SLA Based Profit Optimization in Autonomic Computing Systems

Li Zhang
IBM
T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
zhangli@us.ibm.com

Danilo Ardagna
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, 20133 Milano, Italy
ardagna@elet.polimi.it

ABSTRACT

With the development of the Service Oriented Architecture (SOA), organizations are able to compose complex applications from distributed services supported by third party providers. Under this scenario, large data centers provide services to many customers by sharing available IT resources. This leads to the efficient use of resources and the reduction of operating costs. Service providers and their customers often negotiate utility based Service Level Agreements (SLAs) to determine costs and penalties based on the achieved performance levels. Data centers often employ an autonomic computing infrastructure and use a centralized dispatch and control component (a dispatcher) to distribute the user requests to backend servers, and to set the scheduling policies at each server. This dispatcher can also decide to turn ON or OFF servers depending on the system load. This paper designs a set of dispatching and control policies for the dispatcher in such service oriented environments. The objective is to maximize the provider's profits associated with multiple class of SLAs. We show that the overall problem is NP-hard, and develop meta-heuristic solutions based on the tabu-search algorithm. Experimental results are presented to show the benefits of our approach.

Categories and Subject Descriptors:
[Performance and Reliability]: Quality of Service

General Terms: Performance, Algorithm, Management.

Keywords: Service delivery, monitoring, quality, and management, e-Business, Service reliability and availability Quality of service models.

1. INTRODUCTION

With the development of the Service Oriented Architecture (SOA), organizations often compose complex applications from distributed services supported by third party providers. Under this scenario, large data centers provide services to many customers by sharing available IT resources. This leads to the efficient use of resources and the reduction of the operating cost. The service providers and their customers often negotiate utility based Service Level Agreements (SLAs) to determine costs and penalties based on the achieved performance levels. The service provider need to manage its resource to maximize its profits. Utility based optimization approaches are commonly used to provide load balancing and to obtain the optimal trade-off among job classes for Quality of Service levels. Utility functions are also used as guidelines and for realizing high level trends. One main issues of these systems is the high variability of the workload. The ratio of the peak load to light load for Internet applications is usually on the order of 300% [7]. Due to such large variations in loads, it is difficult to estimate workload requirements in advance, and worst-case capacity planning is either infeasible or extremely inefficient. In order to handle workload variations, many data centers have started to implement *autonomic computing infrastructures* and employ self-managing techniques for resource allocations [12, 6, 4]. In such systems, resources are dynamically allocated among applications considering short-term demand estimates. The goal is to meet the application requirements while adapting the IT architecture to workload variations.

Figure 1 shows the hardware architecture of a data center implementing an autonomic infrastructure. Applications are allocated and de-allocated on demand on heterogeneous server clusters by the dispatcher. Each server can run under different operating systems and instantiate application processes on demand. Operating system, applications and data are accessed by storage networking technologies (SAN/NAS systems).

The main components of the dispatcher [15] include a monitor, a predictor and a resource allocator. The system monitor measures the workload and performance metrics of each application, identifies requests classes and estimates requests service time. The predictor forecasts system load conditions from load history. The allocator determines the best system configuration as well as the assignment of applications to servers.

This paper focuses on the design of a resource allocator for such service oriented environments. The scheduling policy is designed to maximize the revenue while balancing the cost (or energy) of using the resources. The overall profit (utility) includes the revenues and penalties incurred when Qual-

ICSOC'04, November 15–19, 2004, New York, New York, USA.

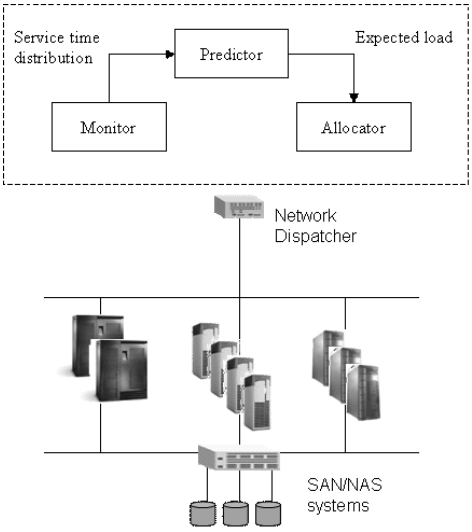

Figure 1: Autonomic Infrastructure Architecture

ity of Service guarantees are satisfied or violated. The revenue depends on the QoS levels in a discrete fashion. The revenue gained per request increases with the achieved performance level. This revenue function is realistic and is currently adopted in many Web hosting contracts. The allocator can establish the request volumes and the scheduling policy at each server. The allocator can also decide to turn ON or OFF servers depending on the system load. We show that the overall problem is NP-hard. We further develop meta-heuristic solutions based on the tabu-search algorithm. The neighborhood exploration is based on a fixed-point iteration, which requires solving a new network allocation flow 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. Section 5 analyzes the structural properties of the optimization problem. The structural properties are the basis for the search algorithms in Section 6. Experimental results in Section 7 demonstrates the quality and efficiency of our solutions.

2. RELATED WORK

Recently, the problem of maximization of SLA revenues in shared data center environments has attracted vast attention by the research community. Overviews of self-managing infrastructures can be found in [3, 7, 14, 6]. The problem of maximizing SLAs can be formulated as the dual problem of minimizing system response times and maximizing throughput as in [16]. That work considers the problem of hosting multiple web sites at the data center and proposes a static algorithm to assign Web sites to a set of servers. This algorithm is executed once a week, based on long term predictions. A dynamic algorithm implements a real time dispatcher and assigns incoming requests to servers considering short term load forecasts. In [13] continuous yield functions are introduced and the problem of maximization of yield is formulated as a scheduling problem. The work proposes a greedy scheduling algorithm while incoming requests are assigned to servers according to the queue length. The effectiveness of the overall approach is verified by simulation. The authors in [6] 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 is identified by the Lagrange multipliers. In [11], the authors proposed an analytical formulation of the problem to maximize the multi-class SLAs in heterogeneous web clusters considering the tail distribution of the requests response times. This problem is solved by a fixed point iteration which converges to the global optimum of the system (the cost function is concave continuous and differentiable), but the number of servers in every cluster is fixed independent of the load condition. The control variables are the GPS parameters at each cluster and the frequency of requests assigned to different clusters. The load is balanced in each cluster. The problem of minimizing the costs associated with servers is considered in [7], where the main costs associated with the use of resources is energy consumption. The authors proposed a greedy resource allocation algorithm which reconfigures cluster farms on the basis of servers utilization. Finally, [5] presents a study which estimates the benefits of resource multiplexing of on-demand data center environments with respect to the spatial allocation granularity and the temporal allocation granularity. The spatial allocation granularity refers to the granularity of the control, or the resource units allocated to customer classes. The temporal allocation granularity refers to the inter-scheduling time of the controller. The study proposes an analysis to evaluate the temporal granularity with respect to the accuracy of the predictor.

