Dynamic Resource Allocation for Shared Data Centers Using Online Measurements *

Abhishek Chandra[1], Weibo Gong[2], and Prashant Shenoy[1]

[1] Department of Computer Science
University of Massachusetts Amherst
{abhishek,shenoy}@cs.umass.edu
[2] Department of Electrical and Computer Engineering
University of Massachusetts Amherst
gong@ecs.umass.edu

Abstract. Since web workloads are known to vary dynamically with time, in this paper, we argue that dynamic resource allocation techniques are necessary to provide guarantees to web applications running on shared data centers. To address this issue, we use a system architecture that combines online measurements with prediction and resource allocation techniques. To capture the transient behavior of the application workloads, we model a server resource using a time-domain description of a generalized processor sharing (GPS) server. This model relates application resource requirements to their dynamically changing workload characteristics. The parameters of this model are continuously updated using an online monitoring and prediction framework. This framework uses time series analysis techniques to predict expected workload parameters from measured system metrics. We then employ a constrained non-linear optimization technique to dynamically allocate the server resources based on the estimated application requirements. The main advantage of our techniques is that they capture the transient behavior of applications while incorporating nonlinearity in the system model. We evaluate our techniques using simulations with synthetic as well as real-world web workloads. Our results show that these techniques can judiciously allocate system resources, especially under transient overload conditions.

1 Introduction

1.1 Motivation

The growing popularity of the World Wide Web has led to the advent of Internet data centers that host third-party web applications and services. A typical web application consists of a front-end web server that services HTTP requests, a Java application server that contains the application logic, and a backend database server. In many cases, such applications are housed on managed data centers where the application owner pays for (rents) server resources, and in return, the application is provided guarantees on resource availability and performance. To provide such guarantees, the data center—typically a cluster of servers—must provision sufficient resources to meet application

* This research was supported in part by NSF grants CCR-9984030 and EIA-0080119.

needs. Such provisioning can be based either on a dedicated or a shared model. In the dedicated model, some number of cluster nodes are dedicated to each application and the provisioning technique must determine how many nodes to allocate to the application. In the shared model, which we consider in this paper, an application can share node resources with other applications and the provisioning technique needs to determine how to partition resources on each node among competing applications.[3]

Since node resources are shared, providing guarantees to applications in the shared data center model is more complex. Typically such guarantees are provided by reserving a certain fraction of node resources (CPU, network, disk) for each application. The fraction of the resources allocated to each application depends on the expected workload and the QoS requirements of the application. The workload of web applications is known to vary dynamically over multiple time scales [14] and it is challenging to estimate such workloads a priori (since the workload can be influenced by unanticipated external events—such as a breaking news story—that can cause a surge in the number of requests accessing a web site). Consequently, static allocation of resources to applications is problematic—while over-provisioning resources based on worst case workload estimates can result in potential underutilization of resources, under-provisioning resources can result in violation of guarantees. An alternate approach is to allocate resources to applications dynamically based on the variations in their workloads. In this approach, each application is given a certain minimum share based on coarse-grain estimates of its resource needs; the remaining server capacity is dynamically shared among various applications based on their instantaneous needs. To illustrate, consider two applications that share a server and are allocated 30% of the server resources each; the remaining 40% is then dynamically shared at run-time so as to meet the guarantees provided to each application. Such dynamic resource sharing can yield potential multiplexing gains, while allowing the system to react to unanticipated increases in application load and thereby meet QoS guarantees. Dynamic resource allocation techniques that can handle changing application workloads in shared data centers is the focus of this paper.

1.2 Research Contributions

In this paper, we present techniques for dynamic resource allocation in shared web servers. We model various server resources using *generalized processor sharing (GPS)* [29] and assume that each application is allocated a certain fraction of a resource. Using a combination of online measurement, prediction and adaptation, our techniques can dynamically determine the resource share of each application based on (i) its QoS (response time) needs and (ii) the observed workload. The main goal of our techniques is to react to transient system overloads by incorporating online system measurements.

We make three specific contributions in this paper. First, in order to capture the transient behavior of application workloads, we model the server resource using a *time-domain queuing model*. This model dynamically relates the resource requirements of each application to its workload characteristics. The advantage of this model is that it

[3] This requirement is true even in a dedicated model where service differentiation between different customers for the same application may be desirable.

does not make steady-state assumptions about the system (unlike some previous approaches [10, 24]) and adapts to changing application behavior. To achieve a feasible resource allocation even in the presence of transient overloads, we employ a *non-linear optimization* technique that employs the proposed queuing model. An important feature of our optimization-based approach is that it can handle non-linearity in system behavior unlike some approaches that assume linearity [1, 25].

Determining resource shares of applications using such an online approach is crucially dependent on an accurate estimation of the application workload characteristics. A second contribution of our work is a *prediction algorithm* that estimates the workload parameters of applications in the near future using online measurements. Our prediction algorithm uses time series analysis techniques for workload estimation.

Third, we use both synthetic workloads and real-world web traces to evaluate the effectiveness of our online prediction and allocation techniques. Our evaluation shows that our techniques adapt to changing workloads fairly effectively, especially under transient overload conditions.

The rest of the paper is structured as follows. We formulate the problem of dynamic resource allocation in shared web servers in Section 2. In Section 3, we present a time-domain description of a resource queuing model, and describe our online prediction and optimization-based techniques for dynamic resource allocation. Results from our experimental evaluation are presented in Section 4. We discuss related work in Section 5 and present our conclusions and future work in Section 6.

