

# Business-Oriented Autonomic Load Balancing for Multitiered Web Sites

John M. Ewing and Daniel A. Menascé

Dept. of Computer Science, MS 4A5

The Volgenau School of Information Technology and Engineering

George Mason University

4400 University Dr.

Fairfax, VA 22030

jewing2@gmu.edu, menasce@cs.gmu.edu

**Abstract**—Autonomic computing systems are able to adapt to changing environments (such as changes in the workload intensity or component failures) in a way that preserves high-level operational goals, such as service level objectives. This paper focuses on autonomic computing systems that are self-optimizing and self-configuring. More specifically, the paper presents the detailed design of an autonomic load balancer (LB) for multitiered Web sites. It is assumed that customers can be categorized into distinct classes (gold, silver, and bronze) according to their business value to the site. While the example used in the paper is that of an auction site, the approach can be easily applied to any other Web site. The autonomic LB is able to dynamically change its request redirection policy as well as its resource allocation policy, which determines the allocation of servers to server clusters, in a way that maximizes a business-oriented utility function. The autonomic LB was evaluated through very detailed and comprehensive simulation experiments and was compared against a round-robin LB and against a situation where each customer category has a dedicated number of servers. The results showed that the autonomic LB outperforms the other load balancing approaches in terms of providing a higher utility for highly dynamic workloads.

## I. INTRODUCTION

In complex environments where the workload varies widely and is hard to predict, there is a need to design and build systems that can regulate themselves without human intervention. Such systems, called autonomic computing systems (also known as self-\* systems), are able to adapt to changing environments (such as changes in the workload intensity or the failure of a component) in a way that preserves given operational goals (e.g., service level objectives). There has been significant research and attention to autonomic computing in recent years [1], [2], [3], [4]. Previous work presented a technique to design self-optimizing and self-tuning computer systems based on the combined use of heuristic search techniques and analytic queuing network models [1], [4]. Other approaches used to design self-configuring systems include control theory [5], machine-learning [6], and fuzzy logic [7].

This paper shows how autonomic techniques for self-optimization and self-configuration can be employed to optimize the utility of a multi-tiered Web site through an autonomic two-level policy adaptation of the application layer load balancer.

Load balancing refers to a number of widely used techniques for distributing work among multiple resources according to a given policy. In recent work, autonomic principles have been applied to the development of dynamic load balancing policies that allow system adaptation in the face of an uncertain and changing environment [8], [9], [10]. Some dynamic load balancing policies seek to improve system efficiency by dispatching a work request to a specific resource where the effort required to process the request is minimized or where service level objectives are most likely to be met [8], [9], [10], [11], [12]. Other dynamic load balancing policies seek to prioritize work requests that generate more utility [13]. Our paper uses some of the dynamic load balancing policies first described in [14] that prioritize the requests most likely to generate utility. Our autonomic controller extends our work in [1], allows for greater precision in the development of these load balancing policies, provides the capability to reallocate cluster resources, and is well-suited for highly dynamic workloads.

The main contribution of this paper is a *business-oriented* approach to dispatching incoming requests to servers and allocating servers to server clusters according to customer priority classes. We differ from our previous work and that of others in that we 1) present a business-oriented utility maximization and workload generation, 2) provide a two-level autonomic policy adaptation, 3) develop an efficient hill-climbing heuristic that can quickly search a 6-dimensional policy space and, 4) demonstrate that our approach adapts and reacts well to highly dynamic loads. The approach considered in this paper is aimed at improving the revenue of an e-commerce site, an auction site in our example, by providing better performance to groups of customers that have higher business value at the expense of other less important customers. More specifically, the contributions of this paper are:

- the design of an autonomic controller that can search for load dispatching and resource allocation policies that maximize business-value, as specified by a utility function,
- an evaluation and comparison of the autonomic controller against commonly employed static load balancing

policies through rigorous experimentation that simulates a large e-commerce website servicing workloads drawn from analysis of real e-commerce systems [15] on a scale that is not practical and very expensive to reproduce in a laboratory setting, and

- a procedure to generate dynamic workloads that mimic extreme phenomena such as flash crowds.

The rest of this paper is organized as follows. Section two describes the multi-tiered web site, its workload, the utility function maximized by the autonomic controller, and its policies. Section three describes the heuristic based controller algorithm in detail. The next section discusses the experimental results used to evaluate the controller as well as a comparison with other load balancing approaches. Finally, section V presents some concluding remarks.

## II. BACKGROUND

### A. The Environment

We consider a multi-tiered environment (see Fig. 1) that consists of a site load balancer that receives requests from the Internet and sends them to one of  $n_{ws}$  web servers. Many requests may require the execution of an application by one of the  $N_A$  application servers, which are divided into three clusters: *gold*, *silver*, and *bronze*. Each server cluster services requests from its corresponding customer category as well as requests from customers from other categories according to the re-direction policy to be explained later.

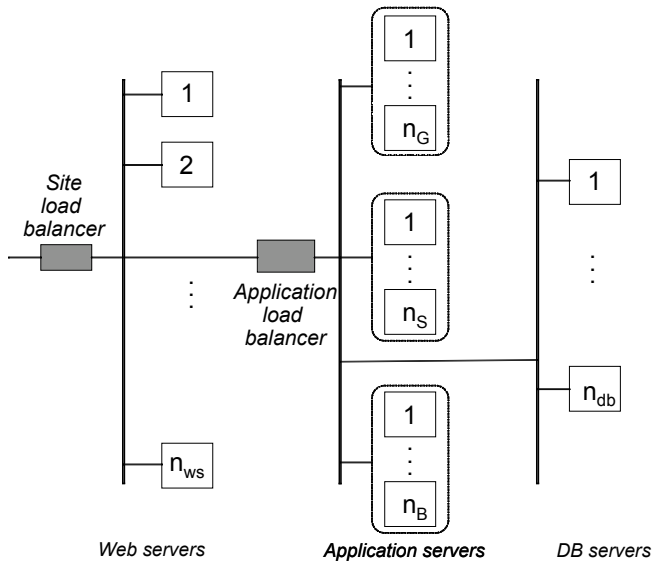


Fig. 1. Multi-tiered Environment

An application load balancer (LB) makes dispatching decisions regarding which cluster should receive an incoming request. The number of servers in each application server cluster is denoted by  $n_G$ ,  $n_S$ , and  $n_B$ , for gold, silver, and bronze, respectively. These numbers vary over time according to the autonomic behavior of the LB. However,  $n_G + n_S + n_B = N_A$ .

Backend database servers may be used by the application servers to process their requests. This paper concentrates

on the autonomic policies used by the LB to manage the application server tier. However, the techniques described and evaluated here can also be used by the site load balancer to make more efficient use of the web servers.

The workload generated by gold, silver, and bronze customers is described using a Customer Behavior Model Graph (CBMG) (see [16]), for each category of customers. Each node of a CBMG represents a state in which a customer may be found during a session. Nodes of the CBMG are connected by directed arcs that indicate the possible transitions between states. Arcs are labeled with the probability that a transition between states occurs.

