QoS-Driven Server Migration for Internet Data Centers

S. Ranjan, J. Rolia, H. Fu, and E. Knightly

Abstract—**Many organizations have chosen to host Internet applications at Internet Data Centers (IDCs) located near network access points of the Internet to take advantage of their high availability, large network bandwidths and low network latencies. Current IDCs provide for a dedicated and static allocation of resources to each hosted application. Unfortunately, workloads for these sites are highly variable, leading to poor resource utilization, poor application performance, or both. In this paper, we develop a framework for QoS-driven dynamic resource allocation in IDCs. Termed QuID (Quality of Service Infrastructure on Demand), the framework's contributions are threefold. First, we develop a simple adaptive algorithm to reduce the average number of servers used by an application while satisfying its QoS objectives. Second, we develop an optimal off-line algorithm that bounds the advantage of any dynamic policy and provides a benchmark for performance evaluation. Finally, we perform an extensive simulation study using traces from large-scale E-commerce and search-engine sites. We explore the gains of the QuID algorithms as a function of the system parameters (such as server migration time), algorithm parameters (such as control time scale), and workload characteristics (such as peak-to-mean ratio and autocorrelation function of the request rate).**

I. Introduction

Both large- and small-scale web content publishers and application service providers are increasingly employing Internet Data Centers (IDC) to host their services. IDCs provide physical resources and space for customer servers, systems management services to help manage the servers, and high speed Internet access.

Clients of such hosted services have performance level expectations, including request throughput, response time, transfer bandwidth, and probability of session completion. In current IDC architectures with thousands of servers and hosted sites, each service must be uniquely engineered to attain its desired performance profile. This may be possible if each site's workload remains static, or at least remains within a pre-specified bound. However, evidence from workload analysis indicates that demands are highly variable (often changing by a factor of between 2 and 20 throughout the day) and unpredictable (with demand surges due to special events) [2]. Since current IDC architectures are manually configured and cannot automatically adapt to these workloads, they result in poor resource utilization or significant performance degradation when loads exceed capacity.

To address these limitations, infrastructure-on-demand architectures have been recently introduced to *dynamically* share the vast computing resources of IDCs [1], [18]. The central idea of such architectures is to securely and adaptively migrate servers from one site to another according to workload demands. For example, if one hosted application is experiencing a demand surge, servers can be migrated to this site from a shared pool or away from underloaded sites. While such architectures are an important step in the design of next-generation IDCs, dynamic resource management techniques are still needed to exploit the efficiency gains of a dynamic policy while ensuring application QoS requirements are satisfied. Furthermore the effectiveness of migration techniques must be assessed.

In this paper, we introduce QuID (Quality of Service for Infrastructure on Demand), a framework for QoS-driven dynamic resource allocation in IDCs. Our contributions are as follows.

First, we devise a simple adaptive algorithm (QuID-online) to provide hosted clients with the same or better performance profile as with a dedicated set of servers. The key is to exploit long-time-scale variations in the hosted application workloads. In this way, QuID-online provides a practical dynamic algorithm to realize significant reductions in resource requirements as compared to static approaches.

Second, we devise an optimal off-line algorithm (QuID-optimal) that computes the minimum resources required for a given application arrival and demand sequence. In particular, QuID-optimal jointly optimizes the server allocation decisions, request load balancing decisions, and CPU scheduling decisions. While QuID-optimal is clearly not implementable in actual systems, it serves two important purposes. First, it provides a performance benchmark for evaluating practical algorithms such as QuID-online. Second, it allows us to explore the fundamental limits of dynamic IDC architectures in that it provides a bound on the resource savings that can be achieved by any dynamic allocation policy subject to satisfying an application's QoS requirements.

Finally, we perform an extensive set of simulation experiments using traces from a large E-commerce site (EC trace) and a major search engine site (SE trace). With trace-driven simulations, we (1) provide a proof-of-concept demonstration of QoS infrastructure-on-demand and quantify the available efficiency gains; (2) compare the available gains of QuID-online and QuID-optimal as compared to static allocation; and (3) study the performance impact of key factors such as server migration time, control time scale, and trace characteristics.

With the experimental study, we quantify the workload, system, and algorithm characteristics essential to realizing significant gains using QuID. Example findings are that for the EC trace with a peak-to-mean ratio near 1.5 and a strong autocorrelation value above 0.99 at a lag corresponding to a 5 minute server migration time, QuID-online reduces the resource requirements over a static allocation policy by 25%. For traces with peak-to-mean ratios near 5, this reduction is increased further to 68%, despite the traces having more rapidly decaying autocorrelation functions, as the workload retains strong autocorrelation values at the key migration and control time scales.

The remainder of this paper is organized as follows. In Sec-

S. Ranjan, H. Fu, and E. Knightly are with Rice University, Houston, TX (http://www.ece.rice.edu/networks). J. Rolia is with HP Laboratories, Palo Alto, CA (jar@hpl.hp.com). This research is supported by HP Labs and NSF grants ANI-0085842 and ANI-0099148.

tion II, we describe the multi-tier IDC architecture and dynamic server-migration system model. In Sections III and IV, we introduce the QuID-online and QuID-optimal algorithms, which we evaluate via trace-driven simulations in Section V. Finally, we discuss related work in Section VI and conclude in Section VII.

II. Background

In this section we describe the basic system architecture of an Internet Data Center that can dynamically migrate servers among hosted applications. Moreover, we describe our system model that serves as an application-tier IDC abstraction to study the QuID framework.

Throughout this paper, a session refers to a sequence of affiliated requests for an end user. For example, an e-commerce session would consist of a number of requests including the download of catalog pages, searches for specific items, management of a "shopping cart", and purchase confirmation and payment authentication at "check out".

A. Architecture

Figure 1 depicts the four-tier architecture prevalent in today's IDCs. To illustrate this architecture, consider the requests of an e-commerce session. First, the *access tier* routes requests to the correct server cluster and performs basic firewall functions such as intrusion detection. Second, upon arriving at the *web tier*, load balancers may parse the request's URL and route it to a web server typically according to a load-balancing policy (e.g., using round robin or more sophisticated policies as in reference [5]). If the request is for a static web page, a server in the web tier serves the requested page. If the request is for an e-commerce function, it is routed to the *application tier*. The application tier stores the state of the request's session such as the contents of the shopping cart and performs operations such as purchase processing. Finally, in cases where the session contains requests such as database searches, the request is serviced by the *database tier*.