2 Problem Formulation and System Model

In this section, we first present an abstract GPS-based model for a server resource and then formulate the problem of dynamic resource allocation in such a GPS-based system.

2.1 Resource Model

We model a server resource using a system of n queues, where each queue corresponds to a particular application (or a class of applications) running on the server. Requests within each queue are assumed to be served in FIFO order and the resource capacity C is shared among the queues using GPS. To do so, each queue is assigned a weight and is allocated a resource share in proportion to its weight. Specifically, a queue with a weight w_i is allocated a share $\phi_i = \frac{w_i}{\sum_j w_j}$ (i.e., allocated $(\phi_i \cdot C)$ units of the resource capacity when all queues are backlogged). Several practical instantiations of GPS exist—such as weighted fair queuing (WFQ) [15], self-clocked fair queuing [18], and start-time fair queuing [19]—and any such scheduling algorithm suffices for our purpose. We note that these GPS schedulers are work-conserving—in the event a queue does not utilize its allocated share, the unused capacity is allocated fairly among backlogged queues. Our abstract model is applicable to many hardware and software resources found on a server; hardware resources include the network interface bandwidth, the CPU and in some cases, the disk bandwidth, while software resource include socket accept queues in a web server servicing multiple virtual domains [25, 30].

2.2 Problem Definition

Consider a shared server that runs multiple third-party applications. Each such application is assumed to specify a desired quality of service (QoS) requirement; here we assume that the QoS requirements are specified in terms of a target response time. The goal of the system is to ensure that the mean response time (or some percentile of the response time) seen by application requests is no greater than the desired target response. In general, each incoming request is serviced by multiple hardware and software resources on the server, such as the CPU, NIC, disk, etc. We assume that the specified target response time is split up into multiple resource-specific response times, one for each such resource. Thus, if each request spends no more than the allocated target on each resource, then the overall target response time for the server will be met.[4]

Since each resource is assumed to be scheduled using GPS, the target response time of each application can be met by allocating a certain share to each application. The resource share of an application will depend not only on the target response time but also on the load in each application. As the workload of an application varies dynamically, so will its resource share. In particular, we assume that each application is allocated a certain minimum share ϕ_i^{min} of the resource capacity; the remaining capacity $(1 - \sum_j \phi_j^{min})$ is dynamically allocated to various applications depending on their current workloads (such that their target response time will be met). Formally, if d_i denotes the target response time of application i and $\bar{T}_i$ is its observed mean response time, then the application should be allocated a share ϕ_i, $\phi_i \geq \phi_i^{min}$, such that $\bar{T}_i \leq d_i$.

Since each resource has a finite capacity and the application workloads can exceed capacity during periods of heavy transient overloads, the above goal can not always be met. To achieve feasible allocation during overload scenarios, we use the notion of utility functions to represent the satisfaction of an application based on its current allocation. While different kinds of utility functions can be employed, we define utility in the following manner.[5] We assume that an application remains satisfied so long as its allocation ϕ_i yields a mean response time $\bar{T}_i$ no greater than the target d_i (i.e., $\bar{T}_i \leq d_i$). But the discontent of an application grows as its response time deviates from the target d_i. This discontent function can be represented as follows:

$$D_i(\bar{T}_i) = (\bar{T}_i - d_i)^+, \tag{1}$$

where x^+ represents $\max(0, x)$. In this scenario, the discontent grows linearly when the observed response time exceeds the specified target d_i. The overall system goal then is to assign a share ϕ_i to each application, $\phi_i \geq \phi_i^{min}$, such that the total system-wide discontent, i.e., the quantity $D = \sum_{i=1}^{n} D_i(\bar{T}_i)$ is minimized.

We use this problem definition to derive our dynamic resource allocation mechanism, which is described next.

---

[4] The problem of how to split the specified server response time into resource-specific response times is beyond the scope of this paper. In this paper, we assume that such resource-specific target response times are given to us.

[5] Different kinds of utility functions can be employed to achieve different goals during overload, such as fairness, isolation, etc.

3 Dynamic Resource Allocation

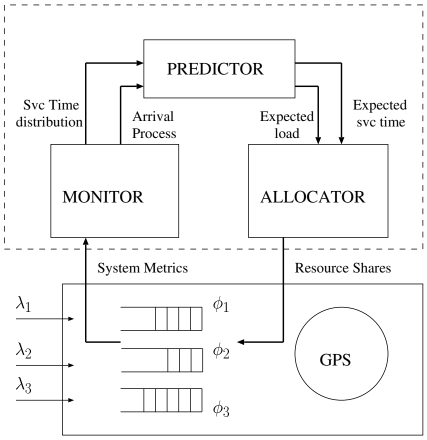


Fig. 1. Dynamic Resource Allocation

To perform dynamic resource allocation based on the above formulation, each GPS-scheduled resource on the shared server will need to employ three components: (i) a *monitoring* module that measures the workload and the performance metrics of each application (such as its request arrival rate, average response time $\bar{T}_i$, etc.), (ii) a *prediction* module that uses the measurements from the monitoring module to estimate the workload characteristics in the near future, and (iii) an *allocation* module that uses these workload estimates to determine resource shares such that the overall system-wide discontent is minimized. Figure 1 depicts these three components.

In what follows, we first present an overview of the monitoring module that is responsible for performing online measurements. We follow this with a time-domain description of the resource queuing model, and formulation of a non-linear optimization problem to perform resource allocation using this model. Finally, we present the prediction techniques used to estimate the parameters for this model dynamically.