Figure 2 shows a simplified version of the CBMG. The figure only shows the most important states; in each CBMG there is an additional state, the Exit state, not shown in the picture to make it easier to read. All the states whose sum of the probabilities of the outgoing transitions does not add to one have a transition to the Exit state with a probability equal to 1 - the sum of the other transition probabilities. Our experiments use a more elaborate version of the CBMGs, one per customer category, with different transition probabilities for each category.

The average number of visits to each state during a session can be computed by solving a system of linear equations as described in [16]. Table I shows the expected percentage of all visits for each customer class to selected states within the CBMG. As the table illustrates, gold customers submit almost 25% ( $= 9.96/8.01$ ) more bids on average than silver customers and silver customers submit 140% ( $= 8.01/3.31$ ) more bids on average than bronze customers. The gold and bronze CBMGs were developed from measurements at a real-world e-commerce site [15]. The silver CBMG was developed for this paper to represent a power user who navigates the site primarily through searches.

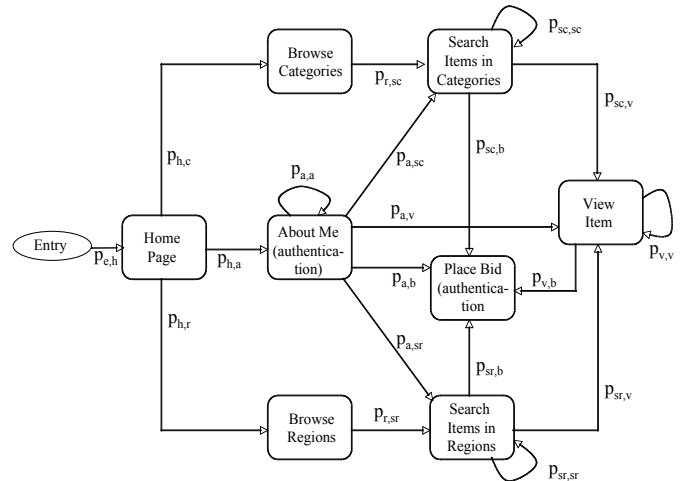


Fig. 2. Customer Behavior Model Graph (CBMG).

### B. Utility Function

The autonomic LB optimizes a global utility function that calculates the business-value generated by the throughput of

TABLE I  
PERCENTAGE OF SESSION VISITS TO SELECTED CBMG STATES FOR EACH CUSTOMER CATEGORY.

State	Customer Category		
	Gold	Silver	Bronze
Home	1.03	2.84	8.76
Browse Categories	1.24	0.37	8.90
Browse Regions	0.52	0.15	3.02
About Me	17.28	11.84	2.98
Search Items in Categories	1.44	18.13	15.34
Search Items in Regions	0.58	13.98	5.21
View Item	4.70	5.35	11.10
Store Bid	9.96	8.01	3.31

specific revenue-generating transactions with a certain expected response time. We use the *bid* transaction throughput because it generates revenue for the bidding site. Good response times are also critical to generating value—when response times are good, current customers continue to use the auction site and new customers are attracted to the site by favorable impressions. If response times are poor, customers are likely to abandon the site and use a competing auction site with better response times—this effect deprives the site of future business. This work uses the sigmoid utility function  $U_s^R(R_s)$  from [1] as the response time factor in the global utility function. This response time utility function, shown in Eq. (1), models whether utility is generated by complying with the response time service level objective (SLO):

$$U_s^R(R_s) = \frac{e^{-R_s + \beta_s}}{1 + e^{-R_s + \beta_s}} \quad (1)$$

where  $s$  is the priority class (i.e., gold, silver, or bronze),  $\beta_s$  is the average response time SLO, in sec, for class  $s$ , and  $R_s$  is the average response time, in seconds, of all class  $s$  transactions. The response time utility function has a value between zero and 1 and goes to zero as the response time goes to infinity. The value of the utility is 0.5 when the response time meets the SLO (i.e.,  $R_s = \beta_s$ ). We would like our load balancing policies to maximize the values of  $U_s^R$  in a way that prioritizes those customer classes that are most likely to generate bids. This is achieved by combining the response time utility factors in a weighted sum:

$$U_{total}^R(\vec{R}) = \sum_{\forall s} w_s \times U_s^R(R_s) \quad (2)$$

where  $w_s$  are weights defined by management to indicate the priority of class  $s$ , and  $\vec{R}$  stands for the vector of average response times for the classes.

The *bid* throughput,  $X_{bid,s}$ , for priority class  $s$  is the second factor in the utility. The total bid throughput,  $X_{bid}$ , is computed as follows:

$$X_{bid}(\vec{X}) = \sum_{\forall s} X_{bid,s} \quad (3)$$

where  $\vec{X}$  stands for the vector of average bid throughputs for the classes. From a business perspective,  $X_{bid}(\vec{X})$  represents how much money is being made today, while  $U_{total}^R(\vec{R})$  represents the likelihood of customers returning tomorrow. Our

goal is to maximize both  $X_{bid}(\vec{X})$  and  $U_{total}^R(\vec{R})$  in a way that maximizes revenue today while ensuring the most important customers are satisfied and will continue using the site in the future. Making the global utility,  $U_g$ , the product of  $X_{bid}(\vec{X})$  and  $U_{total}^R(\vec{R})$  achieves this goal:

$$U_g(\vec{R}, \vec{X}) = X_{bid}(\vec{X}) \times U_{total}^R(\vec{R}). \quad (4)$$

It should be noted that the values of  $\vec{R}$  and  $\vec{X}$  depend on the specific policy vector  $\vec{s}$  used by the LB (see next section) and on the workload intensity  $\mathcal{W}$ . Thus, the utility function can be written as a function  $h$  of  $\vec{R}$ ,  $\vec{X}$ ,  $\vec{s}$ , and  $\mathcal{W}$  as

$$U_g(\vec{R}, \vec{X}) = h(\vec{R}, \vec{X}, \vec{s}, \mathcal{W}). \quad (5)$$

### C. LB Policies

The LB uses two autonomic policies. The first, called *f-policy*, is a re-direction policy that affects the dispatching of requests to server clusters. This policy is specified by three parameters of the form  $f_{i,j} \in [-1, 1]$  that indicate the fraction of requests from priority class  $i$  to cluster  $j$ . In our case, these parameters are  $f_{S,G}$ ,  $f_{B,S}$ , and  $f_{B,G}$ . A positive value for  $f_{i,j}$  indicates redirection of class  $i$  requests to cluster  $j$  and a negative value indicates a redirection in the opposite direction.

An *f-policy* is then characterized by the vector  $\vec{f} = (f_{S,G}, f_{B,S}, f_{B,G})$ . The autonomic LB dynamically adjusts the *f-policy*  $\vec{f}$  to maximize the global utility  $U_g$ .