3. THE SYSTEM MODEL

We consider the data center as a distributed computer system consisting of M heterogeneous clusters. Each cluster is built from a number of homogeneous machines. There are totally K classes of request streams. Each class of requests can be served by a collection of servers. The dispatcher assigns the incoming requests to individual servers in the cluster. The dispatcher can also determine the scheduling policy at each server. Each server has a GPS scheduler. The allocation of weights for each class can be set by the dispatcher. The controller can also turn OFF and ON individual servers inside the clusters in order to reduce the overall cost. In our model, the SLA is defined by considering only service response times. For each job class, a step-wise utility function is defined to specify the per request revenue (or penalty) incurred when the corresponding average response times assumes a given value. Figure 2 shows, as an example, the plot of a utility function. Intuitively, customers are willing to pay a higher rate per request when their requests are served with lower response times. We observe the discontinuity in the function in Figure 2. As we will discuss in the next sections, this discontinuity in the cost function and the discrete nature of the problem make the optimization problem NP-hard. In the literature the load balancing problem with SLA profits was faced considering always continuous convex and differentiable

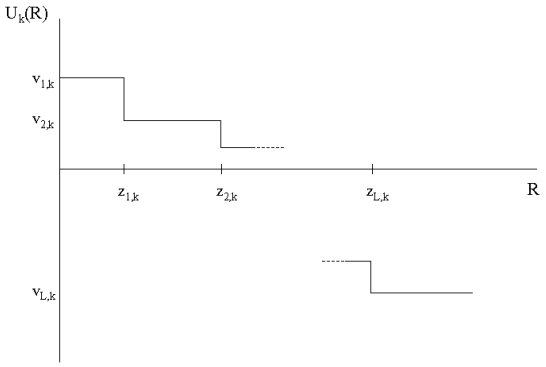

Figure 2: Utility Function

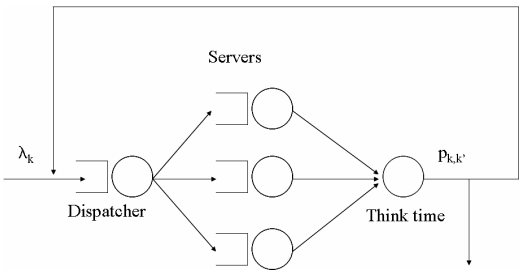

Figure 3: Network Queue Model

cost functions (see for example [7, 11, 16, 12]). Step-wise functions of the mean response times are more intuitive from the customer point of view, and are currently adopted [4]. A user request can be supported by multiple-tier applications. We apply our control schemes and solve the optimization for independent tiers. The data center is modeled by a queueing network composed of a set of multi-class single-server queues and a set of multi-class infinite-server queues. The former represents the collection of servers within heterogeneous clusters. The infinite-server queues represent the user-based delays, or think times, between the server completion of one request and the arrival of the subsequent requests within a session (see Figure 3). User sessions begins with a class k request arriving to the data center from an exogenous source with rate λ_k. 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 $\Lambda_k = \sum_{k'=1}^{K} \Lambda_{k'} p_{k',k} + \lambda_k$. A user request class corresponds to a single interaction of the end user within the execution of a service of the SOA model.

The following notation will be adopted in later sections:

M_i	:=	number of homogeneous servers within cluster i;
y_i	:=	number of cluster i servers ON;
C_i	:=	capacity of a single server in cluster i;
μ_k	:=	service rate for class k jobs at a server of capacity 1;
$\lambda_{i,k}$	:=	load at cluster i for class k jobs;
$\lambda_{i,m,k}$	:=	load at server m in cluster i for class k jobs;
$\phi_{i,m,k}$	:=	scheduling GPS parameter for class k jobs at server m within cluster i;
$R_{i,m,k}$	:=	response time for class k jobs at server m in cluster i;
$U_k(R)$	:=	utility step-wise function for class k jobs;
$v_{l,k}$	:=	revenue/penalty incurred when the response time for class k jobs lays in the l-th level;
L	:=	number of thresholds for utility functions;
c_i	:=	cost of a server in status ON in cluster i.

Note that c_i, the cost associated with turning on a server in cluster i, is a function of inter-scheduler time. In practice, $c_i \propto C_i$, if power is the main cost associated with turning on a server. The analysis of multi-class queueing system is notoriously difficult. We use the GPS bounding technique in [17] to approximate the queueing system. Under GPS the server capacity devoted to class k requests at time t (if any) is $C_i\phi_{i,m,k} / \sum_{k' \in \mathcal{K}(t)} \phi_{i,m,k'}$, where $\mathcal{K}(t)$ is the set of classes with requests waiting at server m in cluster i at time t. Requests within each class at every server are executed either in a First-Come First-Serve (FCFS) or a Processor Sharing (PS) manner. Under FCFS, we assume that the service requirements for class k requests at cluster i have an exponential distribution with mean $(C_i\mu_k)^{-1}$, whereas under PS service requirements of class k requests at cluster i follow a general distribution with mean $(C_i\mu_k)^{-1}$, including heavy-tail distributions for Internet applications. Since many servers exploit the local operating system mechanisms for scheduling work within a class, the assumption of PS within each class is reasonable for a wide range of servers found in practice [11]. In the approximation, each multi-class single-server queue associated with server m in cluster i is decomposed into multiple single-class single-server queues with capacity greater than or equal to $C_i\phi_{i,m,k}$. The response times evaluated in the isolated per-class queues are then upper bounds on the corresponding measures in the original system.

