Dynamic Estimation of CPU Demand of Web Traffic

Giovanni Pacifici, Wolfgang Segmuller, Mike Spreitzer, and Asser Tantawi
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY, USA
{giovanni, werewolf, mspreitz, tantawi}@us.ibm.com

ABSTRACT

Managing the resources in a large Web serving system requires knowledge of the resource needs for service requests of various kinds, and these needs may change over time. Assessing resource needs is commonly performed using techniques such as offline profiling, application instrumentation, and kernel-based instrumentation. Little attention has been given to the dynamic estimation of dynamic resource needs, relying only on external and high-level measurements such as overall resource utilization and request rates. We consider the problem of dynamically estimating dynamic CPU demands of multiple kinds of requests using CPU utilization and throughput measurements. We formulate the problem as a linear regression problem and obtain its basic solution. However, in practice one is faced with issues such as insignificant flows, collinear flows, space and temporal variations, and background noise. In order to deal with such issues, we present several mechanisms such as data aging, flow rejection, flow combining, noise reduction, and smoothing. We implemented these techniques in a Work Profiler component that we delivered as part of a broader system management product. We present experimental results from using this component in scenarios inspired by real-world usage of that product; our technique produces estimates that are roughly within a factor of 2 of the right answer, for the request flows that draw significant CPU power.

1. INTRODUCTION

Web serving systems are becoming increasingly complex, involving a large number of servers, several tiers of interconnected systems, many applications, and fluctuating request patterns. The dynamic management of resources and quality of service in such an environment is of vital importance to the performance and operation of the Web serving system [4] [7]. A key requirement to an effective dynamic resource management is the knowledge of the resource needs of the various kinds of requests. It is important to be able to classify requests with sufficient granularity to support management goals (e.g., multiple service classes within a given application). It is also important to recognize that the resource requirements of a given kind of request may change over time, due to evolution in, e.g., the request parameters or the server states. Focusing on the CPU as the resource of interest, there exist several techniques to assess the CPU cycles consumed by each kind of requests. At application development time, one may utilize profiling mechanisms to measure CPU usage offline, or predict CPU usage of the various branches of an application online. At application run-time, one may instrument application code with calls, such as the ARM (Application Resource Measurement) API [1] to keep track of timimg components at various resources while servicing a (transaction) request. The ARM standard includes measuring availability, performance, usage, and end-to-end transaction response time — all based on an open request classification scheme. An alternative to using high-level calls is kernel-based measurement [9] [8] which may prove to be more efficient but intrusive. Another statistical technique for profiling applications used in Grid computing is Statistical Demand Profiles [6]. Such profiles accumulate over time the distribution of resource needs, and hence could be used for resource allocation.

The profiling techniques that we mentioned so far have several drawbacks: adding instrumentation to the application at development time, relying on middleware or kernel agents to collect and correlate events, incurring overheads in the request execution path, limited ability to track changes in requirements, and/or limited granularity of classification. Little attention has been given to the dynamic estimation of dynamic resource needs, relying only on external and high-level measurements such as overall resource utilization and request rates. We consider the problem of dynamically estimating dynamic CPU demands of various kinds of requests using CPU utilization and throughput measurements. We introduce a problem formulation as a linear regression problem and obtain its basic solution. There are several practical issues that will cause the basic solution to be unstable such as insignificant flows, collinear flows, space and temporal variations, and background noise. We present techniques to deal with such practical considerations such as data aging, flow rejection, flow combining, noise reduction, and smoothing. We have incorporated our dynamic profiling techniques in a Work Profiler component in an IBM product, WebSphere Extended Deployment [2].

The rest of this paper is organized as follows. Section 2 describes the work profiling problem. The solution to the problem is given in Section 3. Section 4 describes practi-

Valuetools'06, October 11-13, 2006, Pisa, Italy

cal enhancements to the basic solution. Section 5 presents experimental results. Finally, in Section 6 we discuss our conclusions and future work.

2. PROBLEM DESCRIPTION

In this section, we describe the dynamic work profiling problem. In general the web serving system will be organized into multiple tiers. For the sake of brevity here we address only one tier; it is straightforward to apply this technique to multiple tiers (provided only the necessary high-level instrumentation). Consider a tier consisting of M machines and K request flows (or, simply flows). The classification of a request into a flow k (for k drawn from $\{1, 2, ..., K\}$) is arbitrary. For example, classification may be based on the URL of the request. A flow is destined to one or more server processes in the tier, depending on the placement of those processes onto machines and the deployment of applications into those processes. We require an infrastructure that can sense the CPU utilization of each machine, producing readings ρ_m in the range [0, 1] occasionally for each machine m. If the infrastructure can provide occasional readings of the CPU utilization of each server process, we can use those instead of the machine utilization readings. We also require occasional sensors for the flow rates $\lambda_{k,m} \geq 0$ req/sec for flow k on machine m. Finally we require knowledge (from sensors and/or configuration management) of the CPU power of each machine m, denoted by Ω_m, in "standard cycles" per second. For simplicity we make an approximation, discussed later, that there is a machine- and application-independent measure of CPU work and call this the "standard cycle". For example: a single clock cycle on one type of machine might do 1.1 standard cycles of work, while a single clock cycle on a less powerful type of machine does 0.8 standard cycles of work. The measurements ρ_m and $\lambda_{k,m}$ are collected periodically. We denote the utilization and flow rates collected during period i (for $i = 1, 2, \ldots$) by $\rho_{m,i}$ and $\lambda_{k,m,i}$, respectively, where $i = 1$ is the current period, $i = 2$ is the previous period, and so on. To simplify our initial description we start by using a sliding window of length I periods to limit the samples used; later we will explain that the actual technique uses all samples with geometric aging.

To simplify our description we presume that the throughput and utilization readings for a given machine are synchronized: in each $\rho_{m,i}$ and $\lambda_{k,m,i}$ pair, we assume both components report on the same period of time. In our implementation, the throughput readings typically follow shortly after the machine utilization readings.

The dynamic work profiling problem is to come up with coefficients α_k, for k in $1, 2, \ldots, K$, that produce a good fit between (a) the observed throught and utilization readings, and (b) the model

$$\rho_{m,i}\Omega_m = \sum_{k=1,2,\ldots,K} \lambda_{k,m,i}\ \alpha_k\ . \tag{1}$$

We use the term "work factor" for one of these coefficients.