In order to preserve the identity of the clusters, we add one more constraint to the values within  $\vec{f}$ :  $f_{S,G}$  and  $f_{B,G}$  must carry the same sign (i.e.,  $f_{S,G} \times f_{B,G} \geq 0$ ). Without this restriction, the autonomic controller will occasionally swap the clusters around (e.g., move gold to silver, silver to bronze, and bronze to gold). At first glance, this behavior might seem acceptable, however, moving clusters around could result in a number of high priority requests being stuck behind a large number of low priority requests from the previous occupant of the cluster. This ultimately yields unacceptable response times for high priority requests.

The second policy, called *s-policy*, is a resource allocation policy that determines how many servers should be allocated to each cluster. In other words, it determines the values of  $n_G$ ,  $n_S$ , and  $n_B$ , which combined with the values of  $f_{S,G}$ ,  $f_{B,S}$ , and  $f_{B,G}$  maximize the global utility function. The *s-policy* is characterized by the vector  $\vec{n}_A = (n_G, n_S, n_B)$ .

The state space  $\mathcal{S}$  of all possible configurations is formally described below with the help of the  $\epsilon(x)$  function defined as 0 for  $x \geq 0$  and 1 for  $x < 0$ .

$$\begin{aligned} \mathcal{S} = \{ & \vec{s} = (f_{S,G}, f_{B,S}, f_{B,G}, n_G, n_S, n_B) \mid \\ & n_G, n_S, n_B \in \{1, 2, \dots, N_A - 2\}, \\ & n_G + n_S + n_B = N_A, \\ & f_{S,G}, f_{B,S}, f_{B,G} \in [-1, 1], \\ & f_{S,G} \times f_{B,G} \geq 0, \\ & \epsilon(f_{S,G})|f_{S,G}| + \epsilon(f_{B,G})|f_{B,G}| \leq 1, \\ & (1 - \epsilon(f_{S,G}))f_{S,G} + \epsilon(f_{B,S})|f_{B,S}| \leq 1, \\ & (1 - \epsilon(f_{B,S}))f_{B,S} + (1 - \epsilon(f_{B,G}))f_{B,G} \leq 1 \}. \end{aligned}$$

The three last constraints say that no more than 100% of the requests initially directed to one cluster can be redirected.

### III. A HEURISTIC-BASED CONTROLLER ALGORITHM

The problem to be solved by the autonomic LB can be cast as the following non-linear constrained optimization problem:

Find the policy vector  $\vec{s}_{\max}$  such that  $\vec{s}_{\max} = \arg \max_{\vec{s} \in \mathcal{S}} \{U_g(\vec{R}, \vec{X}) = h(\vec{R}, \vec{X}, \vec{s}, \mathcal{W})\}$ . Note that the function  $h$  is non-linear and does not have a closed form expression because the response time and throughput values have to be determined by solving a multiclass closed queueing network model. Although no closed form expression exists for solving multiclass closed queueing networks [17], solutions can be found through iterative or recursive algorithms. Section III-C provides more details on this. Moreover, the state space  $\mathcal{S}$  is typically very large. Therefore, using standard optimization techniques is not an option for an autonomic controller that needs to make real-time policy change decisions. For that reason, we present an autonomic LB that uses heuristic techniques.

The LB controller algorithm considers that time is divided into 30-second time intervals called *controller intervals (CI)*. Two control levels are implemented: 1) the f-policy is re-evaluated at the end of each CI, and the s-policy is re-evaluated by the controller at the end of every 10 CIs. Because of the switching cost of moving servers from one cluster to the other, the s-policy should be evaluated at a lower frequency than the f-policy.

Each server manages its own queue of requests. The requests in each queue are ordered by the timestamp of the original request such that the oldest request in the queue is the next to be serviced. The LB redirects a fraction of incoming requests from one cluster to another cluster according to the f-policy. The LB sends requests to servers within a cluster in a round-robin fashion. When the heuristic search completes, the autonomic controller may need to move one or more servers from one cluster to another to comply with a new s-policy. To move servers between clusters, the controller undertakes the following actions: 1) an empty, temporary list for storing requests is established for each cluster donating or receiving a server 2) before each server is moved, the requests in that server's queue are transferred to the temporary list of the donating cluster 3) within a receiving cluster, all requests in server queues are transferred to the temporary list for that receiving cluster 4) the servers are moved to comply with the new s-policy 5) the requests in the temporary lists are distributed round-robin to the server queues within each cluster.

The autonomic controller (see Fig. 3) consists of five elements: the autonomic agent (1), the heuristic search module (2), the utility function computation module (3), the performance model solver (4), and the workload monitor, not explicitly shown in Fig. 3, embedded in the controlled site (5).

#### A. Autonomic Agent

At the end of every CI, the autonomic agent (1) receives from the workload monitor a description  $\mathcal{W}$  of the workload intensity seen in the previous CI. The workload intensity is given by the number of customers of each category and their average think times. The autonomic agent then invokes the heuristic search module (2) to find an f-policy that maximizes business-value according to the utility function. At the end of every 10 CIs, the autonomic agent requests the heuristic component to find a combination of f-policy and s-policy that maximizes the utility function. In the case of a change in f-policy, the LB changes its routing tables to implement the new re-direct policy. In the case of a change in the server allocation, the LB informs the affected clusters (5) that servers have to be moved according to the values in  $\vec{n}_A$ .

#### B. Heuristic Search

The heuristic search component is given the current state  $\vec{s}_0 = (\vec{f}, \vec{n}_A)$  and the workload intensity  $\mathcal{W}$ . For requests asking for an optimal value of the f-policy only, the heuristic search is given a budget of  $B_{f,max}$  evaluations (200 in our implementation), while a budget of  $B_{s,max}$  evaluations (500 in our implementation) is provided when searching for optimal values of  $\vec{f}$  and  $\vec{s}$ . To reduce the size of the state space, searches for  $\vec{f}$  and  $\vec{s}$  are restricted to positive values within  $\vec{f}$ . This is a contrast with the f-policy only search which does allow negative values within  $\vec{f}$ .