4. OPTIMIZATION PROBLEM

Given the system model in Section 3 we formulate the cost optimization problem below. We would like to maximize the overall profit by controlling the request routing and the processor sharing scheduling policies.

$$\max \quad \sum_{i=1}^{M} \left(\sum_{m=1}^{y_i} \sum_{k=1}^{K} U_k(R_{i,m,k}) \lambda_{i,m,k} - c_i y_i \right) \tag{1}$$

$$\sum_{m=1}^{y_i} \lambda_{i,m,k} = \lambda_{i,k} \tag{2}$$

$$\sum_{i=1}^{M} \lambda_{i,k} = \Lambda_k \tag{3}$$

$$\lambda_{i,k} \geq 0 \qquad \text{if } y_i > 0 \tag{4}$$

$$\sum_{k=1}^{K} \phi_{i,m,k} \leq 1 \quad (5)$$

$$R_{i,m,k} = \frac{1}{C_i\mu_k\phi_{i,m,k} - \lambda_{i,m,k}}; \quad y_i > 0; \quad \lambda_{i,m,k} < C_i\mu_k\phi_{i,m,k} \quad (6)$$

$$U_k(R_{i,m,k}) = \sum_{l=1}^{L} v_{l,k}\text{SAT}_{l,k}(R_{i,m,k}) \quad (7)$$

$$\text{SAT}_{l,k}(R_{i,m,k}) = 1 \Leftrightarrow z_{l-1,k} < R_{i,m,k} \leq z_{l,k}; \; z_0 = 0; \; z_{L+1} = \infty$$

$$\sum_{l=1}^{L} \text{SAT}_{l,k}(R_{i,m,k}) = 1 \quad (8)$$

$$y_i \in [0, M_i]; \quad y_i \text{ integral}$$

Equation (2) entails that the traffic assigned to individual servers in a cluster equals the overall load assigned to the cluster. Equation (3) defines the overall load of class-k jobs as a function of exogenous arrivals and feed-back probability. Note that in autonomic computing systems the exogenous arrival rate can be the prediction of the arrival rate for the current inter-scheduler period. Equations (4) assign requests to clusters according to the status of the cluster (servers ON or OFF). Finally, equations (5-7) express GPS scheduling bounds and utility functions in terms of response times (the condition $\lambda_{i,m,k} < C_i\mu_k\phi_{i,m,k}$ guarantees that resources are not saturated). Equation (8) guarantees that at each server, each job class is assigned to one SLA level, $SAT_{l,k}(R)$ is the indicator function for class k job which equals 1 if the response time R lays in the l-th level and is 0 otherwise. Here y_i, $\lambda_{i,m,k}$ and $\phi_{i,m,k}$ are decision variables and overall we have a Mixed Integer Programming problem.

In the following section, after fixing some variables, the problem will be reduced to a multi-choice binary knapsack (MKP) and an NP-hard network flow resource allocation problem. So the overall problem is NP-hard. The problem is solved by implementing a tabu-search algorithm; the evaluation of the neighborhood is based on a fixed point iteration of MKP and network flow resource allocation problems. The optimization technique will be discussed in Section 6.

5. ANALYSIS OF THE PROBLEM

Let us first fix the number of servers ON in a cluster. The problem of finding the best routing and scheduling parameters $\lambda_{i,m,k}$ and $\phi_{i,m,k}$, in order to maximize revenue at a single server, becomes a multiple-choice binary knapsack problem and a network flow resource allocation problem.

The Multi-choice Binary Knapsack Problem (MKP) is a variant of the classical Knapsack Problem. Let there be n groups of items. Group l has n_l items. Each item of the group has a particular value and it requires resources. The objective of the MKP is to pick exactly one item from each group to maximize the total value of the collected items, subject to the resource constraint of the knapsack. In mathematical notation, let $c_{l,k}$ be the value of the k-th item in l-th group, $w_{l,k}$ the resource requirement, W the resource bound of the knapsack and $x_{l,k} = 1$ if the k-th item in l-th group is picked in the solution, 0 otherwise. Then the problem is:

$$\max \sum_{l=1}^{n}\sum_{k=1}^{n_l} c_{l,k}x_{l,k}$$

$$\sum_{l=1}^{n}\sum_{k=1}^{n_l} w_{l,k}x_{l,k} \leq W$$

$$\sum_{k=1}^{n_l} x_{l,k} = 1 \qquad x_{l,k} \in \{0,1\}$$

If the number of servers ON and the load at each server are fixed, then in order to maximize the objective function, one can maximize revenues at single servers obtaining $\sum_{i=1}^{M} \overline{y_i}$ sub-problems:

$$\max \sum_{k=1}^{K}\sum_{l=1}^{L} v_{l,k}\overline{\lambda}_{i,m,k}\text{SAT}_{l,k}(R_{i,m,k}(\phi_{i,m,k}))$$

$$\text{SAT}_{l,k}(R_{i,m,k}(\phi_{i,m,k})) = 1 \Leftrightarrow$$

$$z_{l-1,k} < \frac{1}{C_i\mu_k\phi_{i,m,k} - \overline{\lambda}_{i,m,k}} \leq z_{l,k} \Leftrightarrow$$

$$\frac{1}{z_{l,k}} \leq C_i\mu_k\phi_{i,m,k} - \overline{\lambda}_{i,m,k} < \frac{1}{z_{l-1,k}}$$

$$\sum_{k=1}^{K} \phi_{i,m,k} \leq 1$$

$$\sum_{l=1}^{L} \text{SAT}_{l,k}(R_{i,m,k}(\phi_{i,m,k})) = 1$$

Here $\phi_{i,m,k}$ are the decision variables with $i = 1,..,M$, $m = 1,..,\overline{y}_i$ and $k = 1,..,K$. In order to satisfy the constrains and save resources for scheduling, the previous inequality can be satisfied by setting:

$$\text{SAT}_{l,k} = 1 \Leftrightarrow \phi_{i,m,k} = \frac{1}{z_{l,k}C_i\mu_k} + \frac{\overline{\lambda}_{i,m,k}}{C_i\mu_k}$$

This corresponds to selecting one interval of the utility function (see Figure 2) when the response time equals to the upper bound of the interval. We now have the MKP problem where parameters are defined by:

$$c_{l,k} = v_{l,k}\overline{\lambda}_{i,m,k}$$

$$w_{l,k} = \frac{1}{z_{l,k}C_i\mu_k} \quad (9)$$

$$W = 1 - \sum_{k=1}^{K} \frac{\overline{\lambda}_{i,m,k}}{C_i\mu_k} \quad (10)$$

$$x_{l,k} = \text{SAT}_{l,k}(R_{i,m,k}(\phi_{i,m,k}))$$

All groups have size L, the number of thresholds of utility functions. Note that equation (10) corresponds to the system equilibrium condition defined in equation (7), while the bound that keep selecting one item from every group in the MKP formulation corresponds to select one interval of the utility function for each SLA request job class.