We recognize that model (1) is an approximate one. In particular, one aspect of the approximation is the notion that there is an application- and machine-independent measure of CPU work. This is not actually true, but we suspect the approximation is close enough to be valuable in some management systems. If it is not close enough, the techniques in this paper can easily be adapted to a more fine-grained model that recognizes that the CPU cost of a flow may depend on the kind of machine where it is being served.

In order to estimate the K unknowns α_k in equation (1), we need to select a value for I in such a way that we have at least K nonzero equations. We use the variable N for the number of (throughput, utilization) readings used — that is, the number of distinct equations. In the next section we describe the solution to the dynamic work profiling problem equation (1).

3. SOLUTION

We use linear regression [5] to estimate the α_k, for $k = 1, , 2, \ldots, K$, using equation (1).

3.1 Linear Regression Problem

The multivariate linear regression problem with N samples and K independent variables is stated as

$$\mathbf{Y} = \mathbf{X}\,\alpha + \epsilon, \tag{2}$$

where $\mathbf{Y}$ is $(N \times 1)$, $\mathbf{X}$ is $(N \times K)$, the unknown α is $(K \times 1)$, and ϵ is a $(N \times 1)$ vector of error residuals. The random variable ϵ is assumed to have an expectation $E[\epsilon] = 0$ and a $(N \times N)$ variance-covariance matrix $V[\epsilon] = \mathbf{I}\sigma^2$. The unbiased estimate $\hat{\alpha}$ with minimum mean squared error (MSE) is given by

$$\hat{\alpha} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}. \tag{3}$$

Defining

$$\mathbf{C} = \mathbf{X}^T\mathbf{X}$$

and

$$\mathbf{D} = \mathbf{X}^T\mathbf{Y},$$

equation (3) may be written as

$$\hat{\alpha} = \mathbf{C}^{-1}\mathbf{D}. \tag{4}$$

$\mathbf{C}$ is $(K \times K)$ and $\mathbf{D}$ is $(K \times 1)$. The complexity of solving the linear regression problem is dominated by the matrix inversion operation in equation (4). The mean squared error is estimated by

$$MSE = \frac{(\mathbf{Y} - \mathbf{X}\hat{\alpha})^T\,(\mathbf{Y} - \mathbf{X}\hat{\alpha})}{N - K - 1}\ ,$$

which using equation (4) simplifies to:

$$MSE = \frac{\mathbf{Y}^T\mathbf{Y} - \hat{\alpha}^T\,\mathbf{D}}{N - K - 1}. \tag{5}$$

The validity of using a linear model is tested through the coefficient of determination, R^2, which is defined as,

$$R^2 = 1 - \frac{SSE}{SST}\ ,$$

where

$$SSE = \mathbf{Y}^T\mathbf{Y} - \hat{\alpha}^T\,\mathbf{D}\ ,$$

$$SST = \mathbf{Y}^T\mathbf{Y} - N\,\bar{Y}^2,$$

and

$$\bar{Y} = (\mathbf{1}\,\mathbf{Y})\,/N.$$

Here, $\bar{Y}$ is the average value of column $\mathbf{Y}$. Here, $\mathbf{1}$ is a $(1 \times N)$ identity vector. A value of R^2 close to one signifies that there is a linear relationship between $\mathbf{X}$ and $\mathbf{Y}$.

3.2 Solution Goodness Measure

We need a measure of how well the data fits the linear model. We introduce a goodness of fit measure, denoted by g_k, as

$$g_k = \frac{\hat{\alpha}_k}{S_k},$$

where $S_k = \sqrt{\mathbf{S^2}_k}$, $k = 1, 2, ..., K$, and $\mathbf{S^2}$ is a vector of length K, defined as

$$\mathbf{S^2} = MSE \cdot \mathbf{diag}(\mathbf{C}^{-1}),$$

where MSE is given by equation (5).

3.3 Basic Work Profiling Problem

From equations (1) and (2), we have

$$Y_n = \rho_{m,i}\, \Omega_m, \quad \text{and} \quad X_{n,k} = \lambda_{k,m,i}, \tag{6}$$

where $n = 1, 2, ..., N$, is the n^{th} equation from the set of equations (1) in some arbitrary order.

Now we discuss some variations and extensions to the basic work profiling problem.

3.4 Measurement Data Aging

An alternative to using a sliding window of measurements of size I periods is to weight the measurements depending on their age at the time of solving the linear regression problem. The motivation is that older data is less representative of the current behavior than more recent data is. A common age discounting mechanism is to use a weight that decreases geometrically with age. We define the decay rate of our geometric weights in terms of a half-life value of h periods. Thus, equation (6) becomes

$$Y_n = 2^{(1-i)/h}\, \rho_{m,i}\, \Omega_m, \quad \text{and} \quad X_{n,k} = 2^{(1-i)/h}\, \lambda_{k,m,i}. \tag{7}$$

In addition to discounting older data, the geometrical discounting mechanism is advantageous from an implementation point view. Instead of storing the $\mathbf{X}$ and $\mathbf{Y}$ matrices, one can work with the $\mathbf{C}$ and $\mathbf{D}$ matrices which are smaller in size. Updating the $\mathbf{C}$ and $\mathbf{D}$ matrices with measurements obtained for a new period may be achieved by multiplying the matrices by $2^{-2/h}$ and adding the new measured data.

Since old data points decrease in significance and approach zero asymptoytically, we need to modify the actual number of samples to be used when computing the degrees of freedom in equation (5). Let N_h denote a measure of the number of samples when using the data aging technique. We define

$$N_h = \sum_{i=1}^{N} 2^{(1-i)/h} = \frac{1 - 2^{-N/h}}{1 - 2^{-1/h}}.$$

For large N, we get $\lim_{N\to\infty} N_h = 1/(1 - 2^{-1/h})$. To meet the requirement that $N_h \geq K$, we thus need

$$h \geq \frac{1}{-\log_2 (1 - 1/K)}\ .$$

4. PRACTICAL CONSIDERATIONS

In this section, we mention some of the practical considerations that we have encountered.