We use hill-climbing as the heuristic to search the space  $\mathcal{S}$  of possible policies. When searching only for an optimum f-policy, we set the initial step on the values of  $f_{S,G}, f_{B,S}, f_{B,G}$  to  $\delta_f = 0.1$ . The neighboring states of state  $(f_{S,G}, f_{B,S}, f_{B,G}, n_G, n_S, n_B)$  are states of the form  $\vec{s}' = (f_{S,G} \pm \delta_f, f_{B,S} \pm \delta_f, f_{B,G} \pm \delta_f, n_G, n_S, n_B)$  subject to the constraints that  $\vec{s}' \in \mathcal{S}$ . We are currently assessing

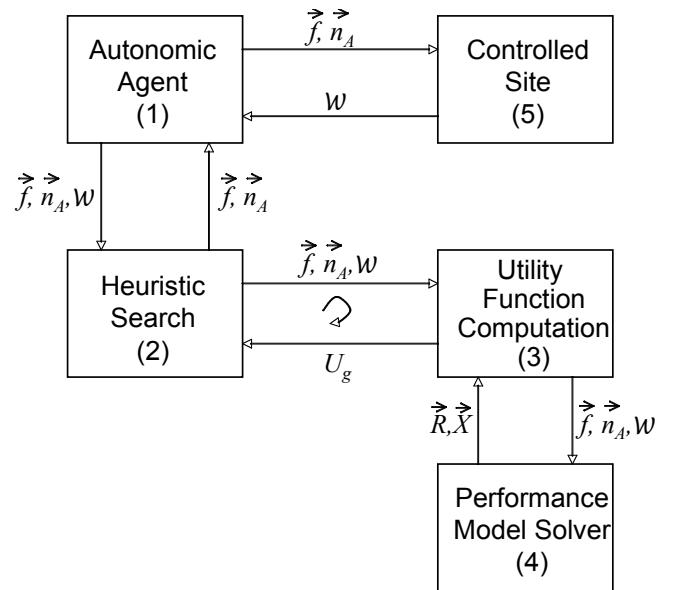


Fig. 3. Architecture of the autonomic controller.

the policy search performance of other heuristic search techniques including beam search, simulated annealing, genetic algorithms, evolution strategies, and particle swarm. Evolution strategies and particle swarm have shown particular promise.

The hill-climbing heuristic visits the solution that offers the greatest improvement in utility, and then evaluates the neighbors of that solution. For improved efficiency, the hill-climbing heuristic stores evaluations with their utility in a hash table. If the heuristic encounters a previously evaluated neighbor, it retrieves the score from the hash table to avoid an unnecessary evaluation. If no neighboring solution offers an improvement, the hill-climbing heuristic divides  $\delta_f$  by 10, unless  $\delta_f$  is already 0.001. When  $\delta_f$  is 0.001 and no neighboring solutions offer improvement, the heuristic search restarts at a randomly selected, legal solution with a  $\delta_f$  of 0.1. This process continues until the evaluation budget has been consumed. Testing has shown that the hill-climber on these policy landscapes is somewhat robust to the selection of step size parameters. The values used here are simple and provide reasonable performance. A more optimal selection of step size parameters may offer a slight improvement in search effectiveness.

In the search for an optimal s-policy, which occurs jointly with the search for an optimal f-policy, the neighboring states of state  $(f_{S,G}, f_{B,S}, f_{B,G}, n_G, n_S, n_B)$  are states of the form  $\vec{s}' = (f_{S,G} \pm \delta_f, f_{B,S} \pm \delta_f, f_{B,G} \pm \delta_f, n_G \pm \delta_s, n_S \pm \delta_s, n_B \pm \delta_s)$  where  $\delta_s = \lfloor N_a \times \delta_f / 2 \rfloor + 1$ , subject to the constraint that  $\vec{s}' \in \mathcal{S}$ . The variation of  $\delta_f$  follows the same approach as described above for the case of the f-policy only.

In order to evaluate the utility of a solution  $(f_{S,G}, f_{B,S}, f_{B,G}, n_G, n_S, n_B)$ , the heuristic search algorithm needs to obtain, from the performance model solver (3), the performance metrics (i.e., response times  $\vec{R}$  and throughputs  $\vec{X}$ ) for a given f-policy, s-policy, and workload  $\mathcal{W}$ .

### C. Performance Model Solver

The performance model solver (4) uses an analytic multi-class closed queueing network (QN) model, which is solved using the Approximate Mean Value Analysis (AMVA) technique [17]. A closed QN model is a tuple  $(\mathcal{D}, \mathcal{C}, \mathcal{W}, D)$  where  $\mathcal{D}$  is the set of  $K$  devices used to represent the servers of the Web site plus a device used to represent users' think times;  $\mathcal{C}$  is the set of user classes—i.e., gold, silver, and bronze in our case;  $\mathcal{W}$  is the set of workload intensity specifications for each class  $s$  given by the pair  $(M_s, Z_s)$  where  $M_s$  is the maximum number of class  $s$  customers that can submit class  $s$  requests and  $Z_s$  is the average think time for class  $s$  customers; and  $D = [D_{i,j}]$  is the  $K \times S$  matrix of service demands where  $D_{i,j}$  is the total average service time of requests of class  $s$  ( $s = 1, \dots, S$ ) at device  $i$  ( $i = 1, \dots, K$ ).

Our performance model has three classes ( $S = 3$ )—representing gold, silver and bronze users—and a number of devices that corresponds to the number of web servers plus the number of application servers plus the number of database servers plus one delay device to represent the think time of customers of each class.

The base service demand values (i.e., the service demands that would be obtained if there were only one server per cluster and no re-direction between clusters) for each user category in the application layer are denoted by  $D_{i,G}, D_{i,S}$ , and  $D_{i,B}$  at device  $i$  and categories gold, silver, and bronze, respectively. The values of  $D_{i,G}, D_{i,S}$ , and  $D_{i,B}$  are computed as

$$D_{i,s} = \sum_{\forall t} p_{t,s} \times D_{i,s}^t \quad (6)$$

where  $s \in \{\text{gold, silver, bronze}\}$ ,  $t$  is a generic transaction requested by a customer according to the CBMG,  $p_{t,s}$  is the percentage of transactions of type  $t$  requested by customers of category  $s$  during a session, and  $D_{i,s}^t$  is the service demand of transactions of type  $t$  at device  $i$  due to users of category  $s$ .

The values of  $f_{S,G}, f_{B,S}, f_{B,G}, n_G, n_S$ , and  $n_B$  are used to compute the values of the service demands at the devices in the QN model that represent the application servers. This is done by adding and subtracting according to the f-values of the redirection policy from the original service demand and then scaling the service demand for the number of servers in the cluster (from the s-policy).

Thus, the performance model solver can be seen as a function  $\mathcal{M}$  that takes as inputs a system state  $\vec{s} = (f_{S,G}, f_{B,S}, f_{B,G}, n_G, n_S, n_B)$  and the workload intensity  $\mathcal{W}$  and returns the pair  $(\vec{R}, \vec{X})$ . In other words,

$$(\vec{R}, \vec{X}) = \mathcal{M}(\vec{s}, \mathcal{W}). \quad (7)$$

### D. Utility Function Computation

The utility function computation (3) invokes the performance model solver (4) to determine the expected response time and bid throughput for each priority class and then uses Eq. (4) to compute the global utility. The use of AMVA makes it feasible for a performance model solver to be used by the autonomic controller, which may require a large number of evaluations of the model at each controller interval.