In scenarios such as the e-commerce example above, each session has associated state information (e.g., shopping cart contents) that must be shared by all requests of the session. Since it is inefficient to transfer session state between servers, each session must be associated with a particular application server. This requirement is referred to as *session affinity*. Sessions are assigned a unique identifier (session ID) that is passed with each request. Web servers use the session ID to route requests to the proper application server. To allocate new sessions to servers, the web servers can use round robin or other more sophisticated measurement-based load balancing techniques.

B. System Model

In this paper, we consider a simplified abstraction of the multi-tier IDC architecture to explore the issue of QoS-driven dynamic server migration. In particular, we consider the system model illustrated in Figure 2. The system model depicts a single application tier of servers as well as the dispatcher that allocates sessions and requests to servers as described above. For a single hosted application, the system can be viewed as dynamically migrating servers to and from a shared pool according to the workload and QoS requirements of the hosted application.

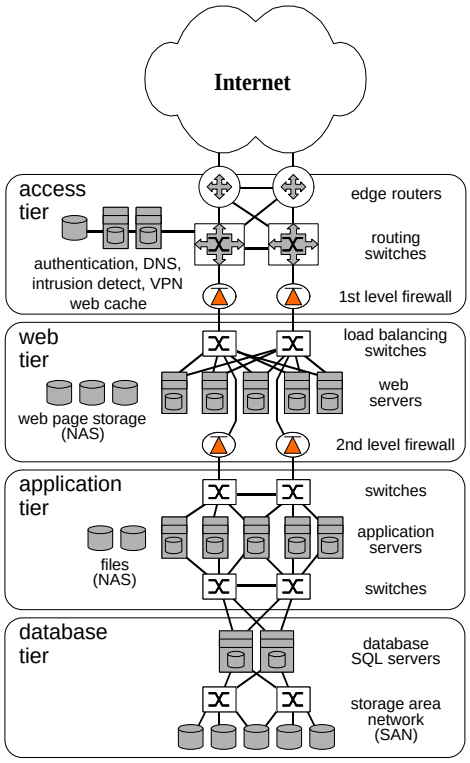


Fig. 1. IDC Four Tier Architecture

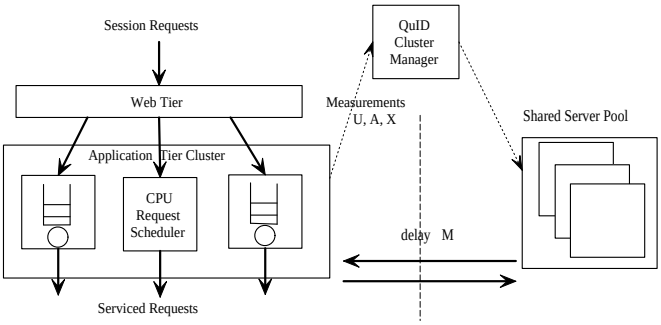


Fig. 2. System Model

Importantly, there is a migration time M associated with migrating a server from the shared pool and into a particular application's server cluster. There are several ways this can be accomplished. For example a migration time could include the booting of an operating system and start of an application, or it may include the time needed to restore a paused/migrated image of an application in execution. We consider migration times between 1 minute and 10 minutes in our experiments.

Similarly, releasing a server from a cluster also takes time, as even after web servers stop sending new sessions to the application-tier server, all sessions currently in progress at the server must be completed or timed out. Based on server logs, we expect typical "warm-down times" to migrate out servers to be in the range of 1 to 15 minutes.

III. An Algorithm for QoS-Driven Infrastructure on Demand

In this section we present an algorithm for server migration in dynamic IDCs that attains predictable Quality of Service. The purpose of the technique is to maintain a targeted cluster-wide average CPU utilization by acquiring and releasing servers in response to changes in load. With simple queueing theoretic arguments and simulation experiments, we show that QuID-online's utilization-target provides an effective mechanism for maintaining cluster-wide QoS. Moreover, by time-averaging workload variations and (necessarily) gradually migrating new servers into and out of the cluster, the algorithm minimizes the impact of rapid fluctuations in load and instead responds to long-time-scale trends.

A. QuID-online

To achieve these goals, the online algorithm requires measurements characterizing the state of the system. Thus, the QuID-online cluster manager depicted in Figure 2 periodically queries the application servers for summary workload information. The measurement interval, which we denote by τ, is typically on the order of minutes in duration. Thus, every τ the cluster manager queries servers for the following measurements: X, the number of request completions in the previous measurement interval; A, the number of request arrivals in the previous measurement interval; and U, the average CPU utilization (over all servers) in the previous measurement interval.[1] Note that average CPU utilizations are generally available from operating system monitors and application level measures such as request arrivals and completions per interval are typically available from server logs. In both cases, this information is summarized at the servers and communicated to the cluster manager.

Given N, the current number of servers, as well as ρ, the target utilization, QuID-online computes N', the required number of servers for the next interval as follows:

1. $D = U/X$, compute the average demand per completion;
2. $U' = \max(A, X)D$, compute the normalized utilization based on the current number of servers and all arrivals;
3. $N' = \lceil NU'/\rho \rceil$, compute the number of servers needed to achieve the target utilization ρ.

Step 1 computes D using cluster-wide aggregate utilization and completion measurements U and X, as well as the utilization relationship $U = XD$. Step 2 normalizes the CPU utilization with respect to the current number of servers and the number of arrivals for the measurement interval. When the system is under increasingly heavy load, arrivals may exceed completions and the computation of U' takes this into account. This allows QuID-online to react more quickly to increases in load than via the use of X or U alone. Moreover, we use the maximum of A and X to avoid releasing servers too quickly, thereby making the algorithm less sensitive to short-term decreases in load. Thus, the aggregate CPU utilization U' is the normalized utilization. In Step 3, we compute the upper bound on the number of servers N' needed to maintain the target utilization ρ.

[1] Namely, $U = (U_1 + U_2 + \cdots + U_N)/N$ from the reported values of individual CPU utilizations.

QuID-online initiates requests to acquire or release servers whenever $N' \neq N$. However, note that the overhead due to server migration time prohibits rapidly changing the number of servers. There are several aspects of the algorithm which mitigate the effects of this overhead. First, the measurement interval τ can be increased to average out short-term load fluctuations. Second, we note that the migration time itself provides a damping mechanism and we employ an additional policy as follows. If more servers are needed $(N' > N)$, servers previously warming down in preparation for migration out of the cluster are reinstated into the cluster in a last-in-first-out order before any requests are made to acquire additional servers. Similarly, if fewer servers are needed $(N' < N)$, servers previously warming up in preparation to join the cluster are released in a last-in-first-out order before any requests are made to release servers doing useful work. Finally, we incorporate the overhead of migration time by accounting for servers migrating to and from the cluster in the computations of the average numbers of servers.