3.1 Online Monitoring and Measurement

The online monitoring module is responsible for measuring various system and application metrics. These metrics are used to estimate the system model parameters and workload characteristics. These measurements are based on the following time intervals (see Figure 2):

– *Measurement interval (I):* I is the interval over which various parameters of interest are sampled. For instance, the monitoring module tracks the number of request arrivals (n_i) in each interval I and records this value.

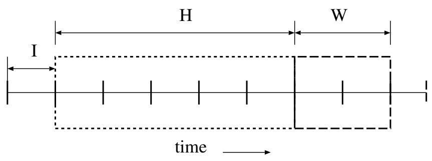


Fig. 2. Time intervals used for monitoring, prediction and allocation

The choice of a particular measurement interval depends on the desired responsiveness from the system. If the system needs to react to workload changes on a fine time-scale, then a small value of I (e.g., $I = 1$ second) should be chosen. On the other hand, if the system needs to adapt to long term variations in the workload over time scales of hours or days, then a coarse-grain measurement interval of minutes or tens of minutes may be chosen.

- *History (H):* The history represents a sequence of recorded values for each parameter of interest. Our monitoring module maintains a finite history consisting of the most recent H values for each such parameter; these measurements form the basis for predicting the future values of these parameters.
- *Adaptation Window (W):* The adaptation window is the time interval between two successive invocations of the adaptation algorithm. Thus the past measurements are used to predict the workload for the next W time units, and the system adapts over this time interval. As we would see in the next section, our time-domain queuing model description considers a time period equal to the adaptation window to estimate the average response time $\bar{T}_i$ of an application, and this model is updated every W time units.

The history and the adaptation window are implemented as sliding windows.

3.2 Allocating Resource Shares to Applications

The allocation module is invoked periodically (every adaptation window) to dynamically partition the resource capacity among the various applications running on the shared server. To capture the transient behavior of application workloads, we first present a time-domain description of a resource queuing model. This model is used to determine the resource requirements of an application based on its expected workload and response time goal.

Time-domain Queuing Model As described above, the adaptation algorithm is invoked every W time units. Let q_i^0 denote the queue length at the beginning of an adaptation window. Let $\hat{\lambda}_i$ denote the estimated request arrival rate and $\hat{\mu}_i$ denote the estimated service rate in the next adaptation window (i.e., over the next W time units). We would show later how these values are estimated. Then, assuming the values of $\hat{\lambda}_i$ and $\hat{\mu}_i$ are constant, the length of the queue at any instant t within the next adaptation window is given by

$$q_i(t) = \left[q_i^0 + \left(\hat{\lambda}_i - \hat{\mu}_i\right) \cdot t\right]^+, \tag{2}$$

Intuitively, the amount of work queued up at instant t is the sum of the initial queue length and the amount of work arriving in this interval minus the amount of work serviced in this duration. Further, the queue length cannot be negative.

Since the resource is modeled as a GPS server, the service rate of an application is effectively $(\phi_i \cdot C)$, where ϕ_i is the resource share of the application and C is the resource capacity, and this rate is continuously available to a backlogged application in any GPS system. Hence, the request service rate is

$$\hat{\mu}_i = \frac{\phi_i \cdot C}{\hat{s}_i}, \tag{3}$$

where $\hat{s}_i$ is the estimated mean service demand per request (such as number of bytes per packet, or CPU cycles per CPU request, etc.).

Note that, due to the work conserving nature of GPS, if some applications do not utilize their allocated shares, then the utilized capacity is fairly redistributed among backlogged applications. Consequently, the queue length computed in Equation 2 assumes a worst-case scenario where all applications are backlogged and each application receives no more than its allocated share (the queue would be smaller if the application received additional unutilized share from other applications).

Given Equation 2, the average queue length over the adaptation window is given by:

$$\bar{q}_i = \frac{1}{W} \int_0^W q_i(t)dt \tag{4}$$

Depending on the particular values of q_i^0, the arrival rate $\hat{\lambda}_i$ and the service rate $\hat{\mu}_i$, the queue may become empty one or more times during an adaptation window. To include only the non-empty periods of the queue when computing $\bar{q}_i$, we consider the following scenarios, based on the assumption of constant $\hat{\mu}_i$ and $\hat{\lambda}_i$:

1. *Queue growth:* If $\hat{\mu}_i < \hat{\lambda}_i$, then the application queue will grow during the adaptation window and the queue will remain non-empty throughout the adaptation window.
2. *Queue depletion:* If $\hat{\mu}_i > \hat{\lambda}_i$, then the queue starts depleting during the adaptation window. The instant t_0 at which the queue becomes empty is given by $t_0 = \frac{q_i^0}{\hat{\mu}_i - \hat{\lambda}_i}$. If $t_0 < W$, then the queue becomes empty within the adaptation window, otherwise the queue continues to deplete but remains non-empty throughout the window (and is projected to become empty in a subsequent window).
3. *Constant queue length:* If $\hat{\mu}_i = \hat{\lambda}_i$, then the queue length remains fixed $(= q_i^0)$ throughout the adaptation window. Hence, the non-empty queue period is either 0 or W depending on the value of q_i^0.

Let us denote the duration within the adaptation window for which the queue is non-empty by W_i (W_i equals either W or t_0 depending on the various scenarios). Then, Equation 4 can be rewritten as

$$\bar{q_i} = \frac{1}{W}\int_0^{W_i} q_i(t)dt \tag{5}$$

$$= \left(\frac{W_i}{W}\right)\left[q_i^0 + \frac{W_i}{2}\left(\hat{\lambda}_i - \hat{\mu}_i\right)\right] \tag{6}$$

Having determined the average queue length over the next adaptation interval, we derive the average response time $\bar{T_i}$ over the same interval. Here, we are interested in the average response time in the near future. Other metrics such as a long term average response time could also be considered. $\bar{T_i}$ is estimated as the sum of the mean queuing delay and the request service time over the next adaptation interval. We use Little's law to derive the queuing delay from the mean queue length.[6] Thus,

$$\bar{T_i} = \frac{(\bar{q_i} + 1)}{\hat{\mu}_i} \tag{7}$$

Substituting Equation 3 in this expression, we get

$$\bar{T_i} = \left(\frac{\hat{s}_i}{\phi_i \cdot C}\right) \cdot (\bar{q_i} + 1), \tag{8}$$

where $\bar{q_i}$ is given by equation 6. The values of q_i^0, $\hat{\mu}_i$, $\hat{\lambda}_i$ and $\hat{s}_i$ are obtained using measurement and prediction techniques discussed in the next section.

This time-domain model description has the following salient features:

- The parameters of the model depend on its current workload characteristics ($\hat{\lambda}_i$, $\hat{s}_i$) and the current system state (q_i^0). Consequently, this model is applicable in an *online* setting for reacting to dynamic changes in the workload, and does not make any steady state assumptions.
- As shown in Equation 8, the model assumes a non-linear relationship between the response time $\bar{T_i}$ and the resource share ϕ_i. This assumption is more general than linear system assumption made in some scenarios.

Next we describe how this model is used in dynamic resource allocation.

Optimization-based Resource Allocation As explained earlier, the share allocated to an application depends on its specified target response time and the estimated workload. We now present an online optimization-based approach to determine resource shares dynamically.

As described in section 2, the allocation module needs to determine the resource share ϕ_i for each application such that the total *discontent* $D = \sum_{i=1}^{n} D_i(\bar{T_i})$ is minimized. This problem translates to the following constrained optimization problem:

$$\min_{\{\phi_i\}} \sum_{i=1}^{n} D_i(\bar{T_i})$$

[6] Note that the application of Little's Law in this scenario is an approximation, that is more accurate when the size of the adaptation window is large compared to the average request service time.

subject to the constraints

$$\sum_{i=1}^{n} \phi_i \leq 1,$$
$$\phi_i^{min} \leq \phi_i \leq 1, \ 1 \leq i \leq n.$$

Here, D_i is a function that represents the discontent of a class based on its current response time $\bar{T}_i$. The two constraints specify that (i) the total allocation across all applications should not exceed the resource capacity, and (ii) the share of each application can be no smaller than its minimum allocation ϕ_i^{min} and no greater than the resource capacity.

In general, the nature of the discontent function D_i has an impact on the allocations ϕ_i for each application. As shown in Equation 1, a simple discontent function is one where the discontent grows linearly as the response time $\bar{T}_i$ exceeds the target d_i. Such a D_i, shown in Figure 3, however, is non-differentiable. To make our constrained optimization problem mathematically tractable, we approximate this piece-wise linear D_i by a continuously differentiable function:

$$D_i(\bar{T}_i) = \frac{1}{2}[(\bar{T}_i - d_i) + \sqrt{(\bar{T}_i - d_i)^2 + k}],$$

where $k > 0$ is a constant. Essentially, the above function is a hyperbola with the two piece-wise linear portions as its asymptotes and the constant k governs how closely this hyperbola approximates the piece-wise linear function. Figure 3 depicts the nature of the above function.

We note that the optimization is with respect to the resource shares $\{\phi_i\}$, while the discontent function is represented in terms of the response times $\{\bar{T}_i\}$. We use the relation between $\bar{T}_i$ and ϕ_i from Equation 8 to obtain the discontent function in terms of the resource shares $\{\phi_i\}$.

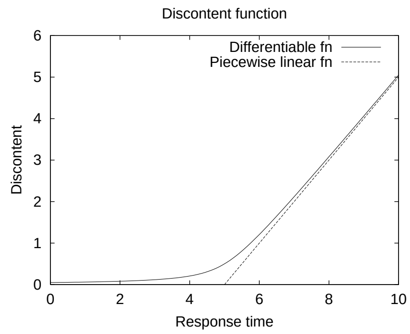


Fig. 3. Two different variants of the discontent function. A piecewise linear function and a continuously differentiable convex functions are shown. The target response time is assumed to be $d_i = 5$.

The resulting optimization problem can be solved using the Lagrange multiplier method [9]. In this technique, the constrained optimization problem is transformed into an unconstrained optimization problem where the original discontent function is replaced by the objective function:

$$L(\{\phi_i\}, \beta) = \sum_{i=1}^{n} D_i(\bar{T}_i) - \beta \cdot (\sum_{i=1}^{n} \phi_i - 1). \tag{9}$$

The objective function L is then minimized subject to the bound constraints on ϕ_i. Here β is called the Lagrange multiplier and it denotes the shadow price for the resource. Intuitively, each application is charged a price of β per unit resource it uses. Thus, each application attempts to minimize the price it pays for its resource share, while maximizing the utility it derives from that share. This leads to the minimization of the original discontent function subject to the satisfaction of the resource constraint.

Minimization of the objective function L in the Lagrange multiplier method leads to solving the following system of algebraic equations.

$$\frac{\partial D_i}{\partial \phi_i} = \beta, \quad \forall i = 1, \dots, n \tag{10}$$

and

$$\frac{\partial L}{\partial \beta} = 0 \tag{11}$$

Equation 10 determines the optimal solution, as it corresponds to the equilibrium point where all applications have the same value of diminishing returns (or β). Equation 11 satisfies the resource constraint.

The solution to this system of equations, derived either using analytical or numerical methods, yields the shares ϕ_i that should be allocated to each application to minimize the system-wide discontent. We use a numerical method for solving these equations to account for the non-differentiable factor present in the time-domain queuing model (Equation 2).

Having described the monitoring and allocation modules, we now describe the prediction module that uses the measured system metrics to estimate the workload parameters that are used by the optimization-based allocation technique.

3.3 Workload Prediction Techniques

The online optimization-based allocation technique described in the previous section is crucially dependent on an accurate estimation of the workload likely to appear in each application class. In this section, we present techniques that use past observations to estimate the future workload for an application.

The workload seen by an application can be characterized by two complementary distributions: the *request arrival process* and the *service demand distribution*. Together these distributions enable us to capture the workload intensity and its variability. Our technique measures the various parameters governing these distributions over a certain time period and uses these measurements to predict the workload for the next adaptation window.

Estimating the Arrival Rate The request arrival process corresponds to the workload intensity for an application. The crucial parameter of interest that characterizes the arrival process is the request arrival rate λ_i. An accurate estimate of λ_i allows the time-domain queuing model to estimate the average queue length for the next adaptation window.

To estimate λ_i, the monitoring module measures the number of request arrivals a_i in each measurement interval I. The sequence of these values $\{a_i^m\}$ forms a time series. Using this time series to represent a stochastic process A_i, our prediction module attempts to predict the number of arrivals $\hat{n}_i$ for the next adaptation window. The arrival rate for the window, $\hat{\lambda}_i$ is then approximated as $\left(\dfrac{\hat{n}_i}{W}\right)$ where W is the window length. We represent A_i at any time by the sequence $\{a_i^1, \ldots, a_i^H\}$ of values from the measurement history.

To predict $\hat{n}_i$, we model the process as an AR(1) process [7] (*autoregressive* of order 1). This is a simple linear regression model in which a sample value is predicted based on the previous sample value

Using the AR(1) model, a sample value of A_i is estimated as

$$\hat{a}_i^{j+1} = \bar{a}_i + \rho_i(1) \cdot (a_i^j - \bar{a}_i) + e_i^j, \tag{12}$$

where, ρ_i and $\bar{a}_i$ are the autocorrelation function and mean of A_i respectively, and e_i^j is a white noise component. We assume e_i^j to be 0, and a_i^j to be estimated values $\hat{a}_i^j$ for $j \geq H + 1$. The autocorrelation function ρ_i is defined as

$$\rho_i(l) = \frac{E[(a_i^j - \bar{a}_i) \cdot (a_i^{j+l} - \bar{a}_i)]}{\sigma_{a_i}^2}, 0 \leq l \leq H - 1,$$

where, σ_{a_i} is the standard deviation of A_i and l is the *lag* between sample values for which the autocorrelation is computed.

Thus, if the adaptation window size is M intervals (i.e., $M = W/I$), then, we first estimate $\hat{a}_i^{H+1}, \ldots, \hat{a}_i^{H+M}$ using equation 12. Then, the estimated number of arrivals in the adaptation window is given by $\hat{n}_i = \sum_{j=H+1}^{H+M} \hat{a}_i^j$ and finally, the estimated arrival rate, $\hat{\lambda}_i = \dfrac{\hat{n}_i}{W}$.

Estimating the Service Demand The service demand of each incoming request represents the load imposed by that request on the resource. Two applications with similar arrival rates but different service demands (e.g., different packet sizes, different per-request CPU demand, etc.) will need to be allocated different resource shares.

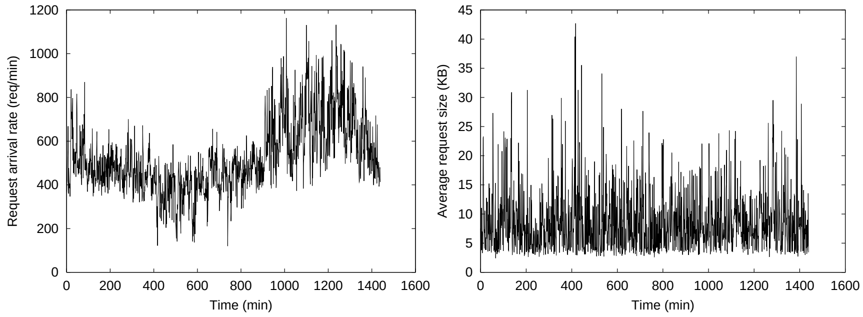
To estimate the service demand for an application, the prediction module computes the probability distribution of the per-request service demands. This distribution is represented by a histogram of the per-request service demands. Upon the completion of each request, this histogram is updated with the service demand of that request. The distribution is used to determine the expected request service demand $\hat{s}_i$ for requests in the next adaptation window. $\hat{s}_i$ could be computed as the mean, the median, or a percentile of the distribution obtained from the histogram. For our experiments, we use the mean of the distribution to represent the service demand of application requests.

Measuring the Queue Length A final parameter required by the allocation model is the queue length of each application at the beginning of each adaptation window. Since we are only interested in the instantaneous queue length q_i^0 and not mean values, measuring this parameter is trivial—the monitoring module simply records the number of outstanding requests in each application queue at the beginning of each adaptation window.

4 Experimental Evaluation

We demonstrate the efficacy of our dynamic resource allocation techniques using a simulation study. In what follows, we first present our simulation setup and then our experimental results.

4.1 Simulation Setup and Workload Characteristics



(a) Request arrival rate (b) Average request size

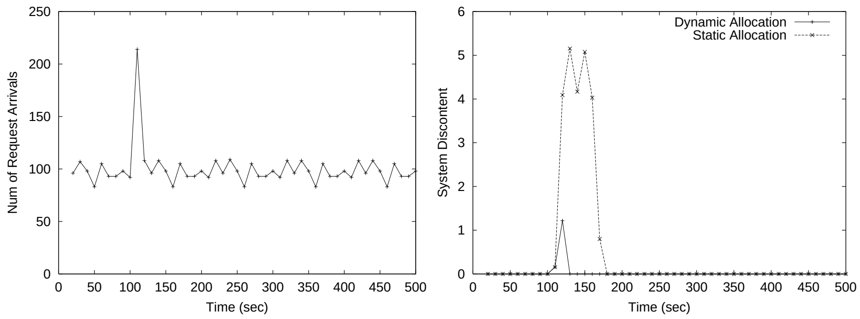
Fig. 4. 24-hour portion of the World Cup 98 trace

Our simulator models a server resource with multiple application-specific queues; the experiments reported in this paper specifically model the network interface on a shared server. Requests across various queues are scheduled using weighted fair queuing [15]—a practical instantiation of GPS. Our simulator is based on the *NetSim* library [22] and *DASSF* simulation package [23]; together these components support network elements such as queues, traffic sources, etc., and provide us the necessary abstractions for implementing our simulator. The adaptation and the prediction algorithms are implemented using the *Matlab* package [28] (which provides various statistical routines and numerical non-linear optimization algorithms); the Matlab code is invoked directly from the simulator for prediction and adaptation.

We use two types of workloads in our study—synthetic and trace-driven. Our synthetic workloads use Poisson request arrivals and deterministic request sizes. Our trace-driven workload is based on the World Cup Soccer '98 server logs [4]—a publicly available web server trace. Here, we present results based on a 24-hour long portion of the trace that contains a total of 755,705 requests at a mean request arrival rate of 8.7 requests/sec, and a mean request size of 8.47 KB. Figures 4 (a) and (b) show the request arrival rate and the average request size respectively for this portion of the trace. We use this trace workload to evaluate the efficacy of our prediction and allocation techniques. Due to space constraints, we omit results related to our prediction technique and those based on longer portions of the trace. More detailed results can be found in a technical report [11].

4.2 Dynamic Resource Allocation

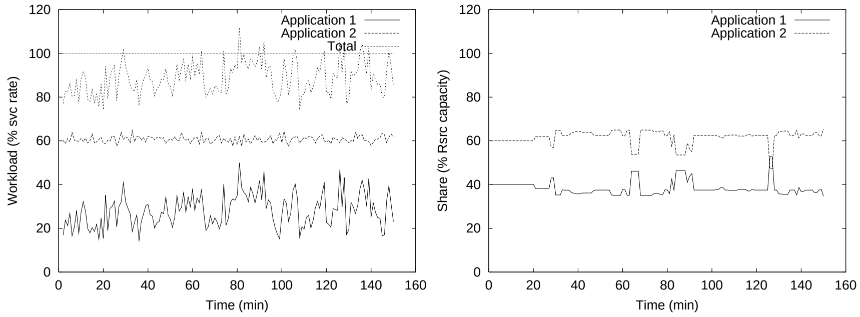
In this section, we evaluate our dynamic resource allocation techniques. We conduct two experiments, one with a synthetic web workload and the other with the trace workload and examine the effectiveness of dynamic resource allocation. For purposes of comparison, we repeat each experiment using a static resource allocation scheme and compare the behavior of the two systems.



(a) Workload of Application 1 (b) System-wide discontent

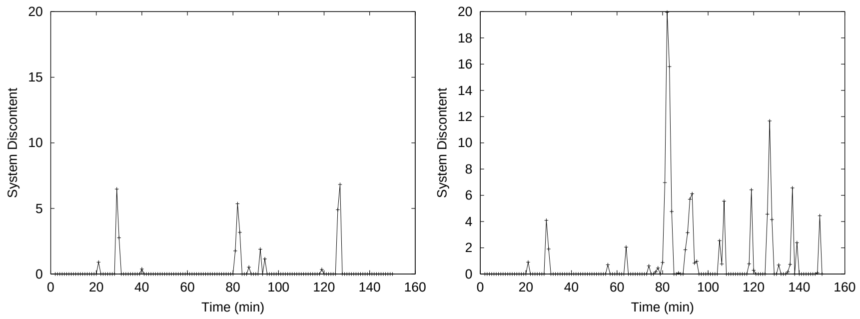
Fig. 5. Comparison of static and dynamic resource allocations for a synthetic web workload.

Synthetic Web Workload To demonstrate the behavior of our system, we consider two web applications that share a server. The benefits of dynamic resource allocation accrue when the workload temporarily exceeds the allocation of an application (resulting in a transient overload). In such a scenario, the dynamic resource allocation technique is able to allocate unused capacity to the overloaded application, and thereby meet its QoS requirements. To demonstrate this property, we conducted a controlled experiment using synthetic web workloads. The workload for each application was generated using Poisson arrivals. The mean request rate for the two applications were set to 100 requests/s and 200 requests/s. Between time t=100 and 110 sec, we introduced a transient overload for the first application as shown in Figure 5(a). The two applications were initially allocated resources in the proportion 1:2, which corresponds to the average request rates of the two applications. ϕ_{min} was set to 20% of the capacity for both applications and the target delays were set to 2 and 10s, respectively. Figure 5(b) depicts the total discontent of the two applications in the presence of dynamic and static resource allocations. As can be seen from the figure, the dynamic resource allocation technique provides better utility to the two applications when compared to static resource allocation and also recovers faster from the transient overload.



(a) Workload (b) Allocation

Fig. 6. The workload and the resulting allocations in the presence of varying arrival rates and varying request sizes.



(a) Dynamic Allocation (b) Static Allocation

Fig. 7. Comparison of static and dynamic resource allocations in the presence of heavy-tailed request sizes and varying arrival rates.

Trace-driven Web Workloads Our second experiment considered two web applications. In this case, we use the World Cup trace to generate requests for the first web application. The second application represents a background load for the experiment;

its workload was generated using Poisson arrivals and deterministic request sizes. For this experiment, ϕ_{min} was chosen to be 30% for both applications and the initial allocations are set to 30% and 70% for the two applications (the allocations remain fixed for the static case and tend to vary for the dynamic case). We present results from only that part of the experiment where transient overloads occur in the system and result in behavior of interest.

Figure 6(a) shows the workload arrival rate (as a percentage of the resource service rate) for the two applications, and also the total load on the system. As can be seen from the figure, there are brief periods of overload in the system. Figure 6(b) plots the resource shares allocated to the two applications by our allocation technique, while Figures 7(a) and (b) show the system discontent values for the dynamic and the static resource allocation scenarios. As can be seen from the figures, transient overloads result in temporary deviations from the desired response times in both cases. However, the dynamic resource allocation technique yields a smaller system-wide discontent, indicating that it is able to use the system capacity more judiciously among the two applications.

Together these experiments demonstrate the effectiveness of our dynamic resource allocation technique in meeting the QoS requirements of applications in the presence of varying workloads.

5 Related Work

Several research efforts have focused on the design of adaptive systems that can react to workload changes in the context of storage systems [3, 26], general operating systems [32], network services [8], web servers [6, 10, 13, 21, 25, 30] and Internet data centers [2, 31]. In this paper, we focused on an abstract model of a server resource with multiple class-specific queues and presented techniques for dynamic resource allocation; our model and allocation techniques are applicable to many scenarios where the underlying system or resource can be abstracted using a GPS server.

Some adaptive systems employ a control-theoretic adaptation technique [1, 25, 27, 34]. Most of these systems (with the exception of [27]) use a pre-identified system model. In contrast, our technique is based on online workload characterization and prediction. Further, these techniques use a linear relationship between the QoS parameter (like target delay) and the control parameter (such as resource share) that does not change with time. This is in contrast to our technique that employs a non-linear model derived using the queuing dynamics of the system, and further, we update the model parameters with changing workload.

Other approaches for resource sharing in web servers [10] and e-business environments [24] have used a queuing model with non-linear optimization. The primary difference between these approaches and our work is that they use *steady-state queue behavior* to drive the optimization, whereas we use *transient* queue dynamics to control the resource shares of applications. Thus, our goal is to devise a system that can react to transient changes in workload, while the queuing theoretic approach attempts to schedule requests based on the steady-state workload. A model-based resource provisioning scheme has been proposed recently [16] that performs resource allocation based on the performance modeling of the server. This effort is similar to our approach of modeling the resource to relate the QoS metrics and resource shares.

Other techniques for dynamic resource allocation have also been proposed in [5, 12]. Our work differs from these techniques in some significant ways. First of all, we define an explicit model to derive the relation between the QoS metric and resource requirements, while a linear relation has been assumed in these approaches. The approach in [5] uses a modified scheduling scheme to achieve dynamic resource allocation, while our scheme achieves the same goal with existing schedulers using high-level parameterization. The approach described in [12] uses an economic model similar to our utility-based approach. This approach employs a greedy algorithm coupled with a linear system model for resource allocation, while we employ a non-linear optimization approach coupled with a non-linear queuing model for resource allocation.

Prediction techniques have been proposed that incorporate time-of-day effects along with time-series analysis models into their prediction [20, 33]. While these techniques work well for online prediction at coarse time-granularities of several minutes to hours, the goal of our prediction techniques is to predict workloads at short time granularities of upto a few minutes and to respond quickly to transient overloads.

Two recent efforts have focused on workload-driven allocation in *dedicated* data centers [17, 31]. In these efforts, each application is assumed to run on some number of dedicated servers and the goal is to dynamically allocate and deallocate (entire) servers to applications to handle workload fluctuations. These efforts focus on issues such as how many servers to allocate to an application, and how to migrate applications and data, and thus are complementary to our present work on *shared* data centers.

6 Conclusions

In this paper, we argued that dynamic resource allocation techniques are necessary in the presence of dynamically varying workloads to provide guarantees to web applications running on shared data centers. To address this issue, we used a system architecture that combines online measurements with prediction and resource allocation techniques. To capture the transient behavior of the application workloads, we modeled a server resource using a time-domain description of a generalized processor sharing (GPS) server. The parameters of this model were continuously updated using an online monitoring and prediction framework. This framework used time series analysis techniques to predict expected workload parameters from measured system metrics. We then employed a constrained non-linear optimization technique to dynamically allocate the server resources based on the estimated application requirements. The main advantage of our techniques is that they capture the transient behavior of applications while incorporating nonlinearity in the system model. We evaluated our techniques using simulations with synthetic as well as real-world web workloads. Our results showed that these techniques can judiciously allocate system resources, especially under transient overload conditions.