### E. Detailed Description

The detailed description of the controller is given in Algorithm 1. The following definitions are used in this description:

- Line 18 of the algorithm shows the operator  $\oplus_f$  defined as follows: if  $\vec{s} = (f_{S,G}, f_{B,S}, f_{B,G}, n_G, n_S, n_B)$  then  $\vec{s} \oplus_f x$  returns the set  $\{\vec{s}' = (f_{S,G} \pm x, f_{B,S} \pm x, f_{B,G} \pm x, n_G, n_S, n_B) \mid \vec{s}' \in \mathcal{S}\}$ , i.e., the set of all feasible policies obtained by modifying the f-policy by a step equal to  $x$  in all directions.
- Line 20 of the algorithm shows the operator  $\oplus_n$  defined as follows: if  $\vec{s} = (f_{S,G}, f_{B,S}, f_{B,G}, n_G, n_S, n_B)$  then  $\vec{s} \oplus_n k$  returns the set  $\{\vec{s}' = (f_{S,G}, f_{B,S}, f_{B,G}, n_G \pm k, n_S \pm k, n_B \pm k) \mid \vec{s}' \in \mathcal{S}\}$ , i.e., the set of all feasible policies obtained by modifying the s-policy by a step equal to  $k$  in all directions.

The parameter Type (see line 1) has a value equal to F to indicate that the controller must search for the best f-policy only. Otherwise, the controller searches for the best combination of f-policy and s-policy.

#### IV. EVALUATION OF THE LB CONTROLLER

We compared the autonomic LB with two other approaches: round-robin (RR) and dedicated servers (DS). In all three

---

##### Algorithm 1 Controller Algorithm

---

```

1: function Controller (Type,  $\vec{s}_{curr}$ ,  $\mathcal{W}$ )
2:     returns ( $\vec{s}_{best}$ )
3: Budget  $\leftarrow$  200
4: if Type  $\neq$  F then
5:     Budget  $\leftarrow$  500
6: end if
7:  $\delta_f \leftarrow$  0.1; NumEvals  $\leftarrow$  0;  $U_{best} \leftarrow$  -1.0;
8:  $\delta_s = \lfloor N_a \times \delta_f / 2 \rfloor + 1$ ;
9: while NumEvals < Budget do
10:     $U_{local} \leftarrow U_g(\mathcal{M}(\vec{s}_{curr}, \mathcal{W}))$ 
11:    if  $U_{local} > U_{best}$  then
12:         $\vec{s}_{best} \leftarrow \vec{s}_{curr}$ ;  $U_{best} \leftarrow U_{local}$ ;
13:    end if
14:    NumEvals  $\leftarrow$  NumEvals + 1;
15:    if NumEvals  $\geq$  Budget then
16:        break while loop
17:    end if
18:     $\mathcal{N} \leftarrow \vec{s}_{curr} \oplus_f \delta_f$ 
19:    if Type  $\neq$  F then
20:         $\mathcal{N} \leftarrow \mathcal{N} \cup (\vec{s}_{curr} \oplus_n \delta_s)$ 
21:    end if
22:    FoundImprovement  $\leftarrow$  False
23:    for all  $\vec{s} \in \mathcal{N}$  do
24:         $U_s \leftarrow U_g(\mathcal{M}(\vec{s}, \mathcal{W}))$ 
25:        if  $U_s > U_{local}$  then
26:             $\vec{s}_{local} \leftarrow \vec{s}$ ;  $U_{local} \leftarrow U_s$ ;
27:            FoundImprovement  $\leftarrow$  True
28:            if  $U_s > U_{best}$  then
29:                 $\vec{s}_{best} \leftarrow \vec{s}$ ;  $U_{best} \leftarrow U_s$ ;
30:            end if
31:        end if
32:    NumEvals  $\leftarrow$  NumEvals + 1;
33:    if NumEvals  $\geq$  Budget then
34:        break for loop and while loop
35:    end if
36: end for
37: if FoundImprovement then
38:    /* move to best neighbor */
39:     $\vec{s}_{curr} \leftarrow \vec{s}_{local}$ 
40: else
41:    if  $\delta_f > 0.001$  then
42:         $\delta_f \leftarrow \delta_f / 10$ ;
43:         $\delta_s = \lfloor N_a \times \delta_f / 2 \rfloor + 1$ 
44:    else
45:         $\vec{s}_{curr} \leftarrow$  random  $\vec{s}' \in \mathcal{S}$ 
46:         $\delta_f \leftarrow$  0.1;
47:         $\delta_s = \lfloor N_a \times \delta_f / 2 \rfloor + 1$ 
48:    end if
49: end if
50: end while
51: end function

```

---

approaches, round-robin load dispatching is used at the web tier, the database tier, and within each application tier cluster. Where the approaches differ is in the dispatch of requests to the application tier clusters. In the RR case, requests are dispatched round-robin to the clusters without regard for customer category. In the DS case, requests are always dispatched to the cluster corresponding to the customer category of the requester. Table II shows the number of servers in each tier and cluster for the different load balancing approaches. The allocation of servers for the DS approach optimizes global utility for the initial workload and is also used by the autonomic LB as the starting s-policy.

TABLE II  
NUMBER OF SERVERS IN EACH TIER AND CLUSTER FOR RR, DS, AND THE AUTONOMIC LB.

	$n_{ws}$	$n_G$	$n_S$	$n_B$	$N_A$	$n_{db}$
RR	99	33	33	33	99	2
DS	99	12	25	62	99	2
Autonomic LB	99	*	*	*	99	2

##### A. Description of the Experiments

The simulation was built using the CSIM 19 C++ library (www.mesquite.com). Transaction service times are exponentially distributed with a mean calculated from the experimental results in [14]. In all experiments, the autonomic controller was initialized with the policy ( $f_{S,G} = 0.0$ ,  $f_{B,S} = 0.0$ ,  $f_{B,G} = 0.0$ ,  $n_G = 12$ ,  $n_S = 25$ ,  $n_B = 62$ ), i.e., no redirection and the optimal number of servers per cluster for the initial workload as in the DS case. The response time SLO for all customer categories is a mean response time  $\leq 1.0$  seconds. The response time weights used in the utility function of Eq. (2) are  $w_G = 0.45$ ,  $w_S = 0.35$ , and  $w_B = 0.20$ .

Each load balancing approach was tested against 50 randomly generated loads, each with a duration of 480 minutes. Most web sites including auction sites experience dynamic loads each day that can sometimes include extreme phenomena such as flash crowds. Our goal for developing a dynamic load was to randomly generate realistic and challenging load tests that would sometimes include extreme phenomena such as a flash crowd. For a more realistic test, we wanted the loads offered by the customer categories to be moderately but not perfectly correlated over time. Before the simulation begins, we randomly generate a load schedule comprised of load vectors,  $\vec{N}_t$ , for each minute  $t$  of the test. Each load vector,  $\vec{N}_t = (N_{t,G}, N_{t,S}, N_{t,B})$ , contains a target number of concurrent customers of each category at the beginning of minute  $t$ . For the first five minutes of each load schedule,  $\vec{N}_t = (5000, 10000, 35000)$  allowing the customer population to become more distributed throughout the CBMG state space.

In generating a dynamic load schedule, time is divided into minutes and sequences of minutes are aggregated into variable length intervals called *epochs* denoted by  $\tau_1, \tau_2, \dots, \tau_k$  (see Fig. 4). The duration of an epoch is an integer number of minutes determined from an exponential distribution with an average of 5.0 minutes rounded up to the nearest minute.