B. Discussion

QuID online's use of a target CPU utilization ρ as a control parameter for QoS is justified as follows. First, assume that servers have a large number of threads so that requests rarely incur software blocking. Moreover, consider that threads are scheduled in a first-in-first-out manner without preemption by their server's CPU. Furthermore, the individual sessions constituting the total workload can be considered to be independent. In this scenario, a server cluster as in Figure 2 can be modeled as a $G/G/N$ system which has a mean response time R under heavy traffic given by [16]:

$$R = \frac{U}{X} + \frac{\sigma_a^2 + \frac{\sigma_b^2}{N^2}}{2\bar{t}(1-U)}$$

where σ_a^2 and σ_b^2 are the variance of interarrival and service times, respectively, and $\bar{t}$ is the mean interarrival time. Since, the response time R is determined by the utilization U, we maintain a target utilization ρ by controlling the number of servers N.

Regardless, QuID-online does not attempt to relate R and U directly and makes no such assumptions about inter-arrival or service time distributions. However, the relationship between response time and target utilization can be determined empirically for a particular workload. We utilize this methodology in Section V and show how a proper setting of ρ can control cluster-wide QoS.

Finally, consider the following numerical example. Suppose the target utilization is $\rho = 0.5$, the number of servers is $N = 14$. Moreover, consider that over the previous measurement interval, we have $X = 500$ completions, $A = 520$ request arrivals, and a measured cluster utilization of $U = 0.6$. In this case, we have $D = 0.6/500 = 0.0012$ and $U' = 520(0.0012) = 0.624$ such that the required new number of servers is given by $N' = \lceil 14(0.624)/0.5 \rceil = 18$. Since $N' > N$ a request is initiated to acquire $N' - N = 4$ additional servers for the cluster.

IV. Optimal Off-line Algorithm

In this section we develop QuID-optimal, an optimal off-line algorithm for resource allocation in dynamic IDCs. QuID-

optimal provides a benchmark for evaluation of practical algorithms by computing the *minimum* number of servers required by a server migration algorithm such that the application's response time requirement (QoS) is bounded cluster-wide. Consequently, it characterizes the maximal available gain of any dynamic policy as compared to a static policy.

QuID-optimal uses as its inputs a workload trace, i.e., a sequence of session and request arrival times as well as the corresponding server CPU processing time required to service each request. For a particular migration time and maximum response time, the algorithm jointly computes the best sequence of dispatching decisions (allocation of requests to servers), scheduling decisions (when to serve each request), and dynamic server allocation decisions (how many servers are required to ensure all request response times are within the required QoS bound). The solution, while not realizable in practice, provides the "best" decision sequence among all three dimensions in that it provides the minimum server solution subject to the maximum response time QoS requirement.

To solve this problem, we first formulate the dynamic server allocation problem as a constrained optimization problem. Next we transform the non-linear problem into a linear programming problem and show that the solutions of the two problems are equivalent. Finally, we discuss a simplified and more computationally feasible (but still not on-line realizable) bound that relaxes some of the most computationally intense constraints.

A. Problem Formulation

Consider a set of requests I such that request $i \in I$ has arrival time a_i and workload or required server processing time w_i. Moreover, consider that the system has migration time M and control time scale τ, and that the application QoS requirement is a maximum response time r^*. Moreover, each request must be serviced by a single server.

As described above, our objective is for each server to service all queued requests according to an optimally computed non-preemptive schedule, optimally computed dispatching, and a minimum number of servers.[2]

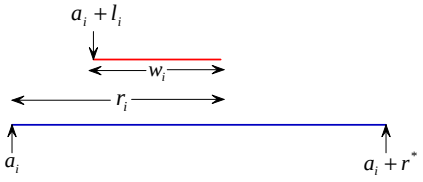


Fig. 3. Workload i

Define l_i ($l_i = 1, 2, \cdots, r^* - w_i + 1$) such that request i begins service at time $a_i + l_i$ and define r_i such that request i completes service at time $a_i + r_i$. Thus, as illustrated in Figure 3, r_i is the response time and we say that request i is being serviced with schedule l_i.[3] Let the indicator z_i^j be defined such that $z_i^j = 1$ if request i is serviced by server j, and $z_i^j = 0$ otherwise. Moreover, let the indicator $y_i^{jl} = 1$ if request i is serviced by server j under schedule l. Finally, let the indicator $x_{it}^{jl} = 1$ if at time t, request i is being serviced by server j under server schedule l.

Denote J as the set of servers, L_i as the set of feasible schedules of request i, $T_i = (a_i, a_i + r^*]$ as the interval of arrival time to departure deadline of request i, S as the set of sessions and s_m as the set of requests in session m, $s_m \in S$. Let T be the trace duration, $K = \frac{T}{\tau}$ be the number of measurement intervals, N_k be the number of servers at the k-th measurement interval, and $N_k \in \{0, 1, 2, \cdots\}$.

Our objective is to jointly optimize the request dispatching decisions z_i^{j*}, CPU scheduling decisions y_i^{jl*}, and dynamic server allocation sequence N_k^* that minimizes the required resource C, expressed as

$$\min\{M \times \sum_{k=1}^{K} (N_k - N_{k-1})^+ + \tau \times \sum_{k=1}^{K} N_k\}, \qquad (1)$$

subject to

$$(C1) \quad N_k \geq \sum_{i \in I} \sum_{j \in J} \sum_{l \in L_i} x_{it}^{jl}, \quad (k-1)\tau + 1 \leq t \leq k\tau, \quad k \in \{1, 2, \cdots, K\}$$

$$(C2) \quad \sum_{j \in J} \sum_{l \in L_i} \sum_{t \in T_i} x_{it}^{jl} = w_i, \quad i \in I$$

$$(C3) \quad x_{it}^{jl} = 0, \quad t \notin T_i, \; i \in I, \; l \in L_i, \; j \in J$$

$$(C4) \quad \sum_{l \in L_i} x_{it}^{jl} + \sum_{l \in L_k} x_{kt}^{jl} \leq 1, \quad i, k \in I, i \neq k, \; j \in J, \; t \in \{1, \cdots, T\}$$

$$(C5) \quad \sum_{l \in L_i} \sum_{t \in T_i} x_{it}^{jl} = w_i \times z_i^j, \quad i \in I, \; j \in J$$

[2] We consider a discrete time problem formulation with the time granularity representing a tradeoff in the accuracy of the solution and the time required to compute the optimal solution. In the experimental results of Section V, we consider a granularity of 1 second.