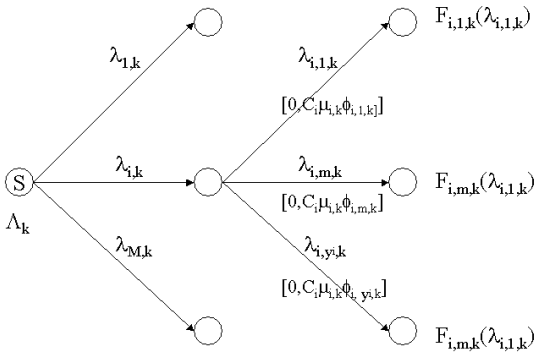

Figure 4: Network Flow Model

Now consider a directed network consisting of nodes $\mathcal{V}$ and direct arcs $\mathcal{A}$. The arcs $a_{v_1v_2} \in \mathcal{A}$ carry flow $f_{v_1v_2}$, from node $v_1 \in \mathcal{V}$ to node $v_2 \in \mathcal{V}$. The flow is a real variable that is constrained to be bounded below by a constant $l_{v_1v_2}$ and above by a constant $u_{v_1v_2}$. Suppose a single source node $s \in \mathcal{V}$, satisfies $\sum_{a_{sv_2}} f_{sv_2} - \sum_{a_{v_1s}} f_{v_1s} = \mathcal{R} > 0$. This value $\mathcal{R}$, the net outflow from the source, is a constant and represents the amount of resource available to be allocated. There are n sink nodes $v_2 \in \mathcal{N} \subseteq V$ which have the property that their net inflow $\sum_{a_{v_1v_2}} f_{v_1v_2} - \sum_{a_{v_2v_3}} f_{v_2v_3} > 0$. All other nodes v_2 are transshipment nodes that satisfy $\sum_{a_{v_1v_2}} f_{v_1v_2} - \sum_{a_{v_2v_3}} f_{v_2v_3} = 0$. A function $F_{v_2}(x)$ is associated with each sink node v_2 and the optimization problem is:

$$\max \sum_{v_2 \in \mathcal{N}} F_{v_2}\Big(\sum_{a_{v_1v_2}} f_{v_1v_2} - \sum_{a_{v_2v_3}} f_{v_2v_3}\Big)$$

$$\sum_{a_{sv_2}} f_{sv2} - \sum_{a_{v_1s}} f_{v1s} = \mathcal{R}$$

$$l_{v_1v_2} \le f_{v_1v_2} \le u_{v_1v_2} \quad \forall v_1, v_2 \in \mathcal{V}$$

If the number of server ON and the scheduling policy at each server are fixed, then in order to maximize the objective function one can establish the load at each server and solve the following K sub-problems:

$$\begin{aligned} \max \quad & \textstyle\sum_{l=1}^{L} v_{l,k}\lambda_{i,m,k}\mathrm{SAT}_{l,k}(R_{i,m,k}(\lambda_{i,k})) \\ & \textstyle\sum_{m=1}^{\overline{y}_i} \lambda_{i,m,k} = \lambda_{i,k} \\ & \textstyle\sum_{i=1}^{M} \lambda_{i,k} = \Lambda_k \\ & \lambda_{i,k} \ge 0 \quad \text{if } \overline{y_i} > 0 \\ & \mathrm{SAT}_{l,k}(R_{i,m,k}(\lambda_{i,k})) = 1 \Leftrightarrow \\ z_{l-1,k} < & \tfrac{1}{C_i\mu_k\overline{\phi}_{i,m,k} - \lambda_{i,m,k}} \le z_{l,k} \Leftrightarrow \\ C_i\mu_k\overline{\phi}_{i,m,k} - \tfrac{1}{z_{l-1,k}} < \lambda_{i,m,k} & \le C_i\mu_k\overline{\phi}_{i,m,k} - \tfrac{1}{z_{l,k}} \\ & \textstyle\sum_{l=1}^{L} \mathrm{SAT}_{l,k}(R_{i,m,k}(\lambda_{i,m,k})) = 1 \end{aligned} \tag{11}$$

Here $\lambda_{i,m,k}$ are the decision variables with $i = 1, .., M$, $m = 1, .., \overline{y}_i$ and $k = 1, .., K$. This problem is a special case of the network flow resource allocation problem where the overall flow is defined by equation (11), (i.e., the aggregate arrivals for class k jobs) and the network is shown in Figure 4. A plot

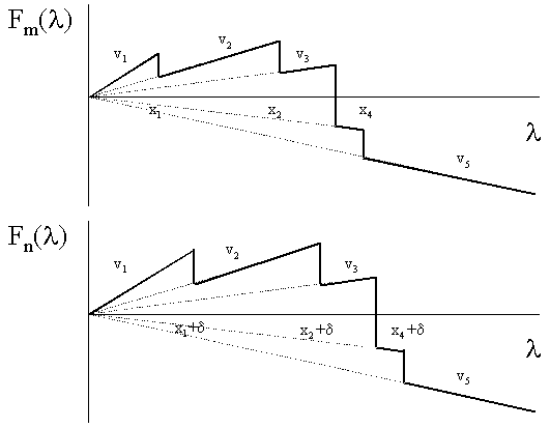

Figure 5: Sink Cost Function

of sink cost functions is shown in Figure 5. The cost function is discontinuous and the intervals where the function is linear are defined by:

$$(f_{l-1}^m, f_l^m] = \left(C_i\mu_k\overline{\phi}_{i,m,k} - \frac{1}{z_{l-1,k}}, C_i\mu_k\overline{\phi}_{i,m,k} - \frac{1}{z_{l,k}}\right].$$

Note that transhipments nodes represent server clusters, while sink nodes correspond to individual servers. The interval bounds are a function of the scheduling policy, the capacity of servers, and of utility function thresholds. Therefore, different sink nodes are characterized by different cost functions, with corresponding intervals characterized by the same slope. Note also that the slopes are decreasing from left to right and a sink function can obtain its maximum in any interval (the maximum of a sink function depends on the slopes and on the widths of intervals). If we consider different cost functions these are translated by $\delta = -C_i\mu_k\overline{\phi}_{i,m,k}$ (see Figure 4). Note also that if a server has limited capacity (i.e., $C_i\mu_k\overline{\phi}_{i,m,k} - 1/z_{l,k} < 0$) some intervals may be missing. The cost functions are neither convex nor differentiable for sink nodes. Therefore, the network flow resource allocation problem is NP-hard [10]. Nevertheless, the following properties of the optimal solution can be proved.