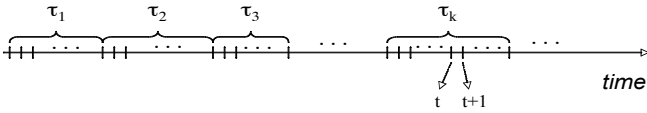


Fig. 4. Process for generating schedules for dynamic loads.

The first step in the generation of a dynamic load schedule is the determination of the total number of concurrent customers  $N_k(t)$  for epoch  $\tau_k$  and time  $t$ . The number of customers during an epoch  $\tau_k$  varies at a rate, given in customers per minute, which is sampled at the beginning of each epoch from a Normal distribution with zero mean and standard deviation equal to 1,000 customers/minute. Web site workloads also tend to be noisy, which may result in autonomic performance agents overreacting to phantom phenomena or underreacting to real phenomena. For that reason, we add a Gaussian noise to the total number of concurrent customers at every minute  $t$ . This value is sampled at every minute from a Normal distribution with zero mean and standard deviation equal to 2,000 customers.

The total number of concurrent customers at time  $t + 1$  in epoch  $\tau_k$  can be written as a function of the total number of customers at time  $t$  in the same epoch as  $N_k(t + 1) = N_k(t) + x_\tau^k \times 1 \text{ minute} + g_{t+1}$ , where  $x_\tau^k$  is the rate of variation of the total number of customers during epoch  $\tau_k$  and  $g_{t+1}$  is the value sampled from the Normal distribution that represents the noise at time  $t + 1$ .

The second step consists in determining the mix of customers in each category (gold, silver, and bronze) at time  $t$  for each epoch. The mix of customers at time  $t + 1$  is derived from the mix at time  $t$  by varying the percentage of customers in each category so that the sum of the percentages remains at 100%. We allow bidirectional moves of percentages between 1) gold and silver, 2) silver and bronze, and 3) bronze and gold. The direction of each move is determined by a flip of a coin and the size of the move in percentage points is sampled from an exponential distribution. To keep the composition of customers within a reasonable range, each customer category  $s$  has a tether percentage,  $\theta_s$ . Random moves towards  $\theta_s$  tend to be larger than moves away from  $\theta_s$ . This is done by shrinking the mean of the distribution for moves away from a tether point. This allows the percentages to fluctuate with no hard boundaries but still stay within a reasonable range. The tether percentages used to generate the dynamic loads tests were  $\theta_G = 10\%$ ,  $\theta_S = 20\%$ , and  $\theta_B = 70\%$ . The parameters (e.g.,  $\theta_G$ ) used to generate the dynamic load schedules are not derived from the data collected in [15] but were selected to produce reasonable and interesting experiments.

Once the full load schedule has been determined, the simulation begins. The simulation always starts with no customers in the system. When a customer chooses to end a session as a result of a CBMG state decision, the customer restarts the session after waiting for an exponentially distributed time with a mean of 3.0 seconds. This delay is also used before a

customer starts their first session.

## B. Experimental Results

The tests provide a wide range of load compositions and load levels. The mean load profile of the randomly generated dynamic test set is shown in Table III. Overall, the average number of customers in the dynamic load tests is about 61,000 concurrent customers, somewhat higher than the starting number of 50,000 customers. The average range in total number of customers is over 106,000 customers. The ranges in number of customers for each customer category are relatively large as well.

TABLE III  
AVERAGE MAXIMUM, MEAN, AND MINIMUM CUSTOMER COUNTS.

	Customer Category			Total
	Gold	Silver	Bronze	
Maximum	17,151	30,033	85,286	119,401
Mean	6,355	12,391	42,561	61,308
Minimum	869	2,060	8,760	12,895

The mean workload composition is depicted in Table IV. As expected, the mean composition of the workload is fairly close to the tether percentages. However, the composition within the tests shows considerable variation with an average range of 15.5% for gold, 22.6% for silver, and 27.9% for bronze. When combined with the distribution of load levels, this diversity in workload composition should pose a significant challenge to the the three load balancing approaches.

TABLE IV  
AVERAGE MAXIMUM, MEAN, AND MINIMUM PERCENTAGE OF TOTAL CUSTOMER COUNT.

	Customer Category		
	Gold	Silver	Bronze
Maximum	18.4	31.9	83.9
Mean	10.2	20.2	69.6
Minimum	2.9	9.3	56.0

Table V shows the 99% confidence intervals for the mean global utility produced by the three approaches during all 50 tests. All three confidence intervals are clearly separated; the autonomic LB generated significantly higher global utility than either the dedicated or round-robin approaches. Figure 5 shows the utility differences between the approaches over time. In the first hour of the tests, the load is generally near the initial settings, and the performance disparity at this load level is not large. As time passes, the loads become more varied, and the autonomic LB provides superior performance. To better understand why the autonomic LB provided outperforms the other approaches, we examine in detail a representative example, test number 46.

The load schedule for test 46 is shown in Fig. 6. The most obvious feature of this schedule is that the load grows substantially over time reaching a peak of just over 200,000 simultaneous total customers at minute 420. The number of bronze customers also peaks at minute 419 with 152,000 customers. The number of silver customers peaks later at

TABLE V  
99% CONFIDENCE INTERVAL FOR MEAN GLOBAL UTILITY.

	Lower Bound	Sample Mean	Upper Bound
Autonomic LB	169.3	189.9	210.5
DS	112.4	124.8	137.3
RR	95.9	109.7	123.4

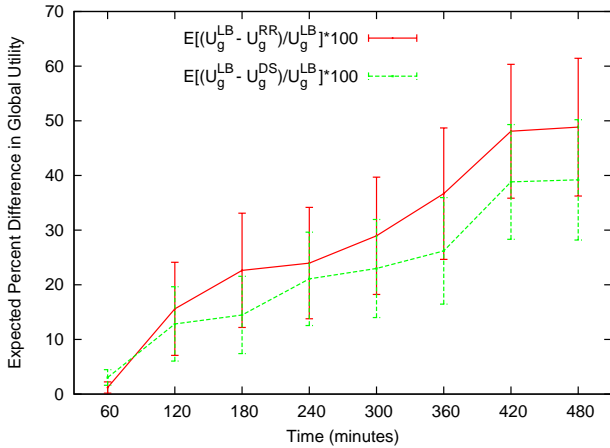


Fig. 5. Expected percent difference in  $U_g$  between autonomic LB and RR and DS with 95% confidence intervals.

minute 444 with nearly 47,000 customers, while the number of gold customers peaks at minute 416 with over 35,000 customers.