[3] For ease of notation, we henceforth drop the subscript i for l_i, as the request indexed is clear from the context.

$$(C6) \quad z_i^j = z_k^j, \qquad i,k \in s_m,\ s_m \in S,\ j \in J$$

$$(C7) \quad x_{it}^{jl} = x_{i(t+1)}^{jl} = \cdots = x_{i(t+w_i-1)}^{jl} = y_i^{jl}, \qquad t = a_i + l,\ l \in L_i,\ j \in J$$

$$(C8) \quad \sum_{j \in J} \sum_{l \in L_i} y_i^{jl} = 1. \qquad i \in I$$

where $N_0 = 0$. The first term in Equation (1) represents the resource overhead due to migration, while the second term represents the resource consumption when servers are in active use. Constraint (C1) is a system constraint that the cluster has sufficient capacity during each measurement interval k to service all scheduled requests. Constraints (C2) and (C3) are QoS (response time) constraints that each request i arriving at a_i should be finished before $a_i + r^*$. Constraint (C4) guarantees that a single server services no more than one request at any time instant t. Constraint (C5) ensures that each request is serviced by a single server. Constraint (C6) is the session-affinity constraint that requests belonging to the same session will be serviced by the same server. Finally, constraints (C7) and (C8) are non-preemption constraints, ensuring that each request is serviced continuously once it begins service.

B. Linear Programming Solution

The first term $(N_k - N_{k-1})^+$ in the objective function hinders us from solving the problem with linear programming method. In this section, we first define a linear programming problem Q, then prove that problems *Q* and *C* have the same solution.

First, we define a linear programming problem *Q* as follows

$$\min\{M \times \sum_{k=1}^{K} \alpha_k + \tau \times \sum_{k=1}^{K} N_k\}, \tag{2}$$

subject to

$$(C9) \quad N_k - N_{k-1} = \alpha_k - \beta_k, \qquad k \in \{1, 2, \cdots, K\}$$

and constraints (C1)-(C8), where $\alpha_k \geq 0$ and $\beta_k \geq 0$.

The following lemmas and theorem show that problems *Q* and *C* are equivalent.

Lemma 1 For any pair (N_k, N_{k-1}) with N_k and N_{k-1} non-negative integers, there exists a sequence (α_{ki}, β_{ki}) with $\alpha_{ki} \geq 0$ and $\beta_{ki} \geq 0$, such that $N_k - N_{k-1} = \alpha_{ki} - \beta_{ki}$ for all i, and $\min_i \alpha_{ki} = [N_k - N_{k-1}]^+$.

Proof: There are three relevant cases for $N_k - N_{k-1}$. If $N_k - N_{k-1} = 0$, then $[N_k - N_{k-1}]^+ = 0$, since $\beta_{ki} \geq 0\ \forall\ i$, $\min_i \alpha_{ki} = 0$. If $N_k - N_{k-1} < 0$, then $[N_k - N_{k-1}]^+ = 0$, since $\beta_{ki} \geq 0\ \forall\ i$, $\min_i \alpha_{ki} = 0$. Finally, if $N_k - N_{k-1} > 0$, then $[N_k - N_{k-1}]^+ = N_k - N_{k-1}$. Thus, since $\beta_{ki} \geq 0\ \forall\ i$, we have that $\min_i \alpha_{ki} = N_k - N_{k-1}$. □

Lemma 2 If α_k^* and N_k^* is the optimal solution of problem Q such that $Q^* = \min Q = \{M \times \sum_{k=1}^{K} \alpha_k^* + \tau \times \sum_{k=1}^{K} N_k^*\}$, with the statement in Lemma 1, then $\alpha_k^* = \min_i \alpha_{ki}^*$ for all k.

Proof: For any pair (N_k^*, N_{k-1}^*), there exists a sequence (α_{ki}^*, β_{ki}^*), $\alpha_{ki}^* \geq 0$, $\beta_{ki}^* \geq 0$, such that $N_k^* - N_{k-1}^* = \alpha_{ki}^* - \beta_{ki}^*$, for all k, i. We need to show that $\alpha_k^* = \min_i \alpha_{ki}^*$, $\forall\ k$ which we prove by contradiction as follows.

Assume that there exists an $l \in [1, 2, \cdots, K]$ such that $\alpha_l^* \neq \min_i \alpha_{li}^*$. Let $\tilde{\alpha_l} = \min_i \alpha_{li}^*$. Then we have $\tilde{\alpha_l} \in \{\alpha_{li}^*\}$, and since $\alpha_l^* \in \{\alpha_{li}^*\}$, we have $\tilde{\alpha_l} < \alpha_l^*$. Replacing α_l^* with $\tilde{\alpha_l}$, we have

$$\tilde{Q} = \{M \times (\sum_{k=1}^{l-1} \alpha_k^* + \tilde{\alpha_l} + \sum_{k=l+1}^{K} \alpha_k^*) + \tau \times \sum_{k=1}^{K} N_k\}.$$

Since $M > 0$, $\tilde{Q} < Q^* = \min Q$, which leads to a contradiction. Thus, $\alpha_l^* = \min_i \alpha_{li}^*$. □

Theorem 1 Optimization problems *Q* and *C* have identical minimum values and identical solutions in terms of x_{it}^{jl*}, y_i^{jl*}, z_i^{j*} and N_k^*.