THEOREM 1. *In the optimum solution of a flow problem at most one server is assigned to a request rate different from a sink function upper edge interval.*

PROOF. If we relax equation (11) bound, the optimization problem has a trivial optimal solution. It assigns to each server the load which corresponds to the individual optimum of sink functions. (Note that each sink node corresponds to a server). Let's assume that in the optimal solution, two servers are assigned to a load λ', λ'' which do not coincide to upper edges of corresponding intervals. Let v' and v'' be the corresponding slopes and assume $v' > v''$. Then the solution can be improved by increasing the load λ' assigned to the first server while decreasing λ'' of the second one in order to satisfy the flow equation (11) until λ' coincide with the upper edge of the interval. This contradicts the hypothesis that the solution is optimal. □

Table 1: Utility Function Proportionality Coefficients

Thresholds	Costs
1.5	1000
5	250
10	150
20	100
50	50
100	30
1000	20
2000	10
5000	5
10000	-10^9

This property basically says that in the optimum solution there is only one λ *free assignment*.

THEOREM 2. *In the optimal solution of the flow problem, the λ free assignment is in the interval which corresponds to the lowest level of performance.*

PROOF. Let's consider an optimum solution, let λ' be the free assignment, and v', the slope of the corresponding interval. Assume that there is an assignment λ'' which coincide to an upper edge of an interval such that the corresponding slope v'' is greater that v'. Then the solution can be improved increasing λ' while decreasing λ'' in order to satisfy the flow equation (11). This contradicts the hypothesis that the solution is optimal. □

6. OPTIMIZATION TECHNIQUE

In the previous section we have shown that the optimization problem is NP-hard. We now provide structural properties of the problem. Based on these insights, we develop a tabu-search algorithm in order to find a quasi-optimal solution. The neighborhood exploration is based on a fixed point iteration procedure together with solving a network flow problem. The fixed point iteration procedure of the MKP determines the optimal scheduling policy, and the network flow problem determines the optimal routing policy. The next section will discuss the fixed point iteration procedure. Section 6.2 will describe the tabu-search algorithm implementation.

6.1 Fixed Point Iteration Procedure

The solution of the MKP problem is based on the HEU heuristic described in [2] which provides solution on average equal to 94% of the optimum. On the other hand, the literature does not provide any results for the network flow allocation problem and we have developed a local search approach in order to find a solution. The local search performs two dual moves in order to improve the initial solution:

- Increase the performance level of low performance servers by a unit and allocate their load to servers with higher performance level.
- Decrease the performance level of high performance servers by a unit and allocate their load to servers with lower performance level.

Based on the properties described in theorem 1 and 2, at each move, the load of the server with higher slope is increased and the load at the server with the lowest slope is decreased, in order to satisfy flow equation (11).

The local search starts increasing GPS parameters determined by the MKP solution. GPS parameters are modified in order to allow more degrees of freedom for performing the two moves in the second phase. We have implemented two approaches to increase GPS parameters which derive two different fixed point iteration procedures:

- At each server, GPS parameters are increased fairly among job classes assigned to the server, until their sum equals to 1,
- At each server only one job class GPS parameter is increased (again until sum locally equals to 1). The job class is chosen by separately solving a flow problem for each SLA class and selecting the class that gives the best improvement in the solution of the flow problem.

In order to evaluate the quality of the solution, results of the two approaches are compared with results of an exhaustive search algorithm. Here we only need to compare the performance of the algorithms for a fixed set of servers. Therefore, the cost associated with servers is neglected and only revenues are considered. Results are obtained by randomly generating service rates μ_k and class request rate Λ_k . The number of thresholds varied between 5 and 9, the utility functions have fixed thresholds, proportional to the service time $1/\mu_k$, and the cost scheme is also proportional to the service time [15]. The data center utilization varies between 0.2 and 0.8. Table 1 shows proportional coefficients for thresholds and costs. Note that independent of the number of steps adopted, the revenue associated with the last step equals -10^9, in this way we are guaranteed that the last thresholds is never violated.

Tests consider some small instances of the problem, three job classes and a data center with alternatively three and four servers. The exhaustive search is very time consuming, when the load is light (0.2 utilization of the overall capacity of the data center) with 5 servers the solution of a single problem requires a day. Overall 2600 tests were run and results are almost independent by the number of thresholds of utility functions. Results are shown in Table 2. We compare the solution with revenues of the proportional assignment scheme. The proportional assignment scheme employs:

$$\lambda_{i,m,k} = \Lambda_k \frac{C_i \mu_k}{\sum_{l=1}^{M} C_l \mu_k}$$

$$\phi_{i,m,k} = \frac{\lambda_{i,m,k}/\mu_k}{\sum_{l=1}^{K} \lambda_{i,m,l}/\mu_l}$$

Note that this proportional allocation scheme is a natural way to assign the traffic and 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 [11].

Table 2 shows the average error, the maximum error (with respect to the exhaustive search), the average improvement and maximum improvement with respect to the proportional assignment scheme. The table also reports the results of a third

Table 2: Fixed Point Iteration Results

	Av. err%	Max err%	Av. impr%	Max impr%
FPI1	15.73%	66.63%	131.75%	423.98%
FPI2	15.40%	69.16%	134.44%	413.37%
FPI 1+2	12.21%	66.17%	145.71%	423.98%

heuristic which takes the maximum between the solutions of fixed point iterations 1 and 2. This is the implementation that has been adopted in the tabu-search. In this way both the average error and the average improvement can be enhanced.

Exhaustive search results show that the optimal solution is very different from the proportional assignment scheme. It often favors the use of dedicated servers for job classes instead of balancing the load among servers of clusters. The main reason is that dedicated servers give better performance. Let's consider as an example two different servers with the same capacity and two job classes. If the two job classes has, for the sake of simplicity, the same arrival rate λ and the same service time $1/\mu$ the proportional assignment scheme has a response time $R_1 = R_2 = 1/(0.5(\mu - \lambda))$ twice than the dedicated server scheme $R_1 = R_2 = 1/(\mu - \lambda)$.

Plot in Figure 6 shows, as an example, the trace of the execution for the two fixed point iteration procedures. The plot shows that the fixed point iterations converges very quickly (usually less than 50 iterations). The execution time is about 10 seconds.