When subjected to the load depicted in Fig. 6, the three load balancing approaches produced the global utilities also seen in Fig. 6. The three approaches produce similar global utility results from the start of the test through an initial drop off around minute 45 until minute 85 when the autonomic LB begins generating marginally more global utility. At this point the bronze response time in the DS approach is inching towards a SLO violation, while the RR approach generates less bid throughput than the autonomic LB because the autonomic LB favors gold and silver users who are more likely to submit bids. When the load spikes at minute 133, the RR approach violates all of its SLOs, while the DS approach violates the bronze SLO and nearly violates the gold SLO. The autonomic LB allows the bronze SLO to be violated while preserving the gold and silver SLOs. This behavior repeats with more severe consequences during the load peak at minute 183. When the overall load ramps up at minute 225, the RR approach violates all of its response time SLOs at approximately the same time. As a result, the RR utility collapses and does not recover during the course of the test. By minute 255, both the autonomic LB and DS approaches are violating the bronze customer response time SLO, but the autonomic LB gets more utility by better satisfying the gold and silver response time SLOs. At minute 257, there is a sharp increase in the number of gold customers and the DS approach fails to satisfy the gold response time SLO—this is reflected in the poor utility production of the DS approach until minute 300. The DS

approach again violates the gold response time SLO during load spikes at minutes 307 and 335. Beyond minute 355, the DS approach struggles to meet any of its response time SLOs under the extreme load.

Figure 6 also shows the autonomic controller’s policy selections. For the first 185 minutes of the test, the autonomic controller makes small changes to the s-policy (i.e., the server allocation policy). The f-policy (i.e., the request redirection policy) makes a substantial adjustment between minute 100 and 116 redirecting nearly 40% of silver requests to the gold cluster. This helps the autonomic controller to initially respond to a spike in the number of silver users. The load peak at 140 minutes induces a large exchange of servers between the bronze cluster and the gold cluster. When the temporary peak load subsides, the controller returns to the previously used s-policy allocation. When the load begins to ramp up, the autonomic controller uses the s-policy to shift resources away from the bronze cluster to the gold and silver clusters. At minute 240, when the number of silver users declines and the number of bronze and gold users grows, the autonomic controller moves servers from the silver cluster to the bronze and gold clusters. The autonomic controller then relies on the f-policy to make sure that the silver cluster does not become over burdened with requests. At minute 355, the number of bronze customers drops while the number of silver and gold customers increases sharply. The resulting s-policy moves most of the bronze servers to the gold and silver clusters. When the number of gold customers passes the number of silver customers near minute 375, the autonomic controller responds by shifting a majority of the application servers to the gold cluster. The number of silver customers passes the number of gold customers near minute 425, and the s-policy begins moving gold cluster servers back to the silver cluster.

The analytic model used by the autonomic controller to evaluate policies provided satisfactory estimates. Considering loads with more than 25,000 users, the average percent difference between the observed global utility ( $U_g$ ) and the analytic queueing model’s prediction of  $U_g$  was  $-1.30\% \pm 0.03\%$  at the 99% confidence level. The single-threaded simulations were executed on systems with two dualcore 2.6 GHz Opteron CPUs. Searches for f-policies (budget of 200 evaluations) took an average  $1.45 \pm 0.18$  seconds at the 95% confidence level, while searches for f and s-policies (budget of 500 evaluations) took an average of  $4.01 \pm 0.49$  seconds at the 95% confidence level.

## V. CONCLUDING REMARKS AND FUTURE WORK

This paper described the design of an autonomic controller that can search for optimal load balancing and resource allocation policies at an e-commerce site according to a business-oriented utility function. This approach is general enough that it could be used at most e-commerce websites. We have thoroughly tested this controller by simulating a large e-commerce site using workloads developed from measurements of real customer behavior [15] and real service demands [14]. We have designed and applied a new method for generating

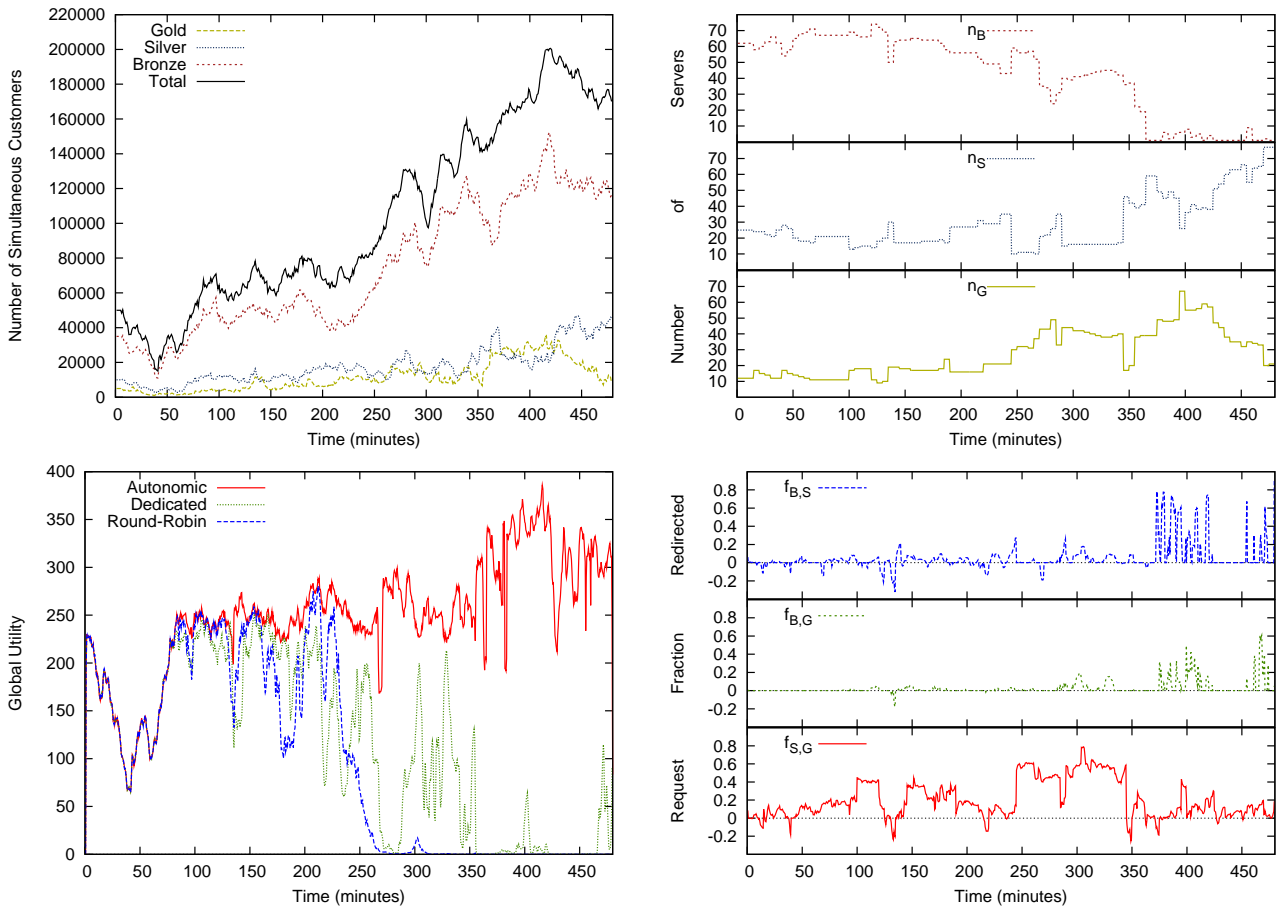


Fig. 6. Results from test 46 over simulated time including load (top left), s-policy values (top right), global utility (bottom left), and f-policy values (bottom right).

dynamic workloads that include extreme phenomena such as flash crowds.