Proof: From Lemmas 1 and 2, we have $\alpha_k^* = (N_k^* - N_{k-1}^*)^+$ for all k. From Equations (1)-(2), problems *Q* and *C* are equivalent. □

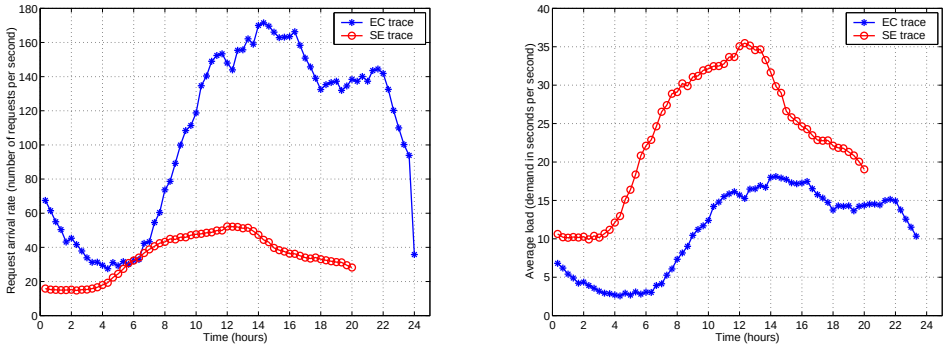
C. Relaxed Optimal

Given the high computational complexity of the above linear programming formulation, we consider the following relaxed optimization problem in order to efficiently compute optimal solutions to the large 24 hour traces described in Section V. Namely, for the experimental investigation of QuID-optimal, we consider system constraint (C1), response time constraints (C2)-(C3) and constraint (C9) while relaxing constraints (C4)-(C8). While significantly reducing the computational complexity, removing these constraints also results in a reduction in the computed minimal number of required servers. For example, by relaxing the session-affinity constraint (C6), requests within a session may be serviced by different CPUs, resulting in fewer required CPUs overall. Hence the reported QuID-optimal experimental results provide a bound on the performance of an on-line algorithm.

In all cases, we use the linear programming package CPLEX [11] to compute the optimal decision sequence.

V. Trace-driven Simulation Experiments

In this section, we use workload traces from two large-scale commercial sites to evaluate the performance of the QuID-



(a) Requests (b) Workload

Fig. 4. Trace Request Arrivals and Workload

Request type	Percentage of requests	Mean demand				
		Request			Response	
		Web	App	DB	App	Web
Static	18.7	10	0	0	0	0
Cache Hit	40.5	10	1	0	0	1
Cache Miss	17.4	10	500	60	1	1
Uncacheable	8.08	10	100	60	1	1
Search	10.2	10	40	0	0	1
Other	5.07	10	20	0	0	1

TABLE I

MEAN CPU DEMANDS FOR DIFFERENT REQUEST TYPES OF THE EC-TRACE. DEMAND VALUES ARE RELATIVE TO THE MEAN CPU DEMAND OF A CACHE-HIT REQUEST ON THE APPLICATION TIER.

online algorithm and to explore the limits of dynamic server migration via the QuID-optimal algorithm. As a further base-line, we also compare with the case of a static system in which the number of servers in the IDC cluster remains fixed for all time. Finally, we explore the impact of trace characteristics such as the workload's peak-to-mean ratio and autocorrelation function on performance by perturbing one of the traces via self aggregation and log transforms. This methodology allows us to study the impact of various trace attributes on the effectiveness of QuID.

A. Methodology

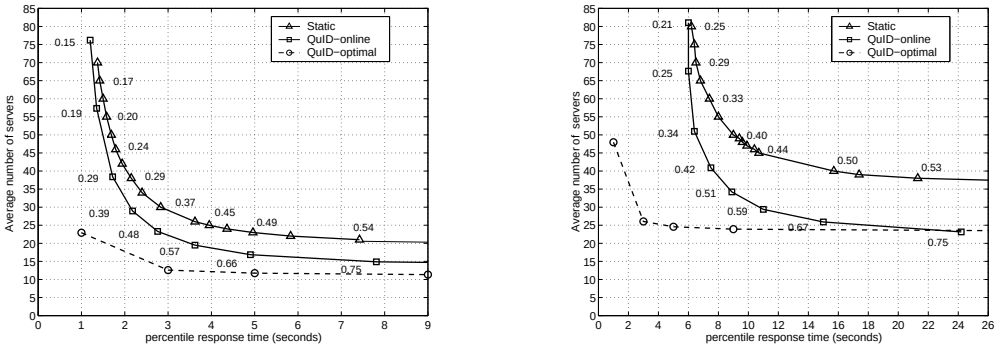
A.1 Traces

We use the two commercial traces depicted in Figure 4 for our evaluation. The "EC trace", studied in [3], is a 24-hour trace from a large-scale E-Commerce site with a multi-tier architecture as illustrated in Figure 1. The "SE trace" is a near 24-hour trace obtained from Google, a large scale Search-Engine website that relies on a cluster of search servers. The SE trace was collected in early 2000 and consists of request IDs and times to service requests (and not actual query information).[4] For the two traces and a 20 minute aggregation granularity, Figure 4(a) depicts the rate of request arrivals and Figure 4(b) depicts the workload which refers to the total CPU demand arriving per second. Note that the SE trace has a smaller request rate but a larger workload than the EC trace due to the larger mean CPU processing time per request for searches as compared to e-commerce requests.

The EC trace was gathered using web server logging mechanisms and is composed of sessions that consist of at least one request per session. Each request has an arrival time, a session-ID, and a classification type according to a static document, cache hit or miss, search, etc. Using testbed measurements, we computed the mean CPU demand (processing time required to serve such a request) for each type of request. The resulting request types, percentage breakdown, and mean CPU demands are summarized in Table 1. To associate a CPU demand (workload) with each request in the trace, we determine the request's classification and generate an exponentially distributed random variable with mean according to Table 1.

The SE trace is based on search engine server logs obtained from a popular search-engine website. The SE trace includes each request's arrival-time and processing time. As all requests are of the same type for the SE trace (search), classification is not required. Thus, we compute the empirical distribution of request processing times (discarding outliers) and generate CPU demands according to this distribution.

[4]Neither trace is available in the public domain and for privacy concerns, both traces have been scaled so that the depicted rates of Figure 4 differ by a constant factor from the actual traces.



(a) EC trace (original) (b) SE trace

Fig. 5. Number of Servers in the Cluster vs. 95%-ile Response Time (100%-ile for QuID-optimal)

A.2 Simulation Preliminaries

We have implemented a trace-driven simulator based on YacSim [14]. The simulator has three key functions as depicted in Figure 2. First, it implements the QuID-online algorithm as described in Section III.[5] Second, it performs round-robin allocation of sessions to servers and ensures session affinity as described in Section II. Third, the simulator models servers according to a first-come-first-serve non-preemptive CPU schedule, with request service times computed as above. Thus, the simulator necessarily ignores many low level details of the server, but by using trace-driven inputs and response time measurements, it allows us to study the performance of server migration algorithms.