4.1 Insignificant Flows

Typically, the flow rates $\lambda_{k,m}$ are quite variant, with some flows being so insignificant that we need to remove them from the regression problem. We use the following criterion to identify insignificant flows. Let

$$\bar{\mathbf{X}} = (\mathbf{1}\ \mathbf{X})\ /N_h,$$

where $\mathbf{1}$ is a $(1 \times N)$ identity vector and $\bar{\mathbf{X}}$ is a $(1 \times K)$ vector representing the average values of the columns of $\mathbf{X}$. Further, let $\|\bar{\mathbf{X}}\|$ be the L_1 norm of $\bar{\mathbf{X}}$. Flow $k, k = 1, 2, ..., K$, with $\bar{\mathbf{X}}_k / \|\bar{\mathbf{X}}\| < \delta$, where δ is a small number, e.g. E^{-5}, is removed from the regression problem.

4.2 Utilization Discounting

Since the data becomes more noisy when the machine CPU utilization is low, we need to discount data as the utilization decreases. We choose a simple piecewise linear discounting function of CPU utiliation, $f(\rho_m)$, as $f(0) = 0$, $f(0.2) = 2/3$, and $f(1) = 1$. Thus, equation (7) becomes

$$Y_n = 2^{(1-i)/h}\, f(\rho_{m,i})\, \rho_{m,i}\, \Omega_m,$$
$$X_{n,k} = 2^{(1-i)/h}\, f(\rho_{m,i})\, \lambda_{k,m,i}.$$

4.3 Low Contribution Flows

When solving linear regression problem (2), one often finds that some columns in $\mathbf{X}$ have minimal significant effect on $\mathbf{Y}$, hence leading to improper results $\hat{\alpha}$. Forming a combined variable out of such subset of insignificant variables is a technique that is often used [5]. We developed the following algorithm for combining flows.

- Order the elements in $\bar{\mathbf{X}}$ in decreasing order.
- Let j be the lowest index of element $\bar{\mathbf{X}}_j$ such that $\bar{\mathbf{X}}_j \leq \|\bar{\mathbf{X}}\|$.
- Combine flows $(j+q), (j+q+1), ..., K$, for some value of $q \geq 1$ (we use $q = 2$).
- Solve regression problem with flows $1, 2, ..., (j+q-1)$ and the combined flow.
- For all p where $\hat{\alpha}_p$ fails according the failing criterion described below, add flow p to the combined flow and re-solve problem.

The above loop is repeated a finite number of times. We chose that number to be $2q+1$. The failing criterion for flow p is: $\hat{\alpha}_p < 0$, $\hat{\alpha}_p$ represents a problem with the floating-point calculations, or $g_p < 1$.

4.4 Dynamic Variations

Due to various kinds of noise, the $\hat{\alpha}_k$ values obtained as above are very noisy; to combat this we apply a smoothing technique, producing smoothed values $\tilde{\alpha}_k$. Let $\hat{\alpha}_{k,r}$ be the value of $\hat{\alpha}_k$ obtained at the r^{th} solution of the linear regression problem, and $\tilde{\alpha}_{k,r}$ be its smoothed value, where $r = 1, 2, ...$ We use the following smoothing function

$$\tilde{\alpha}_{k,r} = P_{k,r} / Q_{k,r}, \tag{8}$$

where $P_{k,r} = 2^{-1/t} P_{k,r-1} + s(g_{k,r}) \hat{\alpha}_{k,r}$, $Q_{k,r} = 2^{-1/t} Q_{r-1} + s(g_{k,r})$, $P_{k,0} = Q_{k,0} = 0$, t is a half-life steps period, and $s(g_{k,r})$ is a piecewise linear function of the goodness of fit measure, $g_{k,r}$, obtained for flow k at the r^{th} solution of the

linear regression problem. We use a piecewise linear function with $s(x) = x$, $0 \leq x < 20$, $s(x) = x/20 + 19$, $20 \leq x < 100$, and $s(x) = 24$, $x \geq 100$.

4.5 Collinear Flows

It is quite common to find flows with flow rates that are proportional to each other. For example, the number of *login* requests is usually the same as the number of *logout* requests. Or, every *lookup* request generates a *photoRetrieval* request. In such cases, related flows are called collinear flows since their entries in the $\mathbf{X}$ matrix are nearly proportional, resulting into a near-singular matrix $\mathbf{C}$. Thus, the values of $\hat{\alpha}$ obtained from equation(4) become unstable for the collinear flows. Flow rates can be effectively collinear for two kinds of reasons. One is that the client behavior links the two flows, as suggested above. The other is simpler but perhaps more common: if there is little variation in the throughput of some flows over a long period of time (compared with the aging half-life), they are trivially collinear.

There are several techniques for the detection and handling of collinear flows [5]. This topic is beyond the scope of this paper. We simply state that the smoothing technique given by equation (8) lessens the effect of collinearity through averaging over time. Further, the technique (described earlier) for combining flows also helps cope with collinearity: when there are collinear flows, at least one of their fit results will often meet the failure criterion — which causes a reduction in the number of collinear flows in the regression problem (and this is iterated).

4.6 Machine and Process CPU

The CPU utilization ρ_m for machine m is a coarse measure since it represents a collection of contributions due to a number of running processes in the system. If we imagine that every flow k is processed by a separate server that runs in a separate process, then knowing the CPU utilization measures due to these processes would partition the multivarate linear regression problem into a set of independent univariate linear regression problems. However, in practice, it may be the case that a collection of flows is processed by a single server which runs in a separate process. In such a case, having the CPU utilization of that process partitions the original multivariate linear regression problem into a set of independent, smaller-sized multivariate linear regression problems. This may yield more efficient and accurate results than having only coarse CPU utilization measures. This effect is sought later in Section 5.

4.7 Degrees of Freedom and Responsiveness

The degrees of freedom when solving the linear regression problem (2) is given by $(N - K - 1)$, where N is the number data point samples and K is the number of independent variables, α. In general, the accuracy of the results improves as the number of degrees of freedom increases. Since data samples are collected over time, then the dynamic behavior of α plays a role in deciding on the number of samples N, and hence on the degrees of freedom. If α is stationary, then a large N is desirable. However, in practice, the values of α change with time in reaction to many factors such as the data in the request, the state of the server, the workload on the system, and the mix of requests. Hence, one is faced with a tradeoff between having a large enough N and obtaining estimates for α which are responsive to dynamic behavior. The value of N is adjusted through the selection of the sliding window size I, or the half-life period h if using the data aging technique.