In future, we plan to evaluate the accuracy-efficiency tradeoff of using more sophisticated time series analysis models for prediction. In addition, we plan to investigate the utility of our adaptation techniques for systems employing other types of schedulers (e.g., non-GPS schedulers such as reservation-based). We would also like to explore optimization techniques using different utility functions and QoS goals. We also plan to evaluate these techniques with different kinds of workloads and traces. Finally, we intend to compare our allocation techniques with other dynamic allocation techniques to evaluate their relative effectiveness.

References

1. T. Abdelzaher, K. G. Shin, and N. Bhatti. Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1), January 2002.
2. J. Aman, C.K. Eilert, D. Emmes, P Yocom, and D. Dillenberger. Adaptive algorithms for managing a distributed data processing workload. *IBM Sytems Journal*, 36(2):242–283, 1997.
3. E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: Running Circles around Storage Administration. In *Proceedings of the Conference on File and Storage Technologies*, January 2002.
4. M. Arlitt and T. Jin. Workload Characterization of the 1998 World Cup Web Site. Technical Report HPL-1999-35R1, HP Labs, 1999.
5. M. Aron, P. Druschel, and S. Iyer. A Resource Management Framework for Predictable Quality of Service in Web Servers, 2001. http://www.cs.rice.edu/∼druschel/publications/mbqos.pdf.
6. N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, 13(5), September 1999.
7. G. Box and G. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976.
8. A. Brown, D. Oppenheimer, K. Keeton, R. Thomas, J. Kubiatowicz, and D. Patterson. ISTORE: Introspective Storage for Data-Intensive Network Services. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, March 1999.
9. A. Bryson and Y. Ho. *Applied Optimal Control*. Ginn and Company, 1969.
10. J. Carlström and R. Rom. Application-Aware Admission Control and Scheduling in Web Servers. In *Proceedings of the IEEE Infocom 2002*, June 2002.
11. A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. Technical Report TR02-30, Department of Computer Science, University of Massachusetts, 2002.
12. J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, pages 103–116, October 2001.
13. H. Chen and P. Mohapatra. The content and access dynamics of a busy web site: findings and implications. In *Proceedings of the IEEE Infocom 2002*, June 2002.
14. M R. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.
15. A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proceedings of ACM SIGCOMM*, pages 1–12, September 1989.
16. R. Doyle, J. Chase, O. Asad, W. Jin, and Amin Vahdat. Model-Based Resource Provisioning in a Web Service Utility. In *Proceedings of USITS'03*, March 2003.
17. K Appleby et. al. Oceano - sla-based management of a computing utility. In *Proceedings of the IFIP/IEEE Symposium on Integrated Network Management*, May 2001.
18. S.J. Golestani. A self-clocked fair queueing scheme for high speed applications. In *Proceedings of INFOCOM'94*, pages 636–646, April 1994.

19. P. Goyal, H. Vin, and H. Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of the ACM SIGCOMM '96 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 157–168, August 1996.
20. J. Hellerstein, F. Zhang, and P. Shahabuddin. A Statistical Approach to Predictive Detection. *Computer Networks*, January 2000.
21. S. Lee, J. Lui, and D. Yau. Admission control and dynamic adaptation for a proportional-delay diffserv-enabled web server. In *Proceedings of SIGMETRICS*, 2002.
22. B. Liu and D. Figueiredo. Queuing Network Library for SSF Simulator, January 2002. http://www-net.cs.umass.edu/fluidsim/archive.html.
23. J. Liu and D. M. Nicol. DaSSF 3.0 User's Manual, January 2001. http://www.cs.dartmouth.edu/∼jasonliu/projects/ssf/docs.html.
24. Z. Liu, M. Squillante, and J. Wolf. On Maximizing Service-Level-Agreement Profits. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, 2001.
25. C. Lu, T. Abdelzaher, J. Stankovic, and S. Son. A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, June 2001.
26. C. Lu, G. Alvarez, and J. Wilkes. Aqueduct: Online Data Migration with Performance Guarantees. In *Proceedings of the Conference on File and Storage Technologies*, January 2002.
27. Y. Lu, T. Abdelzaher, C. Lu, and G. Tao. An Adaptive Control Framework for QoS Guarantees and its Application to Differentiated Caching Services. In *Proceedings of the Tenth International Workshop on Quality of Service (IWQoS 2002)*, May 2002.
28. Using MATLAB. MathWork, Inc., 1997.
29. A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks – the single node case. In *Proceedings of IEEE INFOCOM '92*, pages 915–924, May 1992.
30. P. Pradhan, R. Tewari, S. Sahu, A. Chandra, and P. Shenoy. An Observation-based Approach Towards Self-Managing Web Servers. In *Proceedings of the Tenth International Workshop on Quality of Service (IWQoS 2002)*, May 2002.
31. S. Ranjan, J. Rolia, and E. Knightly H. Fu. QoS-Driven Server Migration for Internet Data Centers. In *Proceedings of the Tenth International Workshop on Quality of Service (IWQoS 2002)*, May 2002.
32. M. Seltzer and C. Small. Self-Monitoring and Self-Adapting Systems. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, May 1997.
33. F. Zhang and J. L. Hellerstein. An approach to on-line predictive detection. In *Proceedings of MASCOTS 2000*, August 2000.
34. R. Zhong, C. Lu, T. F. Abdelzaher, and J. A. Stankovic. Controlware: A middleware architecture for feedback control of software performance. In *Proceedings of ICDCS*, July 2002.