Our experiments provided the following results: 1) the autonomic LB matches the performance of the DS and RR approaches at low load levels, 2) the autonomic LB generates substantially more utility than DS or RR at higher load levels, 3) the autonomic LB shows a significant utility benefit at the 99% confidence level over the DS and RR approaches, and 4) a detailed examination of the results shows that the autonomic controller redirects requests and allocates resources in a manner that maximizes bid throughput while minimizing response times.

We have also tested an autonomic LB that implements only f-policies and an autonomic LB that implements only s-policies. Due to space limitations the detailed results are not included in this paper, however a summary of the results may be instructive. The f-only autonomic LB performed better than both the RR and DS approaches but worse than the f&s autonomic LB presented here at a 99% confidence level. The s-only autonomic LB provided a small (1-2%) but statistically significant performance improvement over the f&s approach. The s-only approach only provides this performance benefit if the s-policy is re-evaluated every 30 seconds as opposed

to every 5 minutes as in the f&s approach. However, the s-only approach may suffer more from the following drawbacks not considered in our simulation: 1) reduced cache efficiency when servers are moved since different user classes may have preferences or access to different sets of objects 2) extra overhead in redistributing queues because servers are moved more frequently.

We believe that our autonomic LB could be extended in a number of ways. It could be extended to handle server failures by allowing the total number of application servers to vary. Workload forecasting similar to that found in [1] could further improve the autonomic LB's performance. Machine learning methods similar to those described in [13] could enable on-line discovery of customer behavior patterns and inform customer priority.

Reducing power consumption in the data center through autonomic computing is an active research topic [5]. We see a few ways in which our autonomic controller might be extended to improve energy efficiency in the data center. The autonomic controller could consider suspending or shutting down servers by 1) relaxing the policy space constraint  $n_G + n_S + n_B = N_A$  to  $n_G + n_S + n_B \leq N_A$ , 2) adding an expression for the utility savings of a shutdown server, 3) adding a utility expression for

the switching cost and 4) adding a utility expression for the resource shortage risk involved in shutting down a server. A more sophisticated approach to reducing power consumption would be to 1) add a voltage scaling parameter for each cluster to the resource allocation policy space and 2) scale service demands appropriately for the voltage scaling during the heuristic search. The voltage-scaling approach creates a nine-dimensional search space that may prove difficult to search with the limited evaluation budget available.

#### ACKNOWLEDGEMENTS

The work of Daniel Menascé is partially supported by NSF award no. CCF-0820060.

#### REFERENCES

- [1] M. N. Bennani and D. A. Menascé, "Resource allocation for autonomic data centers using analytic performance models," in *Proc. IEEE International Conference on Autonomic Computing (ICAC05)*, Seattle, WA, Jun. 2005, pp. 229–240.
- [2] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing – degrees, models, and applications," *ACM Computing Surveys*, vol. 40, no. 3, Aug. 2008.
- [3] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, "Generating adaptation policies for multi-tier server applications in consolidated server environments," in *Proc. IEEE International Conference on Autonomic Computing (ICAC08)*, Chicago, IL, Jun. 2008, pp. 23–32.
- [4] D. A. Menascé, R. Dodge, and D. Barbará, "Preserving QoS of e-commerce sites through self-tuning: A performance model approach," in *Proc. ACM Conference on E-commerce*, Tampa, FL, Oct. 2001, pp. 224–234.
- [5] T. Horvath, K. Skadron, and T. Abdelzaher, "Enhancing energy efficiency in multi-tier web server clusters via prioritization," in *Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS'07)*, Long Beach, CA, Mar. 2007, pp. 1–6.
- [6] G. Tesouro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation," in *Proc. IEEE International Conference on Autonomic Computing (ICAC06)*, Dublin, Ireland, Jun. 2006, pp. 65–73.
- [7] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "On the use of fuzzy modeling in virtualized data center management," in *Proc. IEEE International Conference on Autonomic Computing (ICAC07)*, Jacksonville, FL, Jun. 2007, p. 25.
- [8] A. Bivens, C. Chhuor, D. Dillenberger, G. Ferris, J. Fenton, and W. Chou. (2006, Apr.) Autonomic load balancing, part 1: Cisco content switching module. IBM Developer Works. [Online]. Available: <http://www.ibm.com/developerworks/library/ac-ewlmlload1/index.html>
- [9] D. Breitgand, R. Cohen, A. Nahir, and D. Raz, "On fully distributed adaptive load balancing," *Lecture Notes in Computer Science*, vol. 4785, pp. 74–85, Sep. 2007.
- [10] W. S. Li, D. C. Zilio, V. S. Batra, M. Subramanian, C. Zuzarte, and I. Narang, "Load balancing for multi-tiered database systems through autonomic placement of materialized views," in *Proc. IEEE International Conference on Data Engineering (ICDE06)*, Atlanta, GA, Apr. 2006, p. 102.
- [11] Y. Diao, C. W. Wu, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Parekh, C. Garcia-Arellano, M. Carroll, L. Chu, and J. Colaco, "Comparative studies of load balancing with control and optimization techniques," in *Proc. American Control Conference*, Portland, OR, Jun. 2005, pp. 1484–1490.
- [12] L. Zhang and D. Ardagna, "SLA based profit optimization in autonomic computing systems," in *Proc. International Conference on Service Oriented Computing (ICSOC04)*, New York, NY, Nov. 2004, pp. 173–182.
- [13] N. Poggi, T. Moreno, J. L. Berral, R. Gavaldá, and J. Torres, "Self-adaptive utility-based web session management," *The International Journal of Computer and Telecommunications Networking*, vol. 53, no. 10, pp. 1712–1721, Jul. 2009.
- [14] D. A. Menascé and V. Akula, "A business-oriented load dispatching framework for online auction sites," in *Proc. IEEE International Conference on Quantitative Evaluations of Systems (QUEST07)*, Edinburgh, Scotland, Sep. 2007, pp. 249–258.
- [15] V. Akula and D. A. Menascé, "Two-level workload characterization of online auctions," *Electronic Commerce Research and Applications Journal*, vol. 6, no. 2, pp. 192–208, Jun. 2007.
- [16] D. A. Menascé, V. Almeida, R. Fonseca, and M. Mendes, "A methodology for workload characterization for e-commerce servers," in *Proc. ACM Conference on E-commerce*, Denver, CO, Nov. 1999, pp. 119–128.
- [17] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*. Upper Saddle River, NJ: Prentice Hall, 2004.