4.8 Background Noise

The CPU utilization measure is usually contaminated by noise that is not due to applications processing requests. One source of such a noise is some background work that varies slowly over time. The contribution of such a background work to the CPU utilization may be captured by adding an independent variable to α and a column of ones to the $\mathbf{X}$ matrix. The value of the additional variable is an estimate of the CPU cycles taken by the backgroung work. A degenerate case arises if the CPU utilization varies little over a long period of time. In such a case, the additional α which captures the background work may be erroneously estimated. We tried this technique for characterizing the background work, and found it often failed in this way. For this reason, we do not recommend doing this.

Another kind of background work is intermittent and quite significant when launched. Examples of such a background work are the starting and stopping of application servers. Such operations may last for tens of seconds and contribute significantly to the CPU utilization. By identifying such periods of time, one may throw out sampled data during such periods as outliers so as not to contaminate the values of $\hat{\alpha}$.

5. EXPERIMENTAL RESULTS

In this section we summarize the results of applying our technique in scenarios inspired by real-world usage. First, we present an experiment that validates the linear model. Then, we present a baseline experiment that uses only whole-machine CPU utilization readings and no collinearity among high-power flows. We then show the results of switching to per-process CPU utilization readings. Then, we summarize the results of some tests focusing on collinearity. Finally, we note briefly the results of some trials with hyperthreaded machines.

5.1 Validation of the Linear Model

In our model validation experiments, we used a micro-benchmarking servlet and program-generated requests. We created four flows on a node, each flow having a CPU utilization between 5% and 25%, the load being varied in a sinusoidal wave with different periods for the flows. The periods were 5, 10, 15, and 20 minutes. This yields repetitive cycles of 300 minutes each. We applied this load for two cycles. The CPU utilization on the node varied between 20% and 97%. The coefficient of determination, R^2, during the experiment was between 0.996 and 0.999, hence validating our linear model. Further, the coefficient of determination was calculated for all the experiments presented in this paper, and was between 0.996 and 0.999 through each of the 90 minute tails. As expected, we did see lower values of the coefficient of determination during transients.

5.2 Setup

In the following experiments we use a scenario and synthetic workload inspired by real-world usage. Our technique is available in a product used in an internal web site at our company. In our experiments we use a micro-benchmarking servlet and program-generated requests so that we can confidently say what the alphas really are; in the real-world

usage the alphas are dependent on server state and request parameters that are not captured in the available traces, so we cannot provide any independent verification of the alphas. To inform evaluation of our results, we did offline profiling of our synthetic workload. We next describe the scenario.

As in the real-world usage, there are 16 distinct flows of requests; we will designate the flows by the letters A through P. These requests are served by application server processes. These server processes are organized into nine clusters: three clusters have two processes each, and the other six clusters have just one process each. The processes are spread across four machines (named xd008 through xd011), with three processes on each. Each machine's processing power is 4,585 standard MHz. Table 1 shows the placement for the experiments without collinearity. Table 2 shows which flows load which clusters; we use the same relationship found in the real world usage. This table also shows the relative magnitudes of the throughputs among those flows. Our experiments with collinearity use the same throughput proportions as the real-world usage; for the experiments without significant collinearity we collapse three flows' throughput onto B.

xd008	xd009	xd010	xd011
C1	C1	C2	C2
C3	C4	C5	C3
C7	C8	C9	C6

Table 1: Placement for experiments with no collinearity.

Flow	Cluster	Rel. throughput (no collinearity)	Rel. throughput (with collinarity)
A	C1	539,304	539,304
B	C2	398,758	277,823
C	C2		98,055
D	C2		22,880
E	C3	22,062	22,062
F	C4	18,794	18,794
G	C5	9,806	9,806
H	C2	4,086	4,086
I	C6	2,697	2,697
J	C7	2,697	2,697
K	C2	1,389	1,389
L	C8	82	82
M	C9	82	82
N	C9	82	82
O	C1	82	82
P	C1	82	82

Table 2: Cluster and relative throughputs for each flow.

In our experiments the arrival process is a Poisson arrival of HTTP sessions, each of which consists of a geometrically distributed number of requests. Each session's behavior is to alternately issue a request and think for a stochastic amount of time. Each request is randomly chosen from among the 16 or 14 flows in play; the probability of each flow is proportional to the desired throughput for that flow. After each think an independent choice is made of whether to terminate the session. All the arrival process parameters are identical across the flows, except for the mean session arrival rate. That is set in the proportions given in Table 2, and total magnitude adjusted so that the CPU utilizations on the machines are limited to roughly the 5–40% range — which is what is observed in the real-world usage.

Request kind	Work Factor (std. MC)
1	39.0
2	18.2
3	13.5
4	11.1
5	5.2
6	5.0
7	3.8
8	6.3
9	3.3
10	3.3
11	3.2

Table 3: Work factor for each kind of request, from offline profiling.

In our experiment we used just one micro-benchmarking servlet, whose alpha depends on request parameters, and 11 different combinations of parameters. The servlet does a controllable combination of arithmetic and sleeping. We generated the 11 combinations of parameters by: (a) using a Zipf-like distribution (the exponent was 1.0) for the parameter that controls the total amount of arithmetic done and (b) using randomly generated values for the parameters (uniform within a limited range) that control the number and length of sleeps. We did offline profiling of those 11 different kinds of requests, and got the results in Table 3. In this section we quantify work factors in units of standard megacycles (std. MC).

We designed our main experiments to explore both the transient and steady-state behavior of our technique. The technique in this paper is targeted to situations where the work factors may change slowly over time. To clearly separate the transient and steady-state results, we structure our workload into a series of five phases. During a given phase, the characteristics of the workload are constant, making it easy to examine the steady-state response of our technique. At each phase change, the workload characteristics jump abruptly to new values, making it easy to examine the transient response of our technique. This choppy aspect of the workload behavior is intentionally artificial, for the purpose of making it easy to see different aspects of the performance of our technique; we expect real workloads change gradually.