For both traces, we consider a single bottleneck tier, and study three factors. First, we explore ρ, the target utilization for server CPUs which controls the request response time. Second, we consider τ, the duration of the measurement interval. As described in Section III, τ is the interval over which measurements are collected by the QuID cluster manager. As the QuID cluster manager re-evaluates N' (the new number of required servers) each time it receives new measurements, τ also determines the system's control time scale. We consider values of τ in the range of 0.5 to 20 minutes. Finally, we explore the impact of the migration time M required for a server to join a cluster. We consider values for M in the range of 1 to 10 minutes.

Finally, as simulator outputs, we record the request response times (elapsed time between request arrival and completion) and the number of servers in the cluster at each interval τ. We report 95%-ile response times as our key QoS measure and the average number of servers in the cluster as a measure of QuID's efficiency.

B. QuID Performance Gains

Here, we explore the resource savings and QoS improvements available due to QuID as compared to a static approach. Figure 5 depicts the average number of servers vs. the response times for the QuID-online algorithm, and presents comparisons with QuID-optimal and static allocation as baselines.[6] Consider first the QuID-online curve of the EC trace in Figure 5(a). Each point on the curve represents the result of a simulation for a different target utilization ρ, with measured utilization values depicted next to each point. For example, the figure shows that for a utilization of 0.66, QuID-online achieves a 95%-ile response time of less than 5 seconds and requires 17 servers on average. In contrast, for a static allocation, 24 servers are required to achieve this same 5 second response time. Hence, for this response-time QoS target, QuID-online has reduced the required number of servers by 29%. An equivalent interpretation is that for a fixed number of servers, QuID-online improves QoS as compared to a static approach. For example, for the EC trace and 20 servers, QuID-online achieves a 95%-ile response time of 3.7 sec vs. 7.4 sec for static allocation, a 50% reduction. Moreover, as IDC operators would likely be required to over-provision servers to address unknown and highly variable workloads, the gains of QuID-online as compared to a static approach may be even more pronounced in practice.

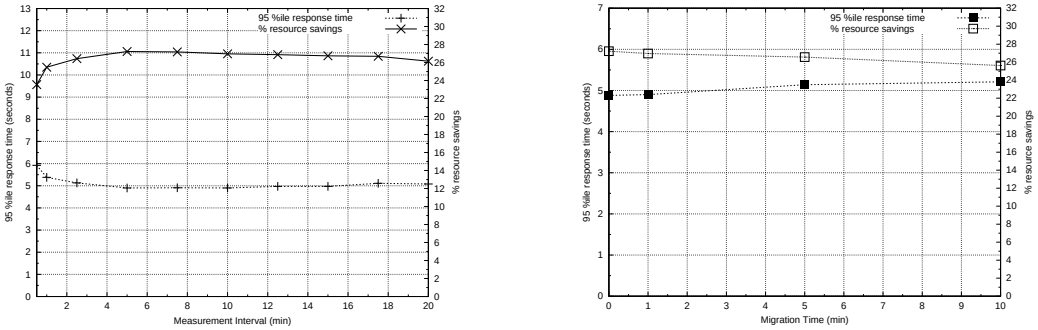
Next, consider the QuID-optimal curve in Figure 5(a). As described in Section IV, this represents the fundamental limits of the efficiency of a dynamic migration policy. For example, the figure shows that for a 100%-ile response time of 5 sec,[7] QuID-optimal utilizes 12 servers as compared to 17 for QuID-online and 24 for static allocation; hence, a further resource savings of 21% is available in theory. Next, notice that with increasing response times, the QuID-optimal curve approaches a limit for the minimum number of servers needed to serve the entire workload without any response-time constraints. This limit is given by $N_{min} = \sum_i w_i/T$ (recall that w_i is the workload of request i and T is 24 hours) which for the EC trace is 11 servers and for the SE trace is 24 servers. Finally, we observe that it would be impossible to realize the complete gains of QuID-optimal in practice. In particular, QuID-optimal is a non-causal algorithm that can (in effect) anticipate future increases in demand and migrate servers to be ready at precisely the right moment. Moreover, QuID-optimal ignores session affinity, optimally schedules CPU time, divides the load of individual requests across multiple servers, etc. (Section IV details the complete scenario for QuID-optimal.) Regardless, QuID-optimal provides a lower bound on the number of servers needed to support the workloads.

Next observe that for the SE trace of Figure 5(b), the general

[5] The initial number of servers is 16.

[6] For the figure, τ = 5 minutes and M = 1 minute.

[7] It is not computationally feasible to compute the 95%-ile for QuID-optimal. Thus, the figure shows the 95%-iles for QuID-online and static, and the 100%-ile for QuiD-optimal.



(a) Measurement Interval τ (b) Migration Time M

Fig. 6. Impact of τ and M on QoS and Resource Savings

trends are the same whereas for higher response times, the resource savings for QuID-online are larger as compared to static and QuID-online performs more closely to QuID-optimal. However, as with Figure 5(a), with very low utilizations and response times, the static and QuID-online curves converge to an asymptote quite different than QuID-optimal. This situation is explained as follows. When the server utilizations are very low, the 95%-ile of the response times are approaching the 95%-ile of the CPU demand times such that each server is normally serving a single session at a time. In such a case, there is no advantage of a dynamic migration policy over a static one. However, QuID-optimal still shows a resource savings in this regime as it divides requests among multiple servers thereby reducing response times for a given number of servers.

C. Control Time Scale and Migration time

In this set of experiments, we fix ρ and study the effects of the migration time M and the measurement interval τ on the performance of QuID-online as compared to static allocation.

Figure 6(a) depicts the results for experiments with $\rho = 0.7$ and $M = 1$ minute, and τ varied in the range of 0.5 to 20 minutes. In the algorithm, τ functions as both a measurement interval and the control-time-scale of migration. Thus, if τ is too small, the algorithm is ineffectively attempting to track fast-time-scale demand fluctuations. Similarly, if τ is too large, the algorithm is not sufficiently responsive to demand changes. The figure illustrates these general trends but indicates that most of the performance gains are obtained across a wide range of τ, with the best value being 5 minutes.

For the experiments of Figure 6(b), $\rho = 0.7$ and $\tau = 10$ minutes and the migration time M is varied in the range of 0 to 10 minutes. The figure illustrates that migration time has a relatively minor impact on the performance of QuID-online, as all of the considered migration times are significantly less than the time-scales of the trace's changes in workload demand. (See also the autocorrelation function of the original trace in Figure 7(b).)