6.2 Tabu Search Algorithm

The overall optimization problem (1) is solved by implementing a tabu-search algorithm [9]. The tabu search is an adaptive procedure with the ability to make use of many other methods and specialized heuristics, which it directs to overcome the limitations of local optimality. Tabu search and meta-heuristic approaches can be seen as a generalization of the local search strategy. Let us consider a combinatorial optimization problem in the following form:

$$\min c(x) : x \in X$$

where the cost function $c(x)$ could be linear or nonlinear and the condition $x \in X$ is assumed to constrain specified components of x to discrete values. The basic idea of local search is to find a solution through a sequence of moves that lead from one solution to another. Formally, a move s is a mapping defined on a subset $X(s)$ of X, $s : X(s) \rightarrow X$. Associated with a feasible solution x is the set $S(x)$ which consists of those moves $s \in S$ that can be applied to x, $S(x) = \{s \in S : x \in X(s)\}$. A neighborhood of an admissible solution x, referred to as $N(x)$, is the set of solutions that can be reached from x by applying all possible moves $S(x)$, $N(x) = \{x' \in X : x' = s(x), s \in S(x)\}$. The local-search algorithm starts from an initial feasible solution $x \in X$ and explores the neighborhood $N(x)$ searching for a solution x' that improves the cost function $c(x)$. If x' is found in $N(x)$, the local search will explore $N(x')$. On the contrary, if $N(x)$ does not contain any solution improving the cost function, the local-search algorithm terminates returning x as a local optimum. The pitfall of this basic idea is that the cost function

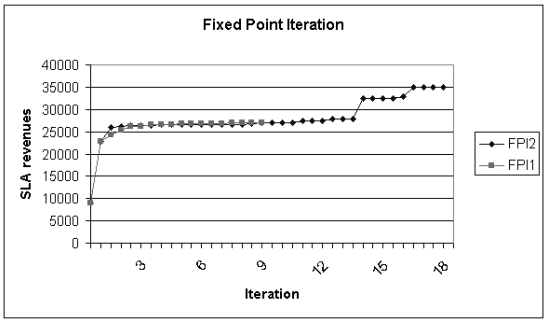

Figure 6: Fixed Point Iteration Execution Trace

$c(x)$ of most real problems has multiple local optima, which can be significantly different from the global optimum. Meta-heuristic approaches have been proposed as extensions of the local search algorithm and do not stop at the first local optimum. Tabu search guides local search to continue exploration without becoming confounded by an absence of improving moves and possibly without falling back into a local optimum from which it previously emerged. From its ability to incorporate and guide another procedure, tabu search may be viewed as a meta-strategy. Tabu search proceeds in the following way: if a solution x is found to be a local optimum with no $x' \in N(x)$ improving the cost function, a solution $x'' \in N(x)$ worsening the cost function can be selected. $N(x'')$ is then explored, excluding x and a predefined number of previously explored solutions in order not to cycle around the same local optimum. Cycling is avoided by introducing tabu moves: a subset T of S is created; elements of T are determined by historical information of the search process, moves in T are not applied to select the next solution for some iterations.

In our implementation, the neighborhood of the current solution is defined by two moves which alternatively increase and decrease the number of servers ON at each cluster. The evaluation of each move requires a fixed point iteration of the MKP and the network flow allocation problems in order to obtain $\lambda_{i,m,k}$, $\phi_{i,m,k}$ values. The fixed point iteration discussed in the previous section converges very quickly. But we can not guarantee it converges to a global optimal solution. For this reason in the evaluation of a move the fixed point iteration is executed twice. The first execution considers as initial solution the solution obtained by applying the proportional assignment scheme. The second execution, tries to take advantage of the current solution for routing and scheduling policies in the following way:

- If the move turns ON a server, then for the new server we apply the scheduling policy of the cluster bottleneck; the new server and the bottleneck server share equally the load. For the other servers in the cluster, the routing and scheduling policy remains the same as the previous iteration.

- If the move turns OFF a server, then its load is assigned to others servers in the same cluster proportionally according to servers' spare capacity, the scheduling policy is unmodified. That is the scheduling of the previous iteration is applied.

The execution that take advantage of the current scheduling and routing assignment is more efficient and leads to the next current solution in almost 70% of cases. The neighborhood of a solution is defined by all solutions that can be obtained by applying these moves to all clusters. The search is guided by a tabu-search meta-heuristic in which only the short-term memory mechanism has been implemented. Since the neighborhood has almost the same size during the algorithm execution, the tabu list has a static size proportional to the number of clusters in the system. Note that the neighborhood exploration has complexity $O(MK\sum_{i=1}^{M} M_i)$. In fact, each fixed point iteration has complexity $O(K\sum_{i=1}^{M} M_i)$, since it solves K flow problems and a scheduling problem for each server ON (the overall number of server is $\sum_{i=1}^{M} M_i$) and the tabu-search neighborhood has size $O(M)$.

In order to obtain good results, we faced the problem of finding a high quality initial configuration for servers at the data center. The number of servers ON y_i are the main variables of the problem, since they affect performance and cost function. On the other hand, $\lambda_{i,m,k}$ and $\phi_{i,m,k}$ affect only performance and can be considered fine tuning variables, while at high level, the performance of a data center mainly depends on the number of servers adopted. In order to find a good initial solution for the tabu-search algorithm, we have developed two different methods to identify the initial configuration of servers ON.

The first method finds the initial solution by applying a greedy algorithm that assigns dedicated servers to job classes, exploiting the structure of the optimal solution discovered from exhaustive search results. The greedy assigns job classes to dedicated servers in a way that job classes have the same performance level at every server. Note that if job class k is assigned to the performance level $\bar{l}$ then the revenue associated with the class is simply $v_{\bar{l},k}\Lambda_k$. The procedure starts assuming that all requests can be satisfied according to the best quality level. In the following, the algorithm iteratively tries to assign job classes to dedicated servers and, if the capacity available at the data center is not sufficient, then the performance level of the class that corresponds to the minimum loss in revenue (that is the class k such that $(v_{\bar{l},k} - v_{\bar{l}+1,k})\Lambda_k$ is minimum) is decreased.

The second method finds the initial solution by assigning again dedicated servers to job classes but the assignment is identified by the solution of a multiple-choice multiple-dimension knapsack problem (MMKP). A multiple-dimension knapsack problem is one kind of knapsack where the resources are multi-dimensional, i.e. there are multiple resource constrains for the knapsack. The MMKP problem is a combination of the MKP problem presented in Section 5 and a multiple-dimension knapsack. Formally, let there be n groups of items, group l has n_l items, let $c_{l,k}$ be the value of the k-th item in l-th group, $w_{l,k,i}$ the amount of resource i required by the k item in the l group and W_i the amount of the i resource. Then the problem is:

$$\begin{aligned} \max \quad & \sum_{l=1}^{n}\sum_{k=1}^{n_l} c_{l,k}x_{l,k}; \sum_{l=1}^{n}\sum_{k=1}^{n_l} w_{l,k,i}x_{l,k} \quad \leq \quad W_i \\ & \sum_{k=1}^{n_l} x_{l,k} = 1; x_{l,k} \quad \in \quad \{0,1\} \end{aligned}$$

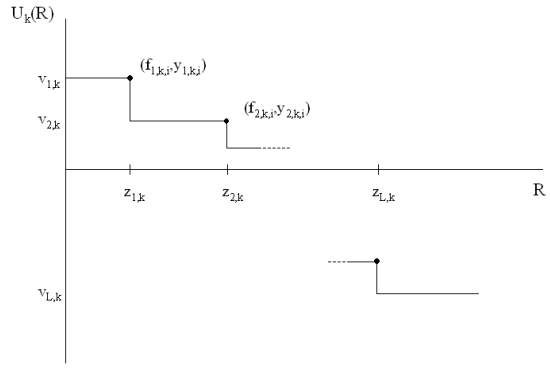

Figure 7: Utility Function, Multiple Class Dedicated Servers

In this case, we assign job classes to dedicated servers inside each cluster which share the load assigned to the cluster. Let us denote with $y_{i,k}$ the number of servers dedicated to job class k at cluster i. Class load is assigned to cluster proportionally to their capacity, i.e., $\lambda_{i,k} = \frac{C_i}{\sum_{i=1}^{M} C_i}\Lambda_k$. Now let us consider a very simple system consists of a single class and single cluster. Let C be the capacity of servers and c the cost associated with servers in status ON. If the load is balanced among servers in the clusters then the response time is given by $R = \frac{1}{C\mu - \lambda/y}; \lambda < C\mu y$ and the cost function becomes $U(R)\lambda - cy$.

If we consider the utility function in Figure 2, it is easy to see that the optimal values of y can be found by considering the discontinuity points of the utility function. This is because in the same interval, the response time is smaller and potentially the number of server ON is greater. But the overall revenue is the same. So, each discontinuity point could be characterized by a couple $y_l = \lceil \frac{\lambda}{C\mu - 1/z_l} \rceil$, f_l, where the latter is the value of the cost function obtained with $y = y_l$. The optimal number of server ON could be evaluated this way simply by inspection. Now consider the general system under study but assign job classes to dedicated servers. Then the response time is given by $R_{i,k} = \frac{1}{C_i\mu_k - \lambda_{i,k}/y_{i,k}}; \lambda_{i,k} < C_i\mu_k y_{i,k}$.

Considering each job class k and cluster i then the set $\{(f_{l,k,i}, y_{l,k,i})\}$ can be determined (see Figure 7) as discussed previously. Let be $x_{l,k,i} = 1$ if class k is assigned to cluster i and the corresponding SLA is l, and $x_{l,k,i} = 0$ otherwise. Then we can consider the problem:

$$\max \sum_{i=1}^{M}\sum_{k=1}^{K}\sum_{l=1}^{L} f_{l,k,i}x_{l,k,i}$$

$$\sum_{k=1}^{K}\sum_{l=1}^{L} y_{l,k,i}x_{l,k,i} \leq M_i \qquad (12)$$

$$\sum_{l=1}^{L} x_{l,k,i} = 1 \qquad x_{l,k,i} \in \{0,1\} \qquad (13)$$

where the set of constrains (12) implies that at most M_i servers in each cluster are assigned and constrains (13) assign each job class to exactly one SLA level in each cluster.

Also this MKKP problem has been solved implementing HEU heuristic proposed in [2]. Next section will show re-

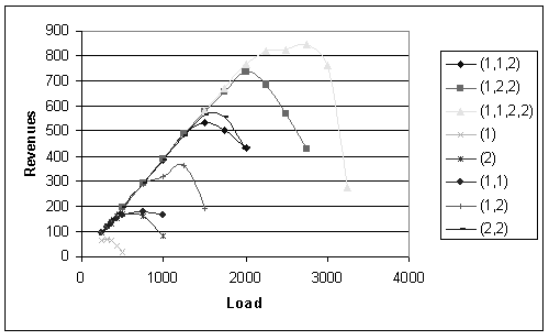

Figure 8: Revenue for Different Data Center Configurations

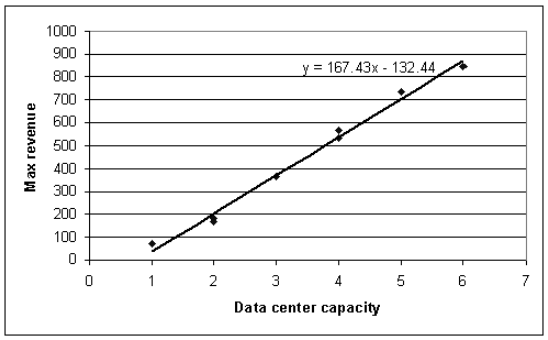

Figure 9: Maximum Revenue vs. Data Center Capacity Plot

sults that can be obtained by adopting the two different initial solutions.

7. EXPERIMENTAL RESULTS

In this section we present experimental results to illustrate the effectiveness of our approach. The number of clusters varies between 2 and 10. We consider the overall data centers with 200 servers and 200 job classes (and utility functions). Service times were random generated and for each test case the load was increased in a way that the utilization of data center resources varied between 0.2 and 0.8.

We begin with a simple exercise to decide the cost associated with servers. The cost of a unit capacity server has been evaluated considering the revenues obtained. Plots in Figure 8 shows revenues obtained by applying an exhaustive search algorithm on data centers characterized by various capacities and configurations for increasing load (numbers in parenthesis specify the capacity of servers adopted). It is interesting to note that revenues increase almost linearly but start decreasing after a maximum that is obtained when the data center utilization is about 0.5-0.6. After the maximum, job classes are assigned to lower levels of performance and the increasing load implies a loss in revenues instead of a potential benefit for the Service Provider. Figure 9 shows that the maximum revenue grows linearly with data center capacity with coefficient almost equal to 160. In our tests we used 120 as unit capacity cost; sensitivity analysis showed that a $\pm 20\%$ variation of unit cost coefficient implies on average a $\pm 15\%$ variation of our results.

Figure 10 reports, as an example, the trace of execution of the tabu-search algorithm, while Table 3 reports the average execution time for different problem instance sizes. Plots show that the tabu-search approach is efficient since the current optima can be improved after the analysis of worsening solutions. Usually the initial solution obtained with the HEU heuristic gives better performance in terms of both the initial and the final solution when the load is high. When the load is light, better results can be obtained by the greedy approach. The second best solution identified by the algorithm usually differs from the best one by at most one server. Sometimes, the second best solution uses the same number of servers as the identified optimal but adopts different scheduling and routing policies.

An estimate of the quality of our solution is obtained by comparing our results with results of an exhaustive search algorithm. Tests considered data centers with two clusters shared by 3 job classes for increasing loads. In order to keep the analysis tractable a cluster with y_i servers ON was modeled as a single server with capacity $C_i y_i$. Results are quite good. The reason is, with this approximation, the exhaustive search performance of a single cluster are y_i times better than that obtained with y_i servers. The average error was about 30% varying between 5-70%. The error increases with the utilization of the data center since the number of servers adopted in the solution also increases and the inaccuracy of the estimation grows.

In order to compare our results with the adoption of the proportional assignment scheme, the number of servers to turn ON is evaluated as the number of servers that keeps the utilization of the data center equals to 0.6. This is according to the greedy solutions which adopt utilization thresholds in resource allocation control as described in [7] and [1] (We empirically verified that applying proportional assignment, SLA are optimized when resource utilization is approximately 0.6). Considering this scenario, our approach improves SLA revenues of one order of magnitude since for the same load our controller is able to reduce the number of servers ON. We further inspect the solutions and identified islands of servers which share the load inside clusters. But in general, the load is not equally balanced among all of the servers of a cluster.

In most of the cases, our resource allocator adopts all servers available at the data center when the load reaches about 50% of its capacity. When the load is light, turning some server OFF provides better results. In order to evaluate the effectiveness of turning servers OFF, Table 4 compares results that can be achieved by our resource allocator with results that can be obtained by turning all servers ON and adopting our optimal routing and scheduling policies (that is by applying the fixed point iteration). Overall 200 tests were considered, as the table shows turning servers OFF allows to improve the cost function by about 25%, exploiting the trade off between higher revenues (which can be obtained by turning all servers ON) and the costs associated with servers.

8. CONCLUSIONS

We proposed an resource allocation controller for autonomic computing data center environments which maximizes the provider profits associated with multi-class Service Level Agree-

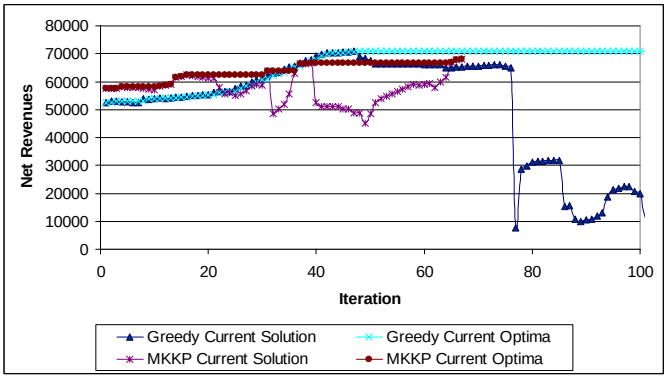

Figure 10: Execution Trace of the Tabu-search Algorithm for Different Initial Solutions

Table 3: Average execution time (in minutes) of the tabu search algorithm for different problem sizes

M	Total numb. of servers	K	Av. execution time
4	80	80	2.5
6	120	120	7.1
8	160	160	13.3
10	200	200	30.6

Table 4: Average improvement of the cost function obtained by turning servers OFF as a function of data center utilization

Data center utilization	Av. impr%
0.2	30.5%
0.3	27.86%
0.4	17.22%

ments. 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 servers. The overall optimization problem is NP-hard and we developed a meta-heuristic solution based on the tabu-search algorithm. Experimental results show that revenues that can be obtained with a proportional assignment scheme can be significantly improved. We verified the quality of our solution with exhaustive search. Future work will extend the model in order to include the tail distribution of response times in the optimization problem.

9. 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 and Distributed Systems. 13, 1, 80-96.

[2] Akbar, M. M., Manning, E.,G., Shoja, G., C., Khan, S. 2001. *Heuristic solution for the Multiple-Choice Multiple-Dimension Knapsack problem*. Conference on Computational Science, San Francisco, USA.

[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 Symposium on Integrated Network Management, 855-868.

[4] Boutilier, C., Das, R., Kephart, G. Tesauro, G. Walsh W. 2003. *Cooperative Negotiation in Autonomic Systems using Incremental Utility Elicitation*. To appear, Uncertainty in Artificial Intelligence.

[5] 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.

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

[7] Chase , J. S., Anderson, D. C. 2001 *Managing energy and server resources in hosting centers*. In Proc. of the eighteenth ACM symposium on Operating systems principles, 103-116.

[8] Chen., X., Mohapathra, P, Chen, H. 2001. *An Admission Control Scheme for Predictable Server Response Time for Web Access*. In Proc. of WWW 2001, 545-554.

[9] Glover, F., W., Laguna, M. 1997. Tabu Search. *Kluwer Academic Publishers*.

[10] Kim, D., Pardalos, P.M. 1999. *Dynamic Slope Scaling and Trust Interval Techinques for Solving Concave Piecewise Linear Network Flow Problems*. Networks 35, 3, 216-222.

[11] Liu, Z., Squillante, M. S., Wolf, J. 2001 *On maximizing service-level-agreement profits*. In Proc. of the 3rd ACM conference on Electronic Commerce, 213-223.

[12] Liu, Z., Squillante, M. S., Wolf, J. 2002. *Optimal Resource Management in e-Business Environments with Strict Quality-of-Service Performance Guarantees*. IEEE Conference on Decision and Control.

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

[14] Urgaonkar, B., Shenoy, P., Roscoe, T. 2002. *Resource Overbooking and Application Profiling in Shared Hosting Platforms*. ACM SIGOPS Operating Systems Review, 36, 239-254.

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

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

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