Each phase is 2 hours long, so a whole experiment is 10 hours long. During a given phase, a given flow makes only one of our 11 kinds of request; this gives ample time to see whether our technique settles on a value and, if so, how accurate that value is. A given flow makes different kinds of requests in different phases; that is, we have a sharp transient at the beginning of each phase. We use five (one for each phase) randomly generated associations between flow and request kind, as follows. Note that the last five flows (L through P) have extremely low throughputs. To give them the best chance of having an effect, in each phase we bind to them a randomly generated permutation of the five "heaviest" kind of request. For the remaining flows, of which there

	Request kind for flow													
	A	B	E	F	G	H	I	J	K	L	M	N	O	P
Phase 1	8	4	9	3	7	6	2	5	1	5	1	2	4	3
Phase 2	6	2	8	3	7	1	9	5	4	1	3	2	5	4
Phase 3	1	4	5	9	8	6	3	7	2	3	4	1	5	2
Phase 4	3	9	5	4	2	1	7	6	8	3	5	2	1	4
Phase 5	1	6	4	9	7	8	2	3	5	4	3	1	2	5

Table 4: Request kind by phase and flow, no collinearity.

are 9 when avoiding significant collinearity and 11 when not avoiding that, in each phase we bind to those flows a randomly generated permutation of the 9 or 11 heaviest kinds of request. Table 4 shows these randomly generated bindings used in the experiments without significant collinearity.

5.3 Baseline

In our first experiment we avoid significant collinearity and use only whole-machine CPU utilization readings. We divide each phase into two stages: a transient stage of 30 minutes, followed by a tail of 90 minutes. Table 5 shows the maximum power error seen for each flow, in each stage of each phase. The power error for a flow k during period i is

$$\lambda_{k,i} \cdot |\tilde{\alpha}_{k,i} - \alpha_{k,i}|$$

where $\lambda_{k,i}$ is the throughput averaged over that period, $\tilde{\alpha}_{k,i}$ is the alpha estimate produced from that period, and $\alpha_{k,i}$ is the true alpha for that flow at that time. Here we use periods of 1 minute, as in the default configuration of the product. We focus on computational power, which can be quantified by the product of throughput and alpha, because that is what is managed by the product that uses our profiling technique. Errors in the estimated power consumption of a flow are less significant if they are smaller. The average power consumed in each phase is roughly as follows: 550 std. MHz in phase 1, 710 in phase 2, 1720 in phase 3, 620 in phase 4, and 1570 in phase 5.

Figure 1 shows the smoothed alpha estimates produced, as a function of time during the experiment, for flow A. Figure 2 shows the alpha estimates for flow B. Figure 3 shows the estimates for flow F. Our work profiler produced estimates significantly above 1 for only A, B, and F.

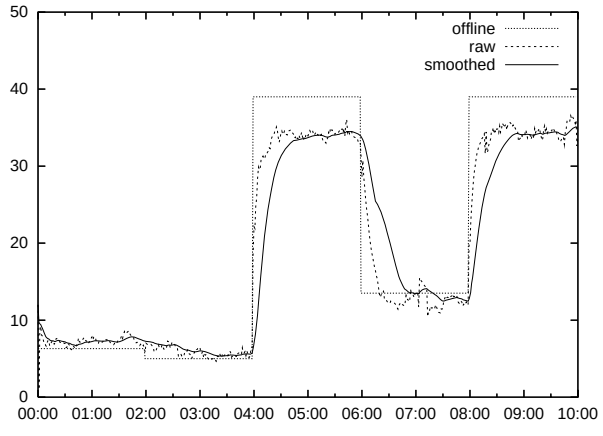

Figure 1: Work factor estimates (in std. MC) for no collinearity, machine CPU, flow A.

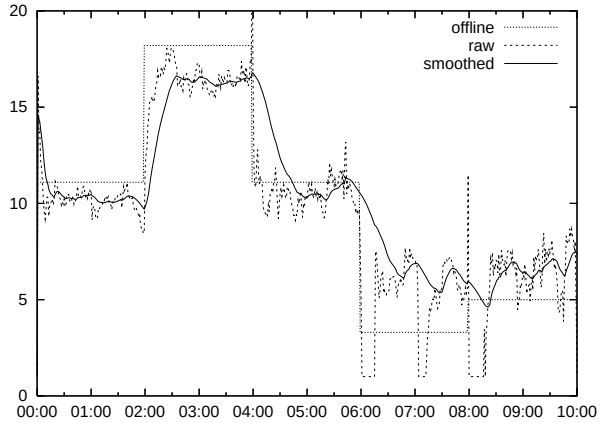

Figure 2: Work factor estimates (in std. MC) for no collinearity, machine CPU, flow B

Table 6 summarizes the alpha and power results for the transient stage of phase 1. For each flow it gives: the minimum and maximum observed throughputs, as averaged over 1 minute periods; the smallest and largest alpha estimates produced during that stage; the alpha derived from offline profiling; the smallest and largest power estimates ($\lambda_{k,i}\tilde{\alpha}_{k,i}$) produced during that stage; and the maximum power error for that stage. Table 7 gives the same quantities for the tail of phase 1. Among the five phases, 1 had the smallest power errors.

Phase 4 had the worst power errors in its tail. Tables 8 and 9 show the results from the transient and tail stages of that phase.

5.4 Introducing Per-Process CPU Readings

Our second experiment was the same as the first, except that the work profiler was configured to use per-process CPU utilization readings. We expected this would allow the profiler to produce more accurate estimates. Table 10 shows this generally did not happen (cf Table 5). We do not yet understand this fully, but can make two important observations. One is that the synchronization between throughput and utilization readings is less good for the per-process utilization readings. The other observation concerns the hardness of the problems. The first collinearity experiment was not really much harder than the whole-machine utilization-reading experiment because the high-power flows (these were at most A, B, and E) all had different placements. Inhomogeneous placement produces samples from different configurations, making it easier to disentangle the effects of multiple flows.

We show some traces of alpha estimates. Figure 4 shows

Phase	Stage	Max power error (in std. MHz) for flow													
		A	B	E	F	G	H	I	J	K	L	M	N	O	P
1	transient	121.55	87.07	3.90	17.87	2.12	1.36	4.30	1.03	5.28	0.12	0.63	0.19	0.22	0.35
1	tail	56.16	35.37	4.02	25.53	2.12	1.49	4.98	1.15	5.91	0.09	0.63	0.38	0.22	0.35
2	transient	87.16	242.37	29.45	28.79	2.15	12.66	0.54	0.96	1.46	0.64	0.14	0.38	0.07	0.17
2	tail	68.09	60.36	57.82	17.79	2.40	16.21	0.61	0.98	1.91	0.85	0.28	0.38	0.09	0.22
3	transient	1147.25	155.75	62.41	33.72	4.27	1.48	2.65	0.66	2.30	0.28	0.17	1.06	0.09	0.29
3	tail	291.98	37.70	28.31	42.36	4.39	1.60	3.28	0.76	2.58	0.35	0.28	1.05	0.12	0.29
4	transient	747.09	199.96	7.23	28.03	13.61	13.51	0.65	1.09	0.68	0.21	0.09	0.38	0.84	0.28
4	tail	271.39	114.13	7.30	35.20	13.72	15.19	0.76	1.00	0.94	0.28	0.09	0.38	0.85	0.28
5	transient	1027.08	28.51	16.90	30.75	2.19	1.80	4.01	3.12	0.65	0.17	0.21	0.63	0.29	0.12
5	tail	312.11	68.48	17.06	37.25	2.32	1.83	4.87	3.13	0.61	0.22	0.21	0.84	0.38	0.12

Table 5: Power errors for no collinearity, machine CPU.

Flow	λ		α			power		power err
	min	max	min est	max est	profiled	min est	max est	max
A	31.890	39.168	7.2	9.4	6.3	233.102	365.139	121.548
B	24.592	28.533	10.3	14.2	11.1	259.229	395.639	87.068
E	1.261	1.694	1.0	1.0	3.3	1.261	1.694	3.897
F	1.022	1.430	1.0	4.6	13.5	1.022	5.314	17.872
G	0.522	0.755	1.0	1.0	3.8	0.522	0.755	2.115
H	0.211	0.339	1.0	1.0	5.0	0.211	0.339	1.356
I	0.117	0.250	1.0	1.0	18.2	0.117	0.250	4.300
J	0.117	0.244	1.0	1.0	5.2	0.117	0.244	1.027
K	0.050	0.139	1.0	1.0	39.0	0.050	0.139	5.277
L	0.000	0.028	1.0	1.0	5.2	0.000	0.028	0.117
M	0.000	0.017	1.0	1.0	39.0	0.000	0.017	0.633
N	0.000	0.011	1.0	1.0	18.2	0.000	0.011	0.191
O	0.000	0.022	1.0	1.0	11.1	0.000	0.022	0.224
P	0.000	0.028	1.0	1.0	13.5	0.000	0.028	0.347

Table 6: Results for no collinearity, machine CPU, phase 1, transient.

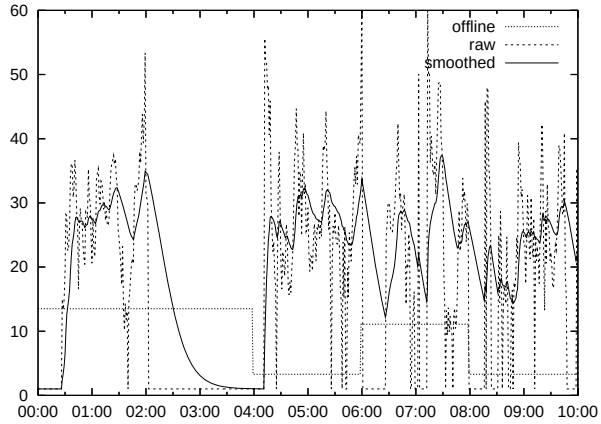

Figure 3: Work factor estimates (in std. MC) for no collinearity, machine CPU, flow F

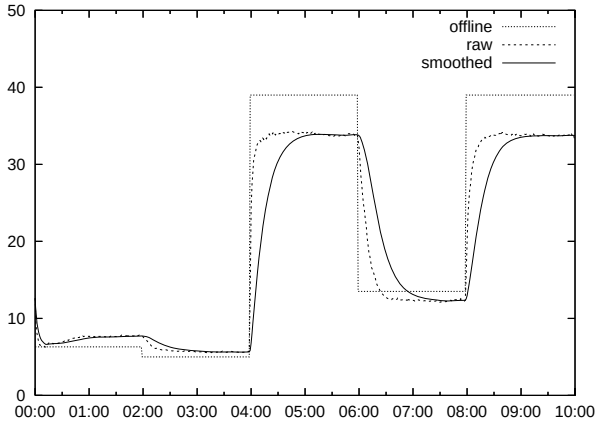

Figure 4: Work factor estimates (std. MC) for no collinearity, process CPU, flow A.

the trace for flow A, figure 5 for flow B, and figure 6 for flow J (which had relatively high power errors).

5.5 Collinearity

We next studied harder cases of collinearity. The easy case is when the collinear flows target different clusters. It is more difficult when collinear flows target the same cluster, and those flows are not combined as in section 4.3. To study this we began by un-collapsing C and D from B. That in turn required new bindings between flow and request kind. Table 11 shows these bindings (which were generated in the stochastic manner outlined above).

Notice that in these bindings, we never have high power draw in two or more out of the three flows B, C, and D. We found that in this relatively non-challenging situation, our work profiler was about as accurate as in the experiments

Flow	λ		α			power		power err
	min	max	min est	max est	profiled	min est	max est	max
A	32.824	38.531	6.7	7.8	6.3	225.507	287.044	56.162
B	23.882	28.518	9.7	10.5	11.1	243.920	297.079	35.375
E	1.094	1.750	1.0	1.0	3.3	1.094	1.750	4.025
F	1.011	1.467	6.5	33.4	13.5	9.317	44.995	25.529
G	0.467	0.756	1.0	1.0	3.8	0.467	0.756	2.115
H	0.167	0.372	1.0	1.0	5.0	0.167	0.372	1.489
I	0.111	0.290	1.0	1.0	18.2	0.111	0.290	4.982
J	0.078	0.273	1.0	1.0	5.2	0.078	0.273	1.145
K	0.044	0.156	1.0	1.0	39.0	0.044	0.156	5.910
L	0.000	0.022	1.0	1.0	5.2	0.000	0.022	0.093
M	0.000	0.017	1.0	1.0	39.0	0.000	0.017	0.634
N	0.000	0.022	1.0	1.0	18.2	0.000	0.022	0.383
O	0.000	0.022	1.0	1.0	11.1	0.000	0.022	0.224
P	0.000	0.028	1.0	1.0	13.5	0.000	0.028	0.347

Table 7: Results for no collinearity, machine CPU, phase 1, tail.

Flow	λ		α			power		power err
	min	max	min est	max est	profiled	min est	max est	max
A	33.123	38.278	21.4	33.9	13.5	755.908	1261.572	747.090
B	24.957	28.430	7.5	10.5	3.3	191.242	291.342	199.963
E	1.267	1.736	1.0	1.1	5.2	1.286	1.792	7.232
F	1.081	1.441	12.2	33.7	11.1	14.289	41.788	28.027
G	0.533	0.791	1.0	1.0	18.2	0.533	0.791	13.612
H	0.200	0.356	1.0	1.0	39.0	0.200	0.356	13.510
I	0.133	0.233	1.0	1.0	3.8	0.133	0.233	0.653
J	0.150	0.272	1.0	1.0	5.0	0.150	0.272	1.089
K	0.056	0.128	1.0	1.0	6.3	0.056	0.128	0.679
L	0.000	0.017	1.0	1.0	13.5	0.000	0.017	0.208
M	0.000	0.022	1.0	1.0	5.2	0.000	0.022	0.093
N	0.000	0.022	1.0	1.0	18.2	0.000	0.022	0.382
O	0.000	0.022	1.0	1.0	39.0	0.000	0.022	0.844
P	0.000	0.028	1.0	1.0	11.1	0.000	0.028	0.280

Table 8: Results for no collinearity, machine CPU, phase 4, transient.

reported earlier above.

To explore more challenging settings, we conducted further experiments with an altered placement and flow/alpha binding. Table 12 shows that placement. Table 13 shows the bindings for the most significant flows.

xd008	xd009	xd010	xd011
C1	C1	C3	C3
C2	C2	C5	C4
C7	C8	C9	C6

Table 12: Placement for second experiments with collinearity.

Flow	Request kind
A	4
B	3
C	2
D	1

Table 13: Flow/request binding highlights for second experiments with collinearity.

Figure 7 shows the results for flows A, B, C, and D when using only whole-machine CPU readings. Figure 8 shows the results for those flows when per-process CPU readings are used. Each chart shows both the on-line alpha estimates and the ones produced from offline profiling (on the right-hand edge). We see that the per-process utilization readings produced a better estimate for flow A and worse estimates for the others; flow A draws significantly more power than the others, so the finer granularity readings at least improved the most significant answer.

5.6 Hyperthreading

The preceding experiments were done with machines that were not hyperthreaded. We found that using hyperthreading has two interesting effects. One is that the machine's power is increased, but by a factor that is significantly smaller than the one you might naively expect (2). The other is that the CPU utilization reading are distorted. When a hyperthreaded machine is delivering half of its maximum possible computing power, Linux reports its CPU utilization as something much less than one half. We think this distortion is due to the interaction of the hyperthreading architecture with the way the OS estimates CPU utilization. We characterized the distortion; it fits rather well to a polynomial of degree 3. The inverse of the distortion function

Flow	λ		α			power		power err
	min	max	min est	max est	profiled	min est	max est	max
A	31.416	39.579	12.5	21.0	13.5	423.779	770.291	271.389
B	24.340	29.351	5.4	7.4	3.3	137.409	209.453	114.129
E	1.217	1.739	1.0	1.0	5.2	1.217	1.739	7.303
F	0.961	1.472	14.6	37.4	11.1	17.140	50.118	35.200
G	0.482	0.798	1.0	1.0	18.2	0.482	0.798	13.721
H	0.172	0.400	1.0	1.0	39.0	0.172	0.400	15.192
I	0.106	0.272	1.0	1.0	3.8	0.106	0.272	0.762
J	0.089	0.249	1.0	1.0	5.0	0.089	0.249	0.998
K	0.039	0.178	1.0	1.0	6.3	0.039	0.178	0.942
L	0.000	0.022	1.0	1.0	13.5	0.000	0.022	0.279
M	0.000	0.022	1.0	1.0	5.2	0.000	0.022	0.093
N	0.000	0.022	1.0	1.0	18.2	0.000	0.022	0.381
O	0.000	0.022	1.0	1.0	39.0	0.000	0.022	0.846
P	0.000	0.028	1.0	1.0	11.1	0.000	0.028	0.280

Table 9: Results for no collinearity, machine CPU, phase 4, tail.

		Power error (std. MHz) for flow													
Ph	Stg	A	B	E	F	G	H	I	J	K	L	M	N	O	P
1	trns	95.59	174.73	133.43	40.53	76.69	179.46	10.92	8.74	26.19	3.47	1.34	1.22	0.22	1.63
1	tail	52.07	59.17	119.51	29.02	70.80	59.75	72.56	102.14	14.29	7.01	2.78	3.35	5.25	6.11
2	trns	105.45	225.37	21.96	12.31	12.37	8.52	15.97	25.31	1.34	4.13	1.62	4.55	2.44	0.66
2	tail	50.59	78.61	25.67	14.49	13.55	7.97	11.82	10.16	11.58	6.77	3.52	5.56	1.98	2.64
3	trns	1159.79	154.83	26.01	26.59	12.12	38.12	7.62	10.51	3.65	4.46	1.13	1.73	1.90	19.73
3	tail	321.71	23.84	25.09	21.69	12.83	18.92	8.82	11.77	1.97	15.09	8.49	4.22	11.29	4.30
4	trns	754.41	196.95	24.79	11.09	6.72	12.30	11.01	10.31	0.56	1.88	3.69	4.17	0.79	0.59
4	tail	194.96	115.73	24.31	13.40	8.77	11.00	11.10	11.14	5.52	5.70	2.44	4.72	1.53	1.58
5	trns	1031.47	16.07	18.50	23.78	17.33	5.07	7.09	7.82	0.92	3.18	1.30	1.75	1.72	0.10
5	tail	333.84	32.88	20.70	20.60	14.63	6.16	8.18	12.57	2.22	4.90	3.33	1.52	0.23	1.82

Table 10: Power errors (in std. MHz) for no collinearity, process CPU.

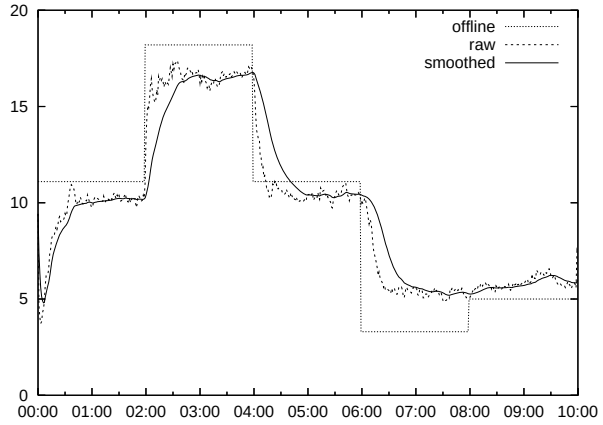

Figure 5: Work factor estimates (std. MC) for no collinearity, process CPU, flow B.

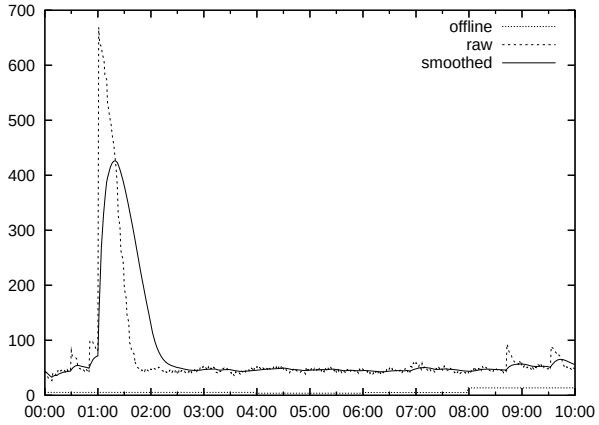

Figure 6: Work factor estimates (std. MC) for no collinearity, process CPU, flow J.

also fits a degree 3 polynomial very well. We repeated experiments like those above but with hyperthreading enabled. When we used an appropriately adjusted figure for the machine's power (Ω), and applied a correcting transformation to the CPU utilization readings, we got very similar on-line profiling results.

6. CONCLUSION

We considered the problem of dynamically estimating CPU demands of applications using CPU utilization and throughput measurements. Using a linear model, we formulated the problem as a multivariate linear regression problem. We addressed practical issues such as insignificant flows, collinear flows, space and temporal variations, and background noise. We presented techniques to deal with such practical consid-

	Request kind for flow															
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Phase 1	4	3	11	9	10	6	2	1	8	7	5	2	4	5	3	1
Phase 2	5	11	6	2	3	7	8	10	1	4	9	2	1	3	5	4
Phase 3	8	1	9	3	5	2	11	7	10	4	6	2	3	4	5	1
Phase 4	10	6	5	11	7	2	8	3	4	9	1	3	2	4	1	5
Phase 5	1	11	6	7	10	3	4	2	9	8	5	2	1	5	4	3

Table 11: Request kind by phase and flow, first collinearity experiments.

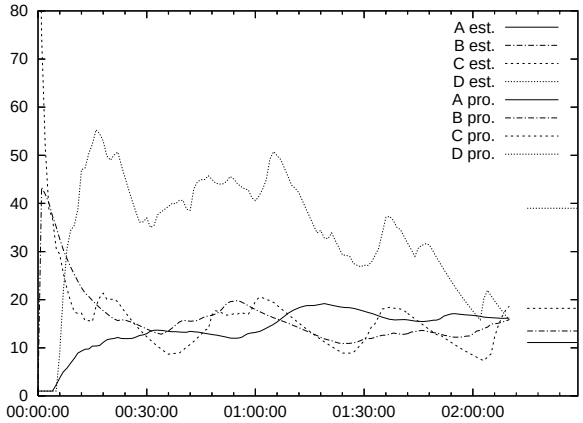

Figure 7: Work factor estimate (std. MC) traces for collinearity, machine CPU.

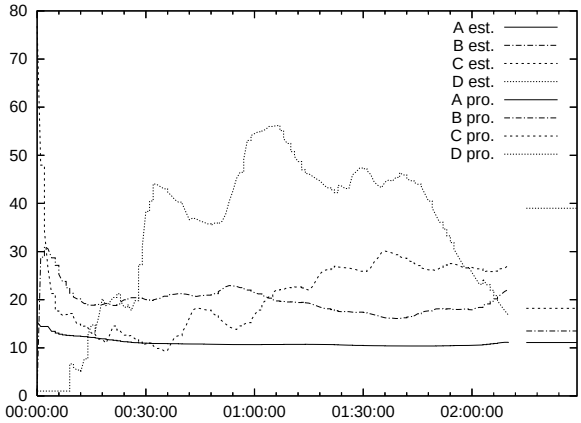

Figure 8: Work factor estimate (std. MC) traces for collinearity, process CPU.

erations. The experimental results proved that our approach is a viable one for the rough estimation of dynamic CPU demand of Web applications.

Interestingly, some of the experimental results present us with a few challenges to be considered as future enhancements. In particular, we notice a bias between the estimated and predicted demands. Further, we see that the smoothed results take minutes to respond to variations in the demand, calling for an investigation of the tradeoff between precision and responsiveness. Special mechanisms to deal with collinear flows, other than just combining flows and averaging through smoothing, are needed. One viable approach is to use collinearity detection mechanisms and ridge regression [3]. The effects of hyper-threading need more investigation. Capturing the background noise in CPU utilization through randomization needs further investigation. Also, scaling and standardization techniques may help with numerical stability [5].

There are several extensions to this work. We have only considered the CPU as the bottleneck resource. In some situations, there may be multiple resources, yielding a multidimensional regression model. The complexity of our approach is dominated by a matrix inversion operation, which may be costly for a very large problem. Other approaches based on stochastic search techniques are under investigation.

7. REFERENCES

[1] *Application Resource Measurement - ARM.* http://www.opengroup.org/tech/management/arm/.

[2] *IBM WebSphere Extended Deployment.* http://www.ibm.com/software/webservers/appserv/extend/.

[3] A. Hoerl and R. Kennard. Ridge regression: Biased estimation for non-orthogonal problems. *Technometrics*, 12, 1970.

[4] G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef. Performance management for cluster based web services. *IEEE Journal on Selected Areas in Communications*, 23(12), December 2005.

[5] J. O. Rawlings. *Applied Regression Analysis: A Research Tool.* Wadsworth & Brooks, Cole Advanced Books & Software, Pacific Grove, CA, 1988.

[6] J. Rolia, X. Zhu, and M. Arlitt. Resource access management for a utility hosting enterprise applications. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, March 2003.

[7] H. Senger and L. M. Sato. Load distribution for heterogeneous and non-dedicated clusters based on dynamic monitoring and differentiated services. In *Proceedings of the International Conference on Computer Communication*, 2003.

[8] B. Urgaonkar and P. Shenoy. Sharc: Managing cpu and network bandwidth in shared clusters. *IEEE Transactions on Parallel and Distributed Systems*, 15(1), January 2004.

[9] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.