D. Trace Characteristics

Here we study the performance of the QuID-online algorithm as a function of the peak-to-mean ratio and autocorrelation function of the trace. To explore this issue, we generate a modified set of traces via a set of log transformations (to modify peak-to-mean workload ratio) and temporal aggregations (to modify autocorrelation).

D.1 Methodology for Trace Transformation

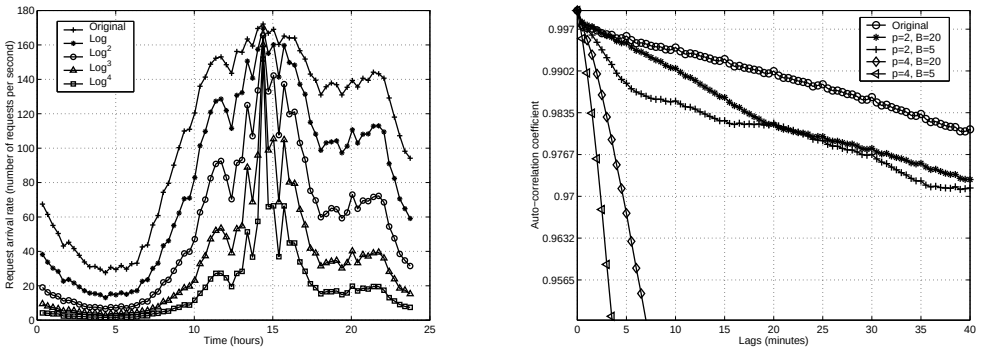
Given the limited number of available traces suitable for IDC simulations, we transform the trace by taking the log of (one minus) the scaled number of arrivals and then again rescaling the modified trace. By repeating this log transformation p times, increasingly higher peak-to-mean ratios are obtained. The effect of this transformation is illustrated in Figure 7(a). Also evident from Figure 7(b), the log transformation modifies the trace's autocorrelation function. To control the autocorrelation function directly, we also self-aggregate the trace with different aggregation "batch" intervals denoted by B. In particular, if the original trace is $X_1, X_2, \cdots$, a trace aggregated with a batch interval B has $Y_i = (X_i + X_{i+1} + \cdots X_{i+B-1})/B$. Thus, a larger value of B results in greater temporal correlation. The net effects of these two transformations are illustrated in Figure 7(a,b).

D.2 Experiments with Transformed Traces

Figure 8 depicts the results of the experiments with the transformed traces. In these experiments, we consider $\rho = .7$, $\tau = 5$, and $M = 1$, and each of the 4 points in the curve represents results of the $\log^1$ to $\log^4$ traces with $B = 5$ and $B = 20$. Figure 8(a) shows the resource savings of QuID-online as compared to static allocation for the same 95%-ile response time. Similarly, Figure 9 depicts the response time as a function of the modified traces' autocorrelation coefficient at a lag of 5 minutes.

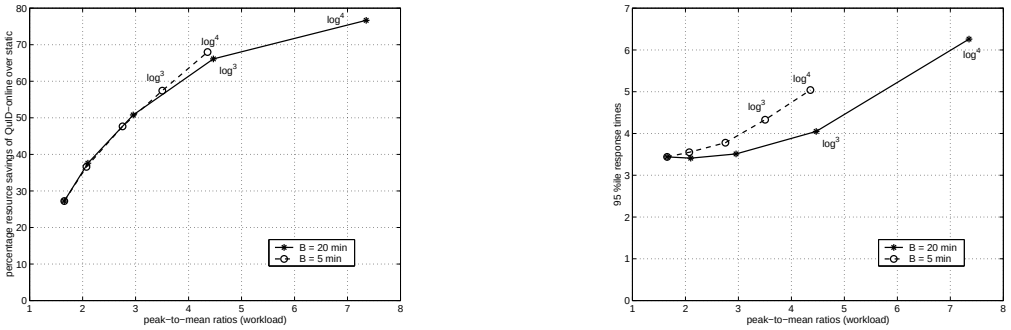
We make the following observations about the figures. First, note from Figure 8 that with an increasing peak-to-mean ratio, QuID-online is able to extract increased resource savings. However, the relationship is not linear as the more widely varying traces are also more difficult for the dynamic algorithm to track due to their more rapidly decaying autocorrelation functions. For example, the demand surge of the $\log^4$ trace results in periods where an insufficient number of servers are allocated such that response times increase (Figure 8(b)).

In summary, while the above log and aggregation transformations are not intended to produce realistic traces, it results in an important conclusion: if IDCs experience "flash crowds" or demand surges as represented by the $\log^4$ trace, QuID-online can track the widely varying load provided that the autocorrelation values are still relatively high (e.g., above 0.9) at lags corre-



(a) Request Arrival Rate (b) Autocorrelation Function

Fig. 7. Original and Transformed EC Traces



(a) Resource Savings (b) Response Time

Fig. 8. Impact of Peak-to-Mean

sponding to the key time scales of the system, namely the server migration time and the measurement window (e.g., 1 to 10 minutes).

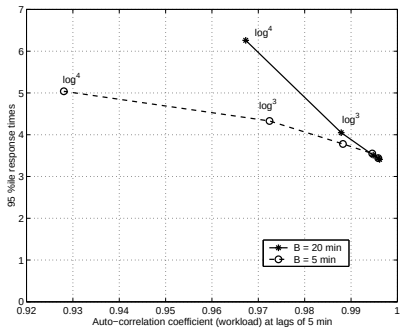


Fig. 9. Impact of Autocorrelation

VI. Related Work

As described in the Introduction and illustrated in Section V, static partitioning of IDC resources results in under-utilization of resources. A number of related approaches have been proposed to address this issue [1], [9], [12], [18], [19]. Each offers a notion of utility computing where resources can be acquired and released when/where they are needed. Such architectures can be classified as employing *shared server utility* or *full server utility* models. With the shared server utility model, many services share a server at the same time, whereas with the full server utility model, each server offers one service at a time.

An example of a shared server utility environment is MUSE [9] in which hosted web sites are treated as services. All services are run concurrently on all servers in a cluster, and a pricing/optimization model is used to determine the fraction of CPU resources allocated to each service on each server. A special dispatcher, a level 4 load balancer, routes web page requests for all services only to those servers where the service has a CPU allocation. The optimization model shifts load to use as few servers as possible while satisfying service level agreements (SLA) for the services. A major goal of the work is to maximize the number of unused servers so that they can be powered down to reduce energy consumption.

