k=1}^{K} N_k)$.

5.4.4 Re-allocation of Application Tiers to Servers

The aim of this move is to allow modifying the value of the z variables. We look for application tiers which can be allocated on a different set of physical servers with respect to those which they are currently assigned to. In fact, if the FPI sets the value of one λ to be 0, that λ will never change to a different value. Furthermore, in the next iteration, the corresponding ϕ is set to 0 and it will stay at 0 forever. Hence, the FPI can only de-allocates application tiers from physical servers. If an application tier is de-allocated from a physical server i, then the corresponding z variable is set to zero, and for constraints family (3) and (5), a tier which was not allowed to be executed on physical server i could now be allocated on it.
Before allocating an application tier to a physical server, we need to create spare capacity on that physical server. This is because for the optimal solution of problem P4) the sum of ϕ variables on each physical server equals to 1 (this can be verified by KKT conditions (4)). We consider the physical servers whose utilization U_i is lower than a constant $\hat{U}$ as candidate physical servers for hosting a new application tier. $\hat{U}$ is experimentally set to 0.6, since it is not easy to allocate another application tier on over-utilized physical servers and generate more profits.
Only one new application tier is allocated on a candidate physical server $\tilde{i}$. And the set of application tiers which can be possibly hosted on each candidate physical server is determined by an exhaustive search because only a small number of alternatives need to be evaluated. The spare capacity is created by setting $U_{\tilde{i}} = \hat{U}$, that is by setting $\phi'_{\tilde{i},k,j} \frac{\lambda_{\tilde{i},k,j}}{C_{\tilde{i}}\mu_{k,j}\hat{U}}$ for all application tiers currently executed on physical server $\tilde{i}$. The value $C_{\tilde{i}}\mu_{\tilde{k},\tilde{j}}\phi_{\tilde{i},\tilde{k},\tilde{j}}$ is the capacity available at physical server $\tilde{i}$ for the candidate application tier $(\tilde{k},\tilde{j})$, where $\phi_{\tilde{i},\tilde{k},\tilde{j}} = 1 - \sum_{k \in \mathcal{K}_{\tilde{i}}} \sum_{j=1}^{N_k} \phi'_{\tilde{i},k,j}$ and $\mathcal{K}_{\tilde{i}}$ is the set of request classes currently executed at physical server $\tilde{i}$. The candidate application tier will reduce the load of a bottleneck physical server $\hat{i}$ (i.e., such that $U_{\hat{i}} > \hat{U}$). The optimal load balancing among physical servers $\tilde{i}$ and $\hat{i}$ is computed by solving problem P4). Neighborhood exploration complexity is $O(M^2 \cdot (\sum_{k=1}^{K} N_k)^2)$.

6 Experimental Results

The effectiveness of our approach has been tested on a wide set of randomly generated instances. All tests have been performed on a 3 GHz Intel Pentium IV workstation. The number of physical servers has been varied between 40 and 400 (with steps of 40). Service centers up to 200 request classes have been considered and the number of application tiers has been varied between 2 and 4. $A_{i,k,j}$ values were randomly generated, and every physical server was shared by at most five customers (see Section 3). Service times were randomly generated and for each test case the load was increased in a way that the utilization of service center resources varied between 0.2 and 0.8. N_k, m_k and z_k values have been randomly generated, z_k is proportional to the number of application tiers N_k and to the overall demanding time at various tiers of class k request. m_k varied uniformly between 2 and 10.
Tests have been run by considering homogeneous (only physical servers of capacity 1) and heterogeneous systems (half physical servers with capacity 1 and half of capacity 2) where the load was evenly shared among different application tiers, or more realistically, in the second case higher application tiers were the system bottleneck.
Cost associated with physical servers have been obtained as in (26) by running our algorithm and considering the revenues obtained from service centers for increasing load. Results have shown that revenues increase almost linearly with the system load and start decreasing after a maximum that is obtained when the service center utilization is about 0.5-0.6. After the maximum, performance degrade and the increasing load implies a loss in revenues. The maximum revenue grows linearly with service center capacity with coefficient almost equal to 40. Hence, in our tests we used 15 as unit capacity cost.

Experiments have been performed to: (i) evaluate the performance of the FPI and local search procedures, (ii) compare the performance of our approach with general-purpose Mixed Integer Non-Linear Programming methods, (iii) evaluate the performance of the overall approach with respect to other solution proposed in the literature, (iv) evaluate the effectiveness of turning physical servers OFF, and (v) validate the results obtained with analytical model via simulation.

Figures 5 and 6 report, as an example, the trace of execution of the FPI and local search algorithm. Note that, at every iteration the FPI performs two steps, i.e. two gradient optimizations, one for ϕ variables (half iterations in Figure 5 plot) and one for λ-s. The example refers to a 400 physical servers 100 request classes on 4 application tiers. Plots are representative, usually the FPI converges very quickly and performs less than 10 iterations and the execution time is always lower than 8 secs. Local search usually performs a greater number of iterations but the execution is stopped when the execution

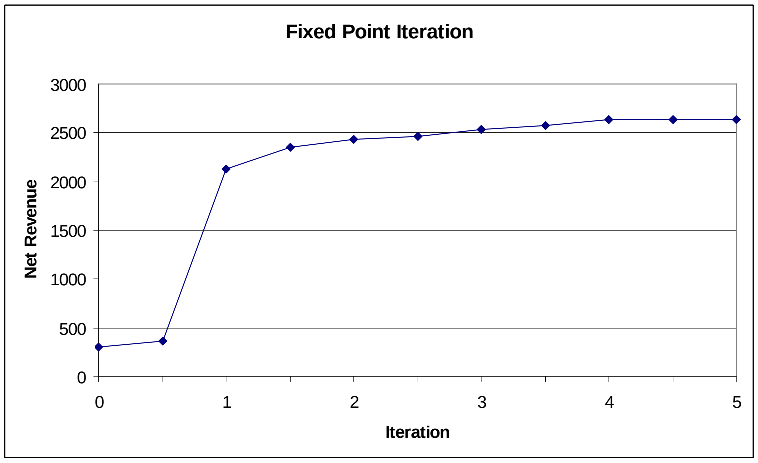

Fig. 5. FPI Execution Trace

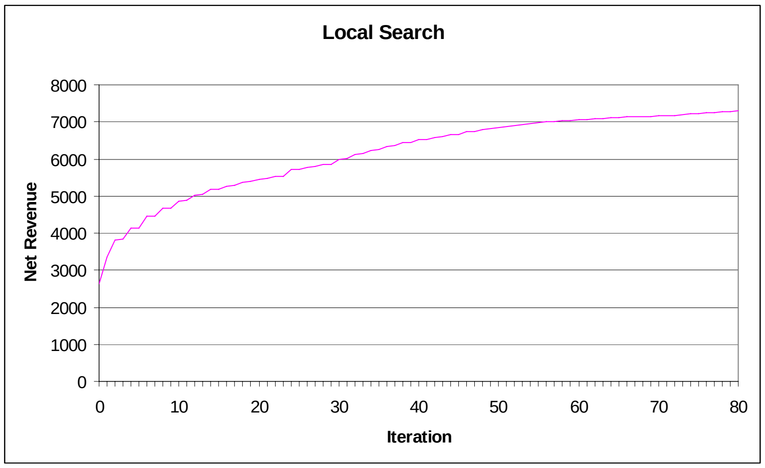

Fig. 6. Local Search Execution Trace

Table 1
Improvement achieved with respect to Proportional Assignment Schema

U	Loss in revenues PAS ($)	Loss in revenues ($)	% Improvement
0.2	25,320	3,796	5.67
0.3	39,120	4,696	7.33
0.4	57,476	6,575	7.74
0.5	83,045	8,419	8.86
0.6	121,052	9,430	11.84

time achieves 30 min.

As discussed previously, problem P1) is too complex to attempt to solve any meaningful instance via well-established general-purpose Mixed Integer Non-Linear Programming solution methods. We have nonetheless tried to solve it

Table 2
Improvement achieved in the Optimization Technique steps

	Homogeneous				Heterogeneous			
U	% IS	% FP	% Server	Time	% IS	% FP	% Server	Time
0.2	257.3%	229.1%	70%	30	300.1%	47.9%	67%	30
0.3	221.7%	90.2%	77%	26	33.6%	18.0%	73%	30
0.4	78.5%	54.2%	85%	22	41.8%	13.8%	77%	25
0.5	120.1%	60.2%	100%	28	29.1%	5.2%	89%	13
0.6	75.1%	39.2%	100%	18	22.0%	2.1%	97%	20
0.7	100.2%	19.3%	100%	12	15.3%	1.1%	100%	15
0.8	21.3%	15.4%	100%	5	115.1%	2.3%	100%	10

for very small test cases (6 physical servers and 6 class of requests on two application tiers) using various global optimization approaches. In particular, we used three global optimization solvers within the ooOPS global optimization framework (15): namely, spatial Branch-and-Bound (sBB), SobolOpt Multi Level Single Linkage, and Variable Neighborhood Search (VNS). All these methods gave very disappointing results due to the presence of a large number of integral variables. An attempt to solve the continuous (non-convex) relaxation was also carried out. For sBB and SobolOpt the formulation was too large to solve to optimality within a reasonable amount of time while VNS did locate a putative global optimum of the relaxed problem in 500s on average.

In order to compare our results with other approaches in the literature the number of physical servers that has to be turned ON is evaluated as the number of physical servers that keeps the utilization of the service center equals to 0.6 and the Proportional Assignment Schema (PAS) is applied to fix λ and ϕ variables. These approaches select the set of physical servers ON by implementing greedy algorithm based on utilization thresholds (see (9) and (1)). The PAS employs $\lambda_{i,k,j} = \Lambda_k \frac{C_i \mu_{k,j}}{\sum_{l=1}^{M} C_l \mu_{k,j}}$ and $\phi_{i,k,j} = \frac{\frac{\lambda_{i,k,j}}{\mu_{k,j}}}{\sum_{l=1}^{K} \frac{\lambda_{i,l,j}}{\mu_{l,j}}}$. Note that, this proportional allocation scheme is a natural way to assign the traffic and physical server capacity. It is provably the best load balancing scheme in terms of stability regions and it is used as a benchmark in the SLA profits maximization literature (16). Considering this scenario, our approach improves SLA revenues by one order of magnitude. In order to compute a meaningful estimation of the relative error of the solutions given by our approach and PAS, we need to add an offset value to the objective function and reverse it from maximum to minimum. Indeed, since the objective function is given by the difference between the total revenues and the total cost of the servers turned

ON, in some solution this difference can be zero or less than zero. This situation indeed occurs when using the PAS. Hence, the ratio between the values obtained with our algorithm and those obtained with PAS can result negative or infinity. We modified the objective function as follows. We computed, for each class, the maximum possible revenue, which corresponds to the minimum possible response time, i.e. $\sum_{j=1}^{N_k} 1/\mu_{k,j}$. Let us denote with G_{max} the summation of all the obtained revenues. The new objective function is now given by the difference between G_{max} and $f(\mathbf{x}, \lambda, \phi)$, the objective function of problem P1), see the beginning of Section 4. In other words we are now minimizing the cost of the used servers plus the loss in revenues. In this way the objective function value of any feasible solution is greater than zero, whereas the relative ranking among the solutions remains unchanged. Results of a homogeneous test case, grouped by the average service center utilization, are reported in Table 1. With the heterogenous case the PAS performs even worst and results are not reported. In practice, for the same load, our controller is able to reduce the number of physical servers ON. Furthermore, by inspecting optimal solutions we found that the load is not equally balanced among physical servers and physical servers are assigned to a limited number of request classes. This can be justified since dedicated physical servers give better performance. In our optimal solutions, physical servers are not fully dedicated to a single class of requests or to a single application tier since, in some situations, the physical server sharing among different application tiers (e.g., servlet engines with Web servers) or in the same application tier (e.g., serving multiple instances of different DBMS) can be exploited to obtain higher revenues.

In general, our resource allocator adopts all physical servers available at the service center when the load reaches about 50% of its capacity. When the load is light turning some physical server OFF allows us to obtain better results. In order to evaluate the effectiveness of turning physical servers OFF, we compared results that can be achieved by our resource allocator with results that can be obtained by turning all physical servers ON and adopting our optimal load balancing and scheduling policies (that is by applying the FPI). A total of 200 tests were considered. Turning physical servers OFF improves the cost function by about 35%, ranging form 44% and 22%, when service center utilization is 0.2 and, 0.4 respectively, exploiting the trade off between higher revenues (which can be obtained by turning all physical servers ON) and the costs associated with physical servers. As a typical example, Table 2 shows the average improvement which can be obtained from the initial solution (% IS column) and from the first FPI (% FP) by applying the local search approach for a test case with 400 physical servers. In this test case, 100 request classes are allocated on four application tiers, and 100 physical servers are assigned at each application tier by $A_{i,k,j}$. The last two columns report the fraction of physical servers adopted at the service center and the overall execution time (in minutes). Results are grouped by the average service center utilization. The values obtained by applying the proportional assignment schema are not reported here, since the solutions are never profitable, i.e., the values of the objective function are always negative. Test results show that the optimal solution for heterogeneous systems adopts a lower number of physical servers of the corresponding homogeneous case using mainly physical servers of higher capacity. This is expected since physical servers of grater capacity give better performance and hence better revenues despite their higher cost.

We have validated our solutions based on analytical models using a simulator which supports GPS policy (27). We generated arrival streams with different classes of interarrival times to investigate the effects of non-Poission arrivals. We also simulated request service times from the log-Normal and Pareto families, with the same mean and standard deviations. The regenerative simulation runs until a minimum number of regenerative cycles have been reached and the collected statistics from the servers and queues all reach the desired confidence level (95%). This level of confidence is reached for systems with about 10 servers and 10 job classes within several minutes. We run several simulations varying the tests parameters as described above. Analytical model and simulation results were always coherent independently on the size of the system. Results obtained by simulation are consistent with the behaviour one can expect since a better service is provided to more important (and more profitable) request classes. The plots reported in Figures 7 and 8 show simulation results of a representative example where a gold and a bronze request class are considered ($|m_{gold}| >> |m_{bronze}|$). The service time distribution is Pareto. In the control time interval the gold class load increases by 70% (Figure 7) the bronze class response time has a greater increase (about 20%) than that of the gold one (about 12%, see Figure 8).

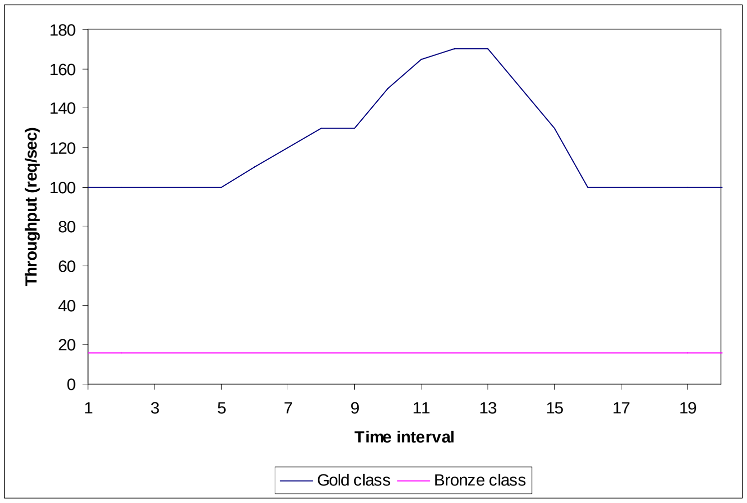

Fig. 7. Throughput variation considered in the simulation scenario

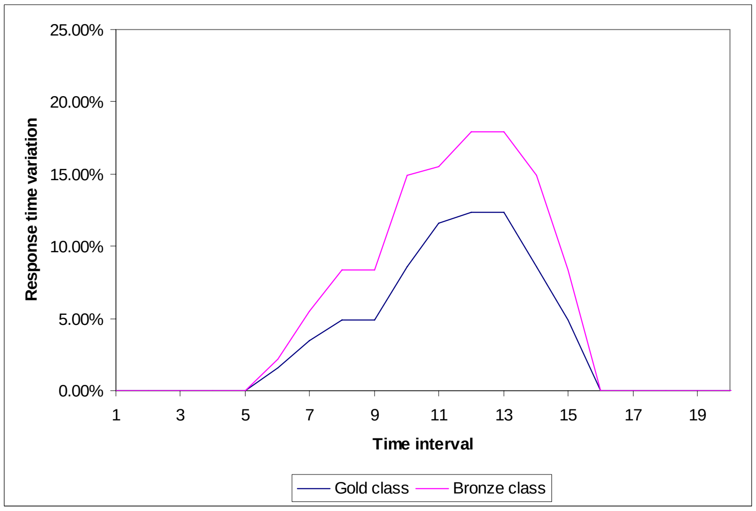

Fig. 8. Response time variation obtained by simulation

7 Conclusions

We proposed an allocation controller for multi-application tier service center environments which maximizes the profits associated with multi-class Service Levels Agreements. The cost model consists of a class of utility functions which include revenues and penalties incurred depending on the achieved level of performance and the cost associated with physical servers. The overall optimization problem which considers the set of physical servers to be turned ON, the allocation of applications to physical servers and load balancing and scheduling at physical servers as joint control variables, is NP-hard and we developed a heuristic solution based on a local search algorithm. Experimental results, up to 400 physical servers and 200 request classes, show that revenues that can be obtained with a proportional assignment schema can be significantly improved and important savings can be obtained on light and medium load conditions. Future work will introduce strict QoS performance guarantees, i.e., deadlines for the response times.

References

[1] Abdelzaher, T. F., Shin, T.,F., Bhatti, N. 2002. *Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach.* IEEE Trans. on Parallel Distr. Systems. 13, 1, 80-96.

[2] Aiber, S., Gilat, D., Landau, A., Razinkov, N., Sela, A., Wasserkrug, S. 2004. *Autonomic Self-Optimization According to Business Objectives.* In Proc of ICAC 2004.

[3] Appleby, K., Fakhoury, S., Fong, L., Goldszmidth, G., Kalantar, M., Krishnakumar, S., Pazel, D. P., Pershing, J., Rochwerger, B. 2001. *Oceano-*

SLA Based Management of a Computing Utility. In Proc. of the IFIP/IEEE Symp. on Int. Network Manag.

[4] Ardagna, D., Trubian, M., Zhang, L. 2004. *On Maximizing SLA in multi-tier Web Applications*. Politecnico di Milano Technical Report n. 2004.34.

[5] Dragovic, B. Fraser, K., Hand, S., Harris T., Ho, A., Pratt, I., Warfield, A., Barham, P., Neugebauer R. 2003 *Xen and the Art of Virtualization*. In Proc. of SOSP 2003.

[6] Bennani, M., Menascé D. 2004 *Accessing the Robustness of Self-Managing Computer Systems under Highly Variable Workloads*. In Proc. of ICAC 2004.

[7] Chandra, A., Goyal, P., Shenoy, P. 2003. *Quantifying the Benefits of Resource Multiplexing in On-Demand Data Centers*. In Proc. of the 1st Workshop on Algorithms and Architectures for Self-Managing Systems, San Diego, CA.

[8] Chandra, A., Gong, W., Shenoy, P. 2003. *Dynamic Resource Allocation for Shared service centers Using Online Measurements*. In Proc. of ACM SIGMETRICS 2003, Poster Session.

[9] Chase , J. S., Anderson, D. C. 2001 *Managing Energy and Server Resources in Hosting Centers*. In Proc. of the 18th ACM symposium on Operating systems principles.

[10] Ciucu, F., Burchard, A., Liebeherr, J. 2005. *A Network Service Curve Approach for the Stochastic Analysis of Networks*. In Proc. of ACM SIGMETRICS 2005.

[11] Foster, I., Kishimoto, H., Savva, A., Berry, D., Dijaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniaman, R., Treadwell, J., Von Reich, V. 2005 *The Open Grid Services Architecture*. http://www.globus.org/ogsa/, last accessed January 2006.

[12] Kephart, J. O., Chess, D. M. 2003. *The Vision of Autonomic Computing*. IEEE Computer 36(1):41-50.

[13] IBM. 1984 *IBM e-server zSeries 990*. www.ibm.com.

[14] Lazowska, E. D., Zahorjan, J., Graham, G. S., Kenneth, C. S. 1984 *Quantitative System Performance Computer system analysis using queueing network models*. Prentice-Hall.

[15] Liberti, L., Tsiakis, P., Keeping, B., Pantelides, C.C. 2002. *ooOPS*. Centre for Process Systems Engineering, Chemical Engineering Department, Imperial College, London, UK.

[16] Liu, Z., Squillante, M. S., Wolf, J. 2001 *On maximizing service-level-agreement profits*. In Proc. of the 3rd ACM conf. on Electronic Commerce.

[17] Shen, K., Tang, H., Yang, T. 2002. *A Flexible QoS Framework for Cluster-based Network Services*. citeseer.nj.nec.com/485133.html.

[18] Urgaonkar, B., Shenoy, P. 2004. *Cataclysm: Handling Extreme Overloads in Internet Services*. In Proc. of PODC 2004.

[19] Urgaonkar, B., Shenoy, P. 2004. *Sharc: Managing CPU and Network Bandwidth in Shared Clusters*. IEEE Trans. on Parallel Distrib. Syst. 15,1, 2-17.

[20] Verma, A., Ghosal, S. 2003. *On Admission Control for Profit Maximization of Networked Service Providers*. In Proc. of WWW 2003 Conference, 128-137.

[21] Walsh, W., Tesauro, G., Kephart, J., Das, R. 2004. *Utility Functions in Autonomic Systems*. In Proc. of ICAC 2004.

[22] Villela, D., Pradhan, P., Rubenstein, D. 2004. *Provisioning servers in the application tier for e-commerce systems*. In IWQOS Proc., 57-66.

[23] Wolf, J., Yu, P. S. 2001. *On balancing the load in a clustered Web farm*. ACM Trans. on Internet Technology, 1,2, 231-261.

[24] Yaron, O., Sidi, M. 1993.*Performance and Stability of Communication Networks via Robust Exponential Bounds*. IEEE Trans. on Networking, 1,3, 372-385.

[25] Zhang, A., Santos, A., Beyer, D., Tang, H.K. 2002. *Optimal Server Resource Allocation Using an Open Queueing Network Model of Response Time*. HP Technical Report HPL-2002-301.

[26] Zhang, L., Ardagna, D. 2004 *SLA Based Profit Optimization in Autonomic Computing Systems*. In Proc. of ICSOC 2004.

[27] Zhang, L., Liu, Z., Riabov, A., Schulman, M., Xia, C. H., Zhang, F. 2003 *A Comprehensive Toolset for Workload Characterization, Performance Modeling, and Online Control*. Computer Performance Evaluation / TOOLS, 63-77, 2003.

[28] Zhang, Z. L., Towsley, D., Kurose, J. 1995. *Statistical analysis of the generalized processor sharing scheduling discipline*. IEEE Journal on Selected Areas in Comm., 13,6, 1071-1080.

[29] Zhou, X., Cai, Y., Godavari, G. K., Chow, C. E. 2004 *An Adaptive Process Allocation Strategy for Proportiobal Responsiveness Differentiation on Web Servers*. In Proc. of ICWS 2004.