Online Web Cluster Capacity Estimation and its Application to Energy Conservation

Chang-hao Tsai, Kang G. Shin,
John Reumann, Sharad Singhal
chtsai@umich.edu, kgshin@umich.edu,
reumann@us.ibm.com, sharad.singhal@hp.com

The University of Michigan, IBM Research, and HP Labs

Abstract

Designers of data centers and Web servers aim to provide clients resources on demand to decrease the initial cost of hosted service deployment models. In addition, they must also minimize operating cost, such as energy consumption, by matching service capacity demand with resource supply. However, since the term "capacity" is typically defined vaguely or inadequately, it is difficult to assess resource needs in a running system and usually results in oversized server deployment. To address this problem, we first define the *capacity* of a server cluster as the sustainable throughput subject to a request retransmission ratio constraint. Then, we analyze different approaches of capacity estimation.

Various capacity estimation mechanisms, including off-line benchmarking, CPU-utilization-based, and queue monitoring (proposed here), are studied and compared. We also employ several different data-collection methods (application instrumentation, user-space tools, SNMP, and kernel modules) to compare their effects on estimation accuracy. Among these, queue monitoring is found to provide a good and stable estimate of server capacity. To validate our findings we applied our proposed estimation method to server cluster sizing for energy conservation. A good combination of data collection and online capacity estimation is found to make significantly more energy savings than traditional approaches (i.e., static estimation and scheduled capacity). Our experimental results show that more than 40% of energy can be saved for regular daily usage patterns without requiring any prior knowledge of the workload and also that long start-up and shutdown delays affect energy conservation considerably.

Keywords

World Wide Web, Server Cluster, Resource Management, Power Management, Capacity Provisioning.

Authors

Chang-hao Tsai, +1-734-763-6131, chtsai@eecs.umich.edu, The University of Michigan, Ann Arbor
Kang G. Shin, +1-734-763-0391, kgshin@eecs.umich.edu, The University of Michigan, Ann Arbor
John Reumann, +1-914-784-3122, reumann@us.ibm.com, IBM Research, Hawthorne
Sharad Singhal, +1-650-857-5907, sharad.singhal@hp.com, HP Laboratories, Palo Alto

Introduction

- **Server clusters provide higher availability and capacity.**
- **Capacity is usually statically and conservatively under-estimated to handle peak load, but...**
  - Servers are under-utilized (excessive equipment purchase)
  - Powering and cooling cost are increasing (operating cost)
  - Capacity changes as content changes dynamically
- **With an accurate capacity estimation, datacenters can dynamically allocate resources to services.**
  - Improve utilization (hosting more services)
  - Reduce energy cost (turn off unused servers)
  - Ensure service quality
- **We estimate capacity by monitoring queues. With simple controls, >40% power can be saved.**

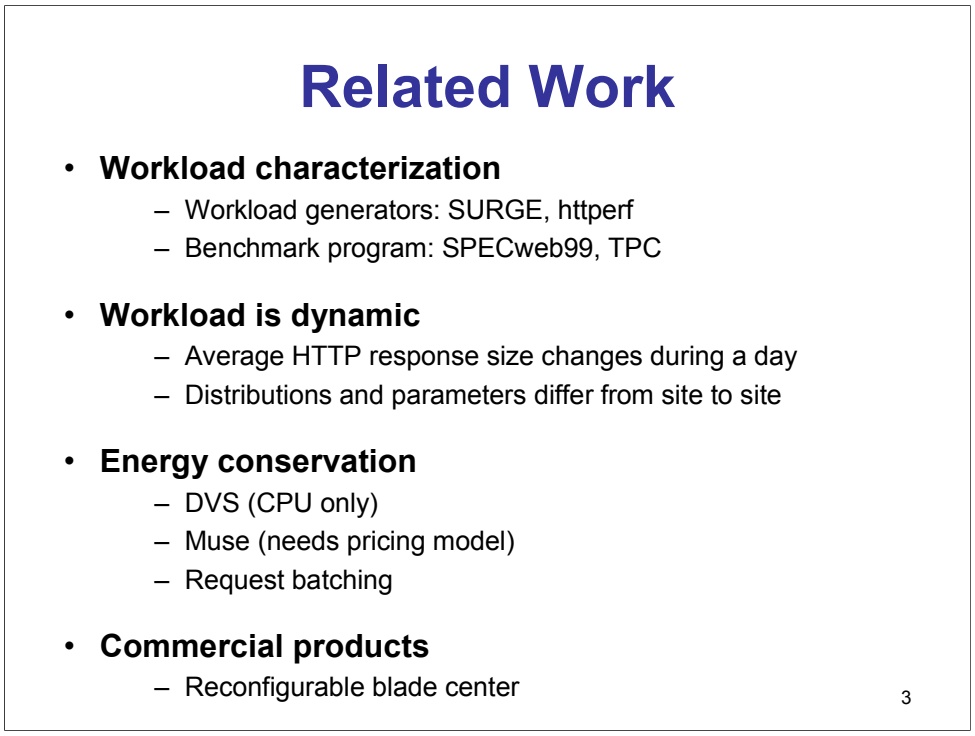
2

Clustered servers are commonly used to provide highly available and scalable services. The *capacity*, or maximum sustainable throughput, of a scalable server cluster is approximately the sum of all individual servers' capacities. However, the capacity of each server is typically unknown. Capacity is also workload-dependent and highly-variable. This is due mainly to the unpredictability of resource demands, content change, and the nonlinear scaling within individual servers.

For the above reasons, the current engineering approach to capacity estimation is to conservatively under-estimate the capacity of each server based on a coarse-grained (weekly or monthly) system utilization check and log analysis that describes user demands as well as server performance. Usually, linear scaling is assumed and then equipment is purchased if the capacity estimate of the current infrastructure does not allow the safety margin that practitioners value. Thus, servers are almost always severely under-utilized. While the excessive equipment purchase may not present a major problem to an IT-driven organization or data centers, excess capacity allocation generally reduces overall system utilization and directly reduces the profit that a service provider can achieve from a given IT infrastructure. The operational cost of powering and cooling the excess equipment reduces the profit margin further. Nonetheless, reducing available capacity to lower operational cost is not an option because this would not allow the system to handle peak loads. Handling peak loads, however, is an important deployment consideration because every lost request translates into lost revenue, which is typically much greater than the marginal cost of energy expenditure. Furthermore, we argue that an application service provider (ASP) also runs the risk of degrading customer satisfaction without appropriate online capacity estimation and may potentially face service outages. The economic reasons for better estimation of capacity are, therefore, compelling.