Other mechanisms for shared server utilities include cluster reserves for multiple-service performance isolation [4] and QoS-differentiation algorithms [20]. Examples of commercial shared server utility computing products that partition server resources in clusters of shared servers include [12], [19], which have resource management systems that aim to satisfy SLAs while better supporting increased resource utilization.

In contrast, a *full server utility* data center is presented in reference [18] and has been realized as a product [13]. It exploits the use of virtual LANs and SANs for partitioning of resources into secure domains called virtual application environments. These environments support multi-tier as well as single-tier applications as in Figure 1 and can also contain systems that internally implement a shared server utility model. In either case, such a system can make use of our QuID framework as a resource migration algorithm.

A second example of a full server utility approach is Oceano

[1], an architecture for e-business applications. The security domains are dynamic as the resources assigned to them may be augmented when load increases and reduced otherwise. Reference [1] presents an overview of an SLA based management system for the multi-tier environment that includes service levels for response time metrics gathered within each domain. Service level violations trigger the addition and removal of servers from clusters. The focus of [1] is the management architecture and measurements of key actions such as server migration times. However, in both [1], [18] migration algorithms are not evaluated and no studies of the performance of dynamic migration policies are presented.

Thus, unlike shared server utility approaches, QuID applies to full server utility environments such as [1], [18]. In particular, we have devised a QuID cluster manager control algorithm and demonstrated its effectiveness in reducing the number of server resources needed by hosted applications while satisfying application response time objectives.

Finally, we note that the issue of web server QoS has received a great deal of attention in contexts such as web server admission control [7], [15], [17], QoS-based server selection among servers located at different network nodes [10], operating system support [6], [8], networking support, etc. Such techniques represent mechanisms at the request and session time scale whereas QuID operates on time scales of minutes. Moreover, the granularity of resource control for QuID is the server, whereas such techniques typically have a shared server utility model.

VII. Conclusions

This paper presents QuID, a framework for QoS-driven server migration in Internet Data Centers. We found via trace-driven simulation that the QuID-online algorithm provides resource savings as compared to static allocation in the range of 25% for E-Commerce and Search Engine traces with peak-to-mean workloads near 1.5, and savings near 68% for workloads with peak-to-mean ratios near 5. This reduction is in comparison with the smallest number of servers that can be statically provisioned to achieve similar response time percentiles. In general, a static allocation would have to over-provision additional servers since it is unlikely that the workload's peak demand would be known in advance.

The results of the technique also compare favorably with an off-line algorithm that gives an upper bound on resource savings for a specific workload trace and a maximum response delay requirement. Potential resource gains appear relatively insensitive to reasonable values for measurement interval and server migration time.

We explored the sensitivity of the technique with respect to trace characteristics by perturbing one of the traces. As the peak-to-mean increases greater resources savings are achieved. However, as a side effect of our method, an increase in peak-to-mean also causes decreases in autocorrelation of the system load. The decreases in auto-correlation did not appear to have as significant an impact on resource savings but do decrease QoS by increasing response time percentiles. In all cases our system load autocorrelation values were above 0.9 for the time scales of interest.

Finally, in future work, we plan to consider the interactions of server tiers (i.e., clusters) within an application as well as multiple applications within an IDC. Moreover, we plan to explore overload policies, i.e., how to migrate servers in scenarios when the entire resource pool is exhausted. Lastly, we plan to implement the algorithm in our large-scale server-migration testbed to provide a proof-of-concept demonstration of QuID.

VIII. Acknowledgements

The authors thank Martin Arlitt (HP Labs) and Jeff Dean (Google) for their help in procuring the EC and SE traces respectively. The authors also thank Li Deng, Richard Tapia, Cong Teng (Rice University), and Xiaoyun Zhu (HP Labs), for their suggestions and comments on the QuID-optimal algorithm.

References

[1] K. Appleby et al. Oceano – SLA based management of a computing utility. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, May 2001.

[2] M. Arlitt and T. Jin. Workload Characterization of the 1998 World Cup Web Site. Technical Report HPL-1999-35R1, HP Laboratories, September, 1999. Trace available at http://ita.ee.lbl.gov.

[3] M. Arlitt, D. Krishnamurthy, and J. Rolia. Characterizing the scalability of a large web-based shopping system. *ACM Transactions on Internet Technology*, 1(1):44–69, August 2001.

[4] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Proceedings of ACM SIGMETRICS 2000*, June 2000.

[5] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *Proceedings of the USENIX 2000 Annual Technical Conference*, June 2000.

[6] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation*, February 1999.

[7] N. Bhatti and R. Friedrich. Web Server Support for Tiered Services. *IEEE Network*, 13(5):64–71, September 1999.

[8] J. Bruno, E. Gabber, B. Ozden, and A. Silberschatz. The eclipse operating system: Providing quality of service via reservation domains. In *Proceedings of the 1998 USENIX Annual Technical Conference*, New Orleans, Louisiana, 1998.

[9] J. Chase et al. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[10] J. Chuang and M. Sirbu. Distributed network storage with quality-of-service guarantees. *Journal of Network and Computer Applications*, 23(3):163–185, July 2000.

[11] CPLEX. Simplex optimizer for linear programming problems. http://www.ilog.com/products/cplex/.

[12] Ejasent. Utility computing white paper, November 2001. http://www.ejasent.com.

[13] Hewlett-Packard. HP utility data center architecture. http://www.hp.com/solutions1/infrastructure/solutions /utilitydata/architecture/index.html.

[14] R. Jump. Yacsim reference manual version 2.1. ECE Department, Rice University.

[15] V. Kanodia and E. Knightly. Multi-class latency bounded web services. In *Proceedings of the 8th International Workshop on Quality of Service*, Pittsburg, PA, June 2000.

[16] L. Kleinrock. *"Queueing Systems, Volume II: Computer Applications"*. Wiley, 1976.

[17] K. Li and S. Jamin. A measurement-based admission controlled web server. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[18] J. Rolia, S. Singhal, and R. Friedrich. Adaptive Internet Data Centers. In *SSGRR'00*, L'Aquila, Italy, July 2000.

[19] Sychron. Sychron Enterprise Manager, 2001. http://www.sychron.com.

[20] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation for cluster-based network servers. In *Proceedings of IEEE INFOCOM'2001*, Anchorage, AK, April 2001.