We explore the realities, pitfalls, and techniques of online or dynamic capacity estimation as it is an important building block in capacity planning, on-demand computing, and energy conservation in server systems. A symmetric server cluster model is used in this paper. After observing the correlation between queue length and client-perceived response time, we define “drop ratio” as a measure of user satisfaction and the true “capacity” of server systems. We propose a new scheme based on queue-length monitoring to estimate server capacity and compare it with other simpler estimation mechanisms.

To evaluate our approach, we implement and compare several different measurement and estimation schemes. Our estimation approach is also demonstrated by its application to the well-known energy conservation problem [1]. In this scenario, we resize a cluster of web servers by utilizing a relatively simple controller to determine cluster size and dynamically powering them up and down to conserve energy. Our ability to realize substantial energy savings (in excess of 40%) demonstrates the accuracy and applicability of our capacity-estimation method. We also show that long start-up and shutdown delays do limit the result of energy conservation.



Significant research efforts have been made on web servers' workload characteristics [2,3,4], as understanding the user model can facilitate the use of workload generators [5,6] and benchmark programs [7] to study server performance problems. However, changes of the average HTTP response size over time of day suggests that users' browsing behavior (which might be affected by browser's caching) also changes over time of day [8]. In addition, more recent workload characterization of dynamic content website indicates diversities in the workload model and parameters as well [9]. Both evidences suggest that online capacity estimation is the key tool in service provisioning.

Cluster Reserves [10] extend resource containers [11] to server clusters and achieve service isolation while services share each server node. However, as server blades provide many small servers to service providers, estimating server capacity and allocating an appropriate number of servers to each single service is an simpler approach.

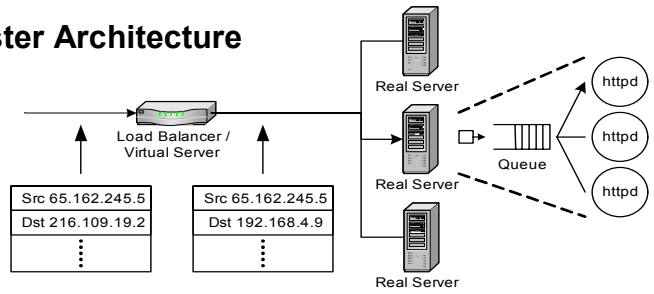
Any performance requirement must come from the users' perspective. Previous research showed that a response time larger than 10 seconds causes users to think there is an error in the system [12]. Many researchers attempted to measure client-perceived response time. In [13], HTML document instrumentation is used to assess response time. However, the result does not include a TCP connection setup time and is only applicable to HTTP requests. Alternatively, in [14], TCP client-server interactions, including any packet drops, are used to infer response time. However, as data centers do not have control of network delay or packet loss in the network, using client-perceived response time directly as a service quality measure for the server is inappropriate and misleading.

Dynamic Voltage Scaling (DVS) [15] is a well-known technique to reduce energy consumption and has also been applied to web servers. Bohrer [16] showed via simulation that CPU energy consumption can be reduced by up to 36%. Our cluster resizing mechanism can be combined with DVS to improve energy conservation. Chase [1] applied an economic approach, called Muse, to manage energy and other resources in data centers. In Muse, the value of each resource is quantified in utility functions and services place bid on resources. They make control decisions by maximizing service revenue and profit, and their results show that 29% or more energy can be saved for a typical workload. However, we argue that resources cannot be priced easily and the resource price could also be highly time-variant. Since resource utilization and performance-resource relationships are difficult to determine, it could make data center operation more complicated and even infeasible. In this paper, we show that similar energy savings can be achieved by a much simpler method.

Rajamani [17] defined system and workload characteristics which could affect energy savings in server clusters and applied control to demonstrate that having knowledge of these characteristics can improve the result of a simple threshold control approach. Unfortunately, a load profile, which is assumed to be available in that study, may not always be available. Pinheiro [18] also applied power management to cluster-based systems, including a web server cluster. They estimate the resource utilizations on each server and add them up to predict the total resource requirement. They also showed that mismatches between decision frequency and workload-change rate can affect service quality. However, resource demand and utilization typically exhibit a non-linear relationship, and performance is not directly managed in the control loop. Elnozahy [19] proposed a request batching policy to reduce energy consumption. It can be combined with DVS and our method to maximize energy savings.

Server Cluster Model

- **Cluster Architecture**



- **Sustainable throughput := maximal throughput subject to not violating certain thresholds performance metrics**

- **Capacity is determined by**
  - Request arrival process, type of request (dynamic, static)
  - Users' performance expectations (patience)
  - Server configuration (hardware, software)
  - Queue size

4

A typical server cluster consists of a load-balancer and a set of servers. In this paper, we assume that the servers have identical hardware and software configurations, so client requests can be dispatched to arbitrary servers and service time is (nearly) independent of the real server being used. Servers of different configurations or servers that can be partitioned would only require straightforward extensions to the results presented in this paper. HTTP/1.0 encapsulates each request in a new TCP connection, which is chosen as the basic unit of service. The request arrival process and the implicit resource demand imposed by each request (and also each TCP connection) are assumed to follow some probabilistic distribution that is unknown to the server cluster's operator. Although in HTTP/1.1 connections are reused to send multiple requests, our model is still valid as the requests in one connection can be modeled as one macro request. From our perspective only arrival process and service time distribution are affected.

Before each request is processed, it may be queued at either the load-balancer or one of the real servers. A request would only be queued at the load-balancer if the network link toward the chosen real server is being used. Since the HTTP traffic is usually asymmetric and the network devices can forward packets with very low blocking probabilities, it is very unlikely that many request packets would be queued at the load-balancer. Therefore, we assume that the load-balancer does not drop any request and its queue length is ignored in our model. On the other hand, a request is always queued at the server before a process can pick it up due to TCP. The size of the server queue (backlog) is determined by the server program and the OS configuration. A server cannot accept any new connection if the backlog is full. The request is dropped silently, and it will be retransmitted automatically after a timeout. Ideally, the queue length in active servers will have the same distributions. With actual queue-length measurements from more than one server, one can estimate the underlying distribution, and therefore server capacity, with higher accuracy. Although some servers may adapt their service quality (*e.g.*, the richness in the provided content) in certain situations [20], we assume that the resource demands by incoming requests are not affected by cluster status. If lower quality content were equally good for business, it would be wise to always provide lower quality content. Otherwise, buying additional equipment is generally a cheaper and more predictable approach.

The server capacity is determined by several factors. First, the arrival process of requests and the nature of each individual request determine both instantaneous and average resource requirements. Second, users' performance expectations and patience define acceptable service levels, which, in turn, affect the meaning of the term "sustainable throughput" and hence real system capacity. Third, server hardware and software configurations directly influence performance and resource requirements. Furthermore, configured queuing limits on the server side affect the maximal surge size that can be absorbed by the server. Depending on the request arrival process, this can affect average client-perceived delays, because a request packet drop results in a long delay due to slow TCP SYN packet retransmission. All of these inter-related factors make accurate capacity estimation a very difficult problem. These issues have not been fully explored because servers are often tested using relatively predictable workloads (e.g., typical web benchmarks), or measured in terms of long running averages that hide inaccuracies in system capacity estimation. With a resizable server cluster like this, it becomes possible to adapt the size of a cluster online to match the demand. Data center administrators would like to optimize the allocation of resources to achieve the highest utilization possible, and therefore, maximize their revenue. On the other hand, they must always provide a high level of service quality to their customers.

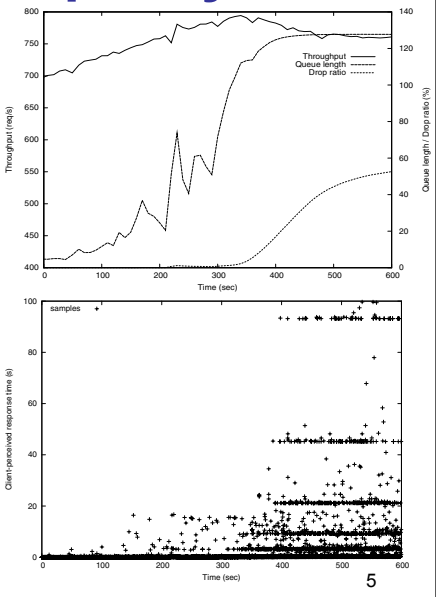
Estimating Capacity

- **Delay-based**
  - difficult to measure
  - high variance
  - arbitrary definition of threshold
  - depend on content served
- **TCP SYN retransmission causes poor response time**
- *Request Drop Ratio*

  # of failed connection attempts

  # of connection attempts
- *Capacity*

  The max throughput while keeping the request drop ratio under a custom threshold (e.g., 1%).



The quality of service is often defined in Service Level Agreements (SLAs) between service providers and customers as the binding contracts, which typically specify rough performance numbers below which the service is considered unavailable or unacceptable. Networking SLA requirements are typically phrased in terms of availability, bandwidth, loss rate, latency, and jitter constraints [21]. In a computing utility environment like Océano [22], availability, response time, server load, assigned resources, and output bandwidth are proposed as requirements and goals in SLAs. Such SLAs are, in general, not very precise with respect to server performance. It is simply too difficult to separate network delay from server delay. Moreover, delays depend on the service time, which, in turn, depends on the type and complexity of the outsourced service. Thus, delay is one of the metrics which a service provider cannot commit except on a very basic level.

For example, generating a page by executing a perl script or other dynamic content-creation mechanism is generally more resource-demanding and, therefore, slower than static pages. A large and infrequently-accessed object naturally results in a long service time. Hence, metrics which depend on the actual service itself seem inadequate for gauging service quality in a real system. On the other hand, loss-rate and failure-based metrics are easier to validate and are much less ambiguous. Therefore, service quality can be expressed in terms of loss-related quantities. Since we assume that requests are encapsulated in reliable TCP connections, the only possibility of dropping requests is encountering full queues at a busy server.

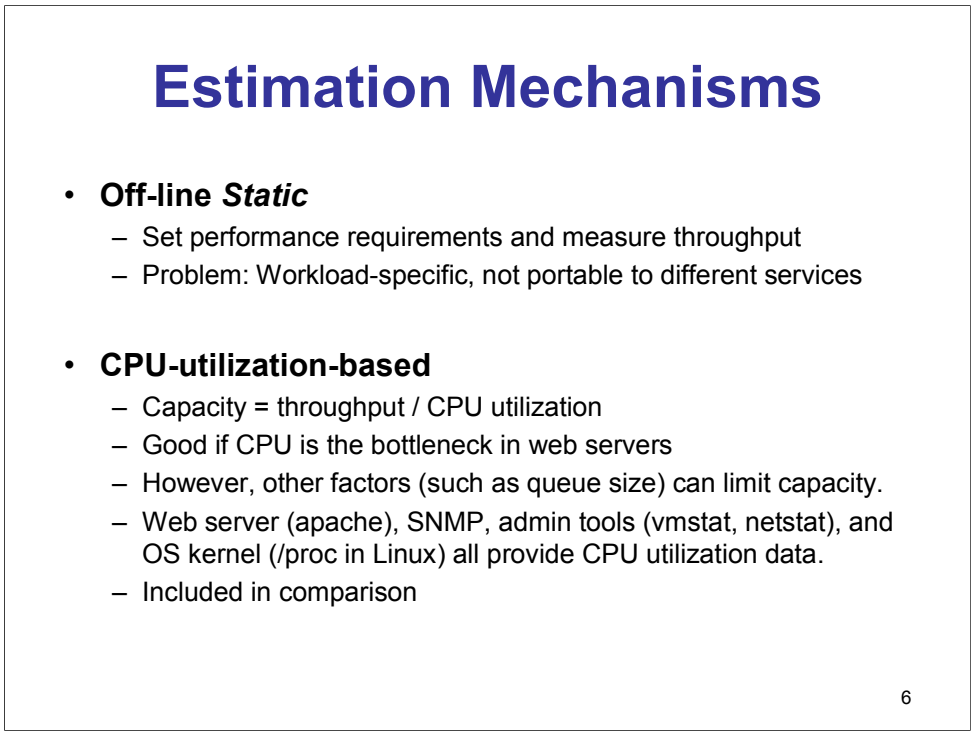
We conducted an experiment to show the relationship between the queue length and service quality. We increase request rate linearly over time with exponentially-distributed client inter-arrival times. All requests are made to the same document. As shown in the upper figure, as request rate grows with time, the queue length starts to increase at around 700 reqs/sec and continues to grow until the queue is full. We also monitor the number of incoming requests that failed to enqueue. As the request rate increases independently of actual throughput, the request drop ratio also increases with time. Dropped requests will be retransmitted and the amount of time waiting for retransmission(s) is also figured in the client-perceived response time, which is plotted in the lower figure on this page.

Since the clients and servers are connected with Fast Ethernet links, the packet-delivery time is usually very short. However, as requests are dropped by the server, the gap between the response times of requests that experienced no drop and one or more drops is obvious. The gaps actually map to different TCP timeout values of 3, 6, 12, 24, and 48 seconds. With users expecting prompt response from servers, especially web servers, even a 3-second delay will deteriorate client-perceived service quality seriously in spite of whatever service time the request may take. Experiencing timeouts of 6 seconds or more (*i.e.*, servers can only respond at least 9 seconds after the request had first arrived at the server) can make users lose patience and leave [12].

Based on the above observations, we define the **request drop ratio** as:
The request drop ratio is measured as the number of TCP connections failed to be established divided by the number of connection attempts, where each retry is counted as a separate attempt.

With the definition of request drop ratio, we then define the **capacity** of a server system as:
The maximum number of requests a server can process within 1 second while keeping the request drop ratio under a certain threshold.

We choose the threshold of request drop ratio as 1% in this paper. We believe that a 1% request drop ratio is a reasonable choice, since statistically only less than 0.01% of request arrivals will experience a delay of 9 seconds or more.



Off-line Static Estimation

Server capacity can be estimated in many different ways. Benchmark programs are commonly used to measure a system's ability to handle requests [5,6,7]. Some of them provide the maximum throughput, and others generate fixed workload and report performance metrics such as request-acceptance ratio and response time. For the latter type of benchmarks, users can set a performance requirement and repeat the program with heavier workloads until the result violates the performance requirements. Although benchmark programs can be used to estimate server capacity, they are also designed with a specific workload model in mind, such as server functions, user access patterns, content popularity distributions, and so on. However, with new service components introduced frequently and workload shifts during the course of a day, the fixed benchmark model and parameters cannot capture these dynamics. Consequently, benchmark programs are good only for comparing hardware performance in a controlled environment, but not for on-line capacity estimation. With an inaccurate estimation by benchmark programs, one may over-estimate or under-estimate the actual capacity of the server cluster, depending on the degree of the benchmark's deviation from the real workload.

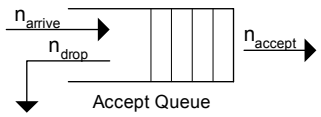
CPU-utilization-based Estimation

Another commonly-used method is to log system utilization and estimate achievable maximum throughput from it. System utilization usually includes CPU and network utilizations, disk activities, etc. Inside a data center, network bandwidth is abundant so it is usually not the bottleneck. Previous research showed that the popularity of web objects follows Zipf's law [4]. Therefore, web servers are readily equipped with sufficient memory, and hence, most objects can be cached in memory, thus eliminating most disk accesses. Besides, remaining disk accesses are made via a high-speed Storage Area Network (SAN). Hence, CPU becomes the most probable bottleneck in web servers. Therefore, one may want to estimate capacity by dividing current throughput by CPU utilization, in order to assess the number of requests a fully-utilized CPU can serve.

This method is simple and can adapt to changes in the workload model. For example, when the proportion of dynamic content requests increases in the incoming workload, estimation based on CPU utilization can capture the change as an increase in average CPU demand and then adjust the capacity estimation accordingly. However, servers have only a limited queuing capacity to hold transient incoming requests. This becomes a problem when utilization approaches 100%, because the very long predicted queue will not fit within the system-imposed queuing limits. Therefore, utilization-based estimation typically over-estimates server capacity by an unknown, workload-dependent factor.

Queue Monitoring for Capacity Estimation

- **Avoiding SYN retransmissions is key to dynamic cluster sizing**
  - Need to predict when retransmissions become likely, i.e., when capacity is too limited



- **Estimate time needed to fill the queue**

$$t_{fill} = \begin{cases} K / (\lambda_{\max} - c) & \text{if } \lambda_{\max} > c \\ \text{undefined} & \text{otherwise} \end{cases} \quad \text{(Eq. 1)}$$

- **Estimate avg queue length that causes overflows**

$$\ell_{th} = \begin{cases} \dfrac{K \cdot \lambda_{\max} \cdot t_{fill}}{2(c(T - t_{fill}) + \lambda_{\max} \cdot t_{fill})} & \text{if } t_{fill} \le T \\ \dfrac{[K - (\lambda_{\max} - c)T] + K}{2} & \text{if } t_{fill} > T \end{cases} \quad \text{(Eq. 2)}$$

- **Capacity is estimated by**

$$c' = \begin{cases} \mu & \text{if } \varepsilon \ge \varepsilon_{th} \\ c \cdot \alpha + \mu \cdot (1 - \alpha) & \text{if } \varepsilon < \varepsilon_{th} \text{ and } \begin{cases} \min(\bar{\ell}, \hat{\ell}) \ge \ell_{th} \wedge c > \mu, \text{or} \\ \max(\bar{\ell}, \hat{\ell}) < \ell_{th} \wedge c < \mu \end{cases} \\ c & \text{otherwise} \end{cases} \quad \text{(Eq. 3)}$$

7

As the cost of having requests dropped from the queue is high and our definition of capacity directly depends on the request drop ratio, we derive capacity estimates from the queue itself. We assume that the queue can hold up to K connections and is sampled every T seconds. In each sampling period, we count the number of enqueuing attempts n_{arrive}, the number of connections accepted by server processes n_{accept}, and also the number of connections dropped n_{drop}. The time index t of each sampled value and its derivatives are omitted in the text for clarity. With these numbers, we know that queue length is increased by $\Delta\ell = n_{arrive} - (n_{accept} + n_{drop})$ at the end of this sampling period. Instead of having an initial queue length and keeping track of queue length by adding $\Delta\ell$, we decided to have the average queue length avg(ℓ) during each sample period as it also captures the bursty arrivals (and also burst acceptances) in web traffic. We also define arrival rate λ to be n_{arrive}/T and service rate μ to be n_{accept}/T. The current request drop ratio ε is defined as n_{drop} / n_{arrive} which will be used to adapt the capacity estimate c, and is also used to verify the constraint in the definition of capacity.

As previously shown, a queue with an average queue length less than a certain threshold will never cause any requests to be dropped. The value of this threshold depends on workload characteristics, such as arrival rate and burstiness. Since an accurate workload model is not available, we first estimate the threshold from the queue monitoring result.

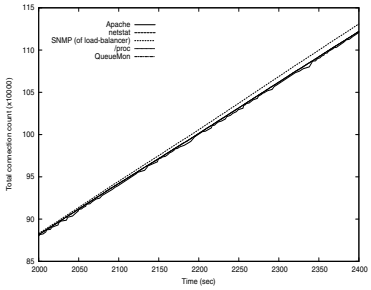
The difference between peak arrival rate, λ_{max}, and current capacity estimation, c, represents the maximal mismatch between arrival rate and service rate. Also, it is the fastest rate that connection requests accumulate in the queue. The time needed to fill up the queue can be calculated as shown in the slide. As long as t_{fill} is defined (t_{fill} is undefined only if the capacity estimate is also the peak arrival rate), we can estimate the threshold, which is the minimal average queue length in a sample period that could lead to an overflow. When the time needed is shorter than a sample period, the queue can be empty at the start of a sample period. Otherwise, there must be some connections queued up. The threshold calculation is shown in the slide. After the threshold has been determined, we predict future queue lengths by assuming that average queue length grows (or shrinks) linearly with time. If current average queue length grows at a rate of ℓ' connections/sec, and a $t_{forcast}$-second forecast, the predicted average queue length ℓ is therefore est(ℓ) = avg(ℓ) + ℓ' × $t_{forcast}$.

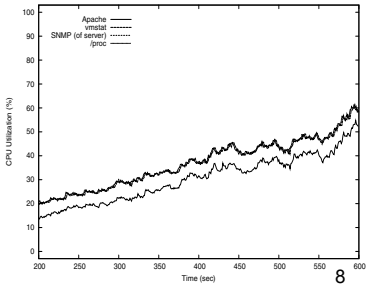
We adapt the capacity estimate by combining predicted average queue length and the threshold derived above. In Eq. 3, ε_{th} is the specified request drop ratio constraint, c' is the new capacity estimate and α is a constant to smooth out changes in the estimate. The first case clamps down the capacity estimate to current service rate if the request drop ratio constraint is violated. Otherwise, if the queue is predicted to drop requests and the current estimate is too optimistic, we adjust the estimate toward a more conservative value. Similarly, if a higher service rate does not make the average queue length growth beyond the threshold, the estimate can be relaxed. If none of these conditions holds, the original estimate is retained.

The insight behind this queue monitoring is that the average queue length grows in spite of the type of bottleneck resource. As long as there is a speed mismatch between request arrival and service rates, the queue length will reflect it immediately. The parameters in this mechanism, including K, T, ε_{th}, and $t_{forcast}$, are all tunable to match system specification and the request drop ratio constraints. After a certain period in the adaptation phase, an accurate capacity estimate can be obtained.

Measurement Accuracy

- **Small discrepancies on connection count. Linux and apache report smaller values.**
- **Linux kernel is less accurate in handling `tcpPassiveOpens` SNMP counter.**
- **Apache only counts CPU time used by httpd processes.**
- **Lessons learned: measurement agent should be light-weight and scheduled at high-priority.**





Before estimating server capacity, we first compare the accuracy of different measurement implementations. We set up all of them to simultaneously measure one running server at a 1Hz sampling frequency and compare the thus-obtained results. The server is fed with increasing loads, which are based on the SURGE [5] workload model with an increasing number of users, so as to assess the accuracy of measuring both a normal server and an overloaded server. Although running multiple measurement methods simultaneously increases the overhead of the server under test, this is the only way to make direct comparison of all of them.

To estimate capacity, one of the most important parameters to measure is the total number of connections, which is also the total number of requests in our model. Before the server became saturated, all five methods yield exactly the same result (at the lower left corner of the upper figure on this slide). However, as the server became saturated, it took more than 1 second, which is our sampling period, to retrieve the sample from Apache, causing zigzags in this figure.

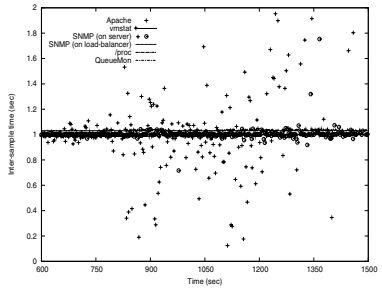
Moreover, the total number of connections obtained by SNMP grows faster than others after the server is saturated. This inconsistency is due to the interpretation of SNMP specification in the Linux kernel. In kernel version 2.4, when the accept queue is full, the kernel still responds to SYN packets with SYN+ACK packets. The load-balancer regards each three-way handshake as the creation of a connection, and hence, increases the SNMP `TotalConnections` counter by one as soon as the client responds with an ACK packet. However, the Linux kernel increases the `tcpPassiveOpens` counter only after the ACK packet is received *and* there is room in the accept queue. Therefore, as the accept queue is full for a non-negligible period of time, many connections are not be able to move to the accept queue even after a few retries and the inconsistency between counters arises. The way the Linux kernel handles the counter is less accurate according to the definition in RFC 1213 (also as STD 17) [23]: *the number of times TCP connections have made direct transitions to the SYN-RCVD state from the LISTEN state*.

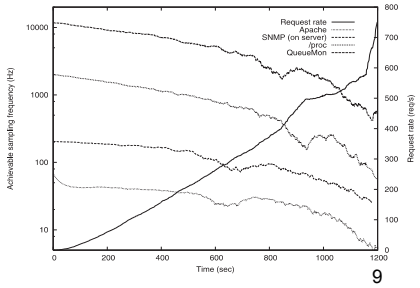
We also compare the CPU utilization, which we define as the fraction of time the CPU is not idle. The user-space program `vmstat` represents CPU utilization in percentages while all other methods express CPU utilization in total number of jiffies (of 10ms each) spent in each mode. With the actual sampling time of each point, we convert all CPU performance counters to percentages.

The CPU utilizations obtained by using different methods are plotted in the lower figure. At first, it was impossible to even retrieve CPU performance counters via SNMP when the server is saturated, because the SNMP agent cannot get CPU time slot from OS scheduler; this problem was solved by promoting the SNMP agent's priority. The result shows that CPU utilization grows with request rate (which increases linearly with time). Compared with other three approaches, Apache consistently under-estimates CPU utilization as it only counts the CPU time used by itself.

Measurement Overhead

- **Time-stamping after retrieve measurement result is more accurate.**
- **Measurement processes must be promoted to higher priority.**
- **Inter-sample time of measurement results from Apache deviates a lot.**
- **Queue monitoring is a light-weight process. More than 500 samples can be obtained under heavy load.**





9

As the server under test becomes saturated, we also noticed that some of the measurement methods cannot keep up with the sampling frequency (1 Hz). The unpredictability of sampling interval is due to the OS scheduler. We tagged each sample with the current system time when the sample is taken. Compared with "before-sampling" time, we found "after-sampling" time is closer to the time when measurement entities (*e.g.*, Apache server, SNMP agent, etc.) actually processed the counter-retrieval request. The inter-sample intervals of these measurement methods are plotted in the upper figure.

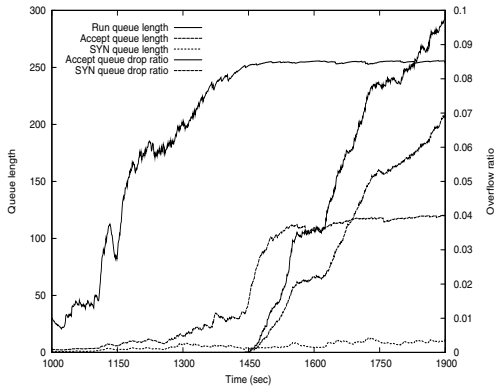
Similar to the SNMP agent, we also promote the priority of other measurement-related processes. As retrieving measurement results from the `/proc` filesystem does not create any new process, once the measurement-retrieval process is scheduled, the result collection can be done immediately. If we did not adjust the priority, there will be a fluctuating delay between 0.8s to 1.2s. The load-balancer also responds immediately to SNMP queries in this scenario.

Although the SNMP agent is running at a higher priority, there are still some inter-sample time swing from 1s. As we use the command `vmstat 1` to trigger user-space counter logging, we can also see that the inter-sample time is more likely to be 1.01s due to the timer accuracy of `sleep()`. The only method suffered from frequent deviations around a fixed sampling delay is the Apache status retrieving, as it must be scheduled with other Apache processes.

Using the same evaluation setup as above, we also compare the overheads of the measurement methods. Instead of periodically sampling the server, we execute one method at a time and measure the system continuously. The maximum sampling frequency of each method is plotted in the lower figure. As the figure shows, when there is little load on the server, four of the five measurement methods can acquire samples at a rate of 50Hz or much higher, with the exception of user-space measurement tools `vmstat` which has a limit of 1Hz sampling frequency by design. However, as request rate increases linearly with time, the maximum achievable sampling frequency drops exponentially. A high-overhead measurement method like Apache can only achieve a sampling rate of 5Hz. On the other hand, we can still retrieve queue length measurement results at 500Hz. Although we do not require a high sampling frequency to estimate capacity, it shows that kernel instrumentation is a low-overhead measurement method.

Queuing Behavior

- **Queue length grows**
  - Run queue
  - Accept queue
  - SYN queue
- **Pitfall: in Linux, SYN+ACK packets are dropped if both accept queue and outgoing packet scheduler queue are full. Increasing default outgoing queue size helps.**



10

Before using the queue-length monitoring to estimate capacity, we first examine how various queues in a server system interact with each other and when incoming packets are dropped.

As the figure shows, as request rate increases, the run queue length first reflects the increasing workload. After we reached the limit of the allowable number (256) of httpd processes, incoming requests start to be accumulated in the accept queue. No request will be dropped because of the httpd processes count limit. As the accept queue length grows, we observed that SYN packets were dropped even without any queue filled up. This was due to the queue size limit of an outgoing packet scheduler. Although the packet scheduler normally does not drop any packet, the SYN+ACK packet is an exception in Linux. If the SYN+ACK packet cannot be scheduled to send, the Linux kernel does not retry before dropping it. As the default packet scheduler has a queue of only 100 packets, bursty traffic can easily fill up the queue before the device driver has a chance to drain some of the packets. With the limit increased to 1000, no more SYN packets were dropped at this stage.

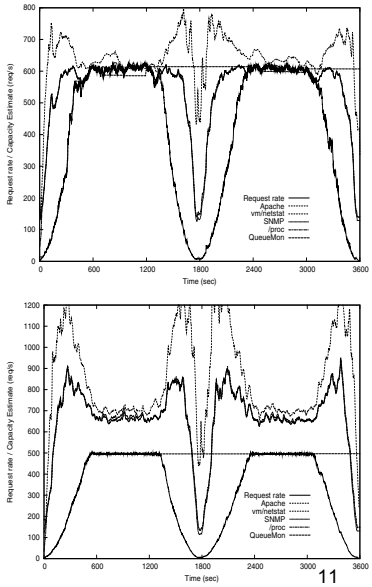
Before the average accept queue length approaches its limit (128), the instantaneous accept queue length, which is highly fluctuating and we choose not to plot in the figure for clarity, has reached the limit every now and then. A full accept queue not only blocks established TCP connections out of the queue (when the ACK packet is received from the client), but also the first SYN packet from the client will be dropped. Therefore, we can see that both accept queue and SYN queue drop ratios increase dramatically; almost 10% of new connections cannot be established if the accept queue is full for a few minutes. Since many SYN packets are rejected, the SYN queue length grows at a very moderate pace and is far from its limit (1024).

Capacity Estimation -- Baseline

Using the measurement results, we apply all three capacity estimation mechanisms to the server under test. To determine the actual capacity of the server, we feed linearly-increasing SURGE workloads to the server. We calculate the drop ratio over a one-minute window and determine the capacity of the server as the average throughput during that minute. The process is repeated 10 times and the average number, 595 requests per second, is determined as the server capacity under this workload.

Capacity Estimation

- **Capacity under SURGE workload: 595 requests/sec**
- **CPU-utilization-based estimates are good given CPU-bound workload and medium loading. Queue-monitoring gives most stable result (586 req/s)**
- **A 0.5s wait for backend services for each request shifts the workload to I/O bound. Only queue-monitoring produces useful and good estimates.**



Instead of using the simple linearly-increasing workload, we create a sinusoid-like workload, where the number of emulated users changes like a sine wave, to evaluate the capacity estimation mechanisms under fluctuating workloads. Due to the nature of SURGE user model, the workload we used here is still bursty microscopically. When the workload is CPU-bound, CPU-utilization-based mechanism can effectively approximate system capacity. While this method is simple, the quality of its inputs, *i.e.*, measured data, can seriously affect its prediction. To this end, we compare the accuracy of four measurement mechanisms that can be used as input to any capacity-estimation method, including the simple one mentioned above. The result of capacity estimation by queue monitoring is also included for the comparison purpose.

The upper plot shows how these mechanisms perform relative to each other. Use of Apache's status yields a poor estimate of server capacity. It typically over-estimates load when the workload is low. This is due mainly to various layers of indirection and inaccuracies introduced by each of them, *e.g.*, CPU time used by background processes is not taken into account. Estimates by using the output of `vmstat` and `netstat` commands, using SNMP counters and measurements in the Linux's `/proc` filesystem are all very close to each other. They all produce a reasonable estimate when there is a medium-level load on the server, *e.g.*, during 200s and 400s in this experiment. Once the server is overloaded, the estimate essentially equals current throughput while a large number of incoming packets are dropped. The capacity estimated by the queue-monitoring scheme at first grows with request rate. As soon as the server gets overloaded, our scheme actually estimates a capacity lower than current throughput, and the estimate, 586 reqs/s, is actually very close to the one we obtained by static estimation, 595 reqs/s. After the overloaded situation disappears, it yields an estimate of 614 reqs/s as the queue length drops to a safe region. Most importantly, in between the two overloaded periods (from 1200s to 1800s), since there are no new triggers to adapt capacity estimation, it keeps the current estimate. A stable estimate is very important for making control decisions.

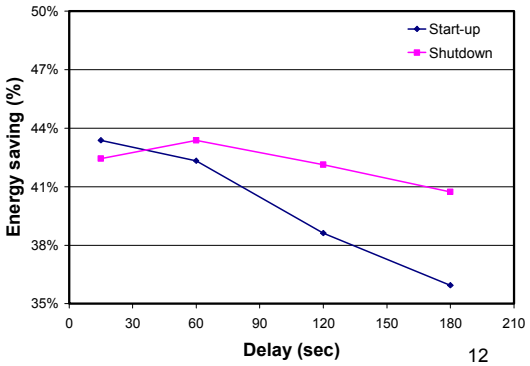
The CPU is not the only potential bottleneck of a system, albeit the most trivial to measure. Estimating server capacity while a resource other than CPU is the bottleneck, is much more difficult because I/O utilization is fuzzier than CPU utilization. To simulate an I/O bound workload, before the completion of each request, we artificially insert a 0.5-second sleep to simulate I/O waiting time, such as the time a front-end web server waiting for back-end database committing transactions, which is common in multi-tier server designs. Although a fixed I/O time is unrealistic, it suggests that a batch of requests that result in some I/O delay can easily change the workload from CPU-bound to I/O-bound.

Estimates of server capacity when an I/O-bound workload is present are plotted in the lower figure. Obviously, all estimations based on CPU utilizations are too optimistic by a large margin. Only the queue-monitoring scheme provides a stable output regardless of the shift of the bottleneck. Therefore, we conclude that our server capacity estimation based on queue monitoring is adaptive and can handle a wide range of workloads.

Energy Conservation

- **With capacity estimates, trivial cluster size controllers can be built.**
- **Static estimates are dangerous with increasing workload requirements.**
- **Even prior knowledge of workload didn't perform better than queue measurement.**
- **Request drop ratio is well under controlled.**
- **Long start-up and shutdown delay reduced the benefits of adapting.**

Resource demand per request	"Standard"	Decreased	Increased
Static capacity estimation	38%	35%	failure
Dynamic capacity estimation	38%	44%	31%



12

Traditionally, sizing a server cluster is based on a fixed schedule, which is usually derived by observing typical daily demand fluctuations from logs. A fixed number of servers are activated to provide services regardless of current utilization or service quality. The time granularity of a schedule depends on how fast the workload changes and it is usually in the order of one hour. Although this scheme is simple, it does not provide any service-quality guarantee, and the cluster is usually under-utilized.

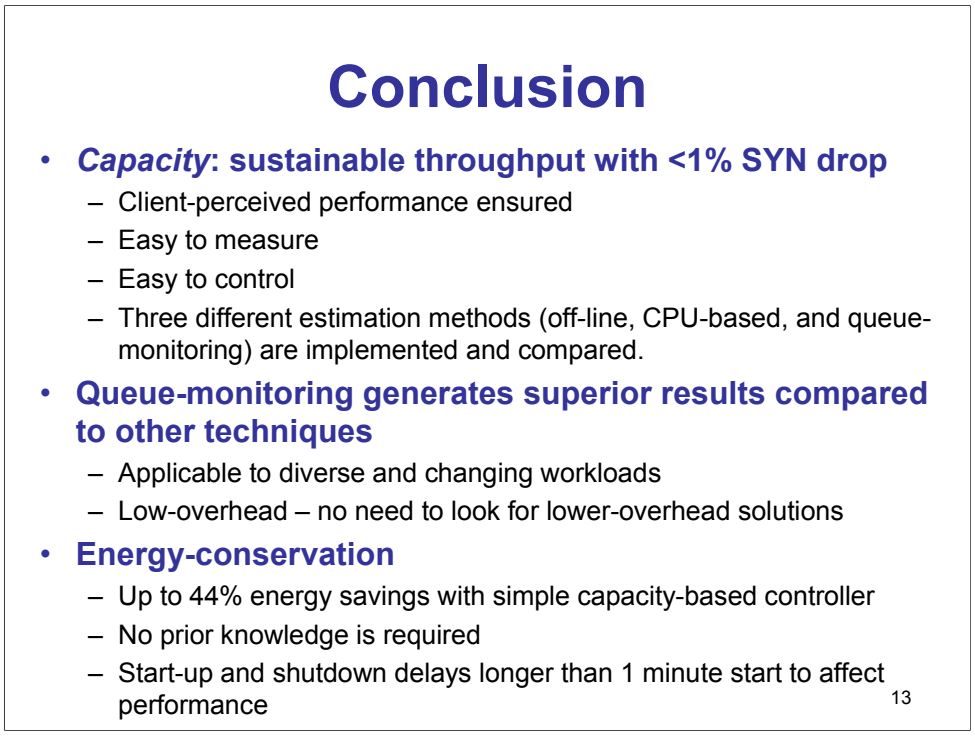
Another commonly-used practice is to benchmark each server's capacity and then use controllers to adjust the number of active servers accordingly. We call this *static controller*. Safety margins are also frequently used to avoid capacity over-estimation even if the utilization is lowered at the same time. In addition, since a benchmark workload usually differs from actual workload and the workload itself also changes during the course of a day, static controllers cannot achieve the highest utilization.

With on-line capacity estimation, a controller as simple as a static controller can still be utilized and is more effective. We call this *dynamic controller*. With more accurate capacity estimation, the timing to adjust cluster size can be more precise, and thus, more energy can be saved without sacrificing service quality. In what follows, percentage energy-savings are compared with an always-on server clusters. While much energy is wasted for very little productivity, it is still the most common practice today.

We first compare static controllers with dynamic controllers by using the same workload model that was used to determine server capacity statically. Since the workload is identical, both controllers perform almost the same. Both controllers achieve 38% of energy savings and merely 0.001% of total requests experienced accept queue overflows and penalties of TCP timeouts. Although the performance is similar, the dynamic controller does not require any prior knowledge of the workload and is, therefore, easier to deploy. Also, for comparison with fixed-schedule controllers, we have also built a capacity schedule based on the user population profile. If the schedule has a time slice of 30 minutes, 31% of energy can be saved. If the size of the time slice increases to two hours, energy savings are reduced to 25%.

When per-request resource demand is reduced by halving the object size, the static controller still turns on new servers when the throughput reaches 595 reqs/s for each server. However, active servers did not reach full utilization at that moment, and hence, turning on new servers would consume extra energy. On the other hand, the dynamic controller estimated each server to be able to handle about 750 reqs/s, so it turned on servers later and turned off servers earlier. At the end of the experiment, static controller saves 35% of energy while dynamic controller saves 44% of energy. The service quality is well maintained as there is less than 0.001% of total requests experienced accept queue overflows. This demonstrates the ability of capacity estimation to correctly recognize the decreased resource demand in each request, thus saving more energy.

(continue on next page)



(continued from previous page)

To simulate increased per-request resource demand, we replace each static document with a Server-Side Include (SSI) dynamic document. When the static controller is applied to this scenario, the result becomes totally unacceptable as the server saturated before reaching its outdated estimated capacity. Therefore, no new server is turned on to share the workload. A large number of requests failed to go through. On the other hand, the dynamic controller does not suffer from the same problem. As the active server gets saturated, new servers are activated and the quality of service is maintained. The result shows that 31% of energy is saved in this experiment and only 0.002% of total requests experienced queue overflows.

We also simulate the effect of different start-up and shutdown delays. While keeping shutdown delay at 60s, we simulate various start-up delays between 15s and 180s. As the result shows, a delay longer than 60s starts to affect the performance of energy conservation. Similarly, long shutdown delays also reduce the benefits but at a less extent.

Conclusion

In this paper, with a symmetric server cluster model and empirical evidence that shows the impact of dropping TCP handshake packets, we defined the capacity of Internet servers as sustainable throughput with a low (e.g., 1%) SYN request drop ratio. This definition ensures that the client-perceived performance remains acceptable as long as enough capacity is allocated. To estimate capacity, we proposed a new mechanism based on listen queue monitoring and demonstrated that it is superior to utilization-based estimation approaches. Also, different means of collecting system performance measurements are all implemented, tested, and evaluated. We found that pitfalls exist in different measurement methods, such as the priority of measurement-collection processes and the actual meaning of each counter value. More importantly, our queue-monitoring mechanism can obtain a good estimate of server capacity irrespective of whether workload is CPU-bound or I/O-bound.

We applied the server capacity estimation scheme to adapt the size of a web server cluster. By turning on and off servers to match actual workload, energy consumption can be reduced significantly and utilization (and therefore profits) of a data center can be maximized. Without requiring prior knowledge of workload or cost, we achieved 31 to 44% of energy savings under various workloads, thus affirmatively supporting the value of a good capacity-estimation mechanism.

References

[1] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In Proc. of the 18th ACM SOSP, 2001.

[2] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. In Proc. of the ACM SIGMETRICS, 1996.

[3] M. Arlitt and T. Jin. Workload characterization of the 1998 World Cup web site. HP Labs Tech Report HPL-1999-35(R.1), 1999.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. In Proc. of IEEE INFOCOM, 1999.

[5] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In Proc. of ACM SIGMETRICS/PERFORMANCE, 1998.

[6] D. Mosberger and T. Jin. httperf: a tool for measuring web server performance. In Proc. of the 1st Workshop on Internet Server Performance, 1998.

[7] SPEC. SPECweb99 Benchmark. http://www.specbench.org/osg/web99/, 1999.

[8] J. Judge, H. Beadle, and J. Chicharo. Sampling HTTP response packets for prediction of web traffic volume statistics. In Proc. of Globecom, 1998.

[9] W. Shi, R. Wright, E. Collins, and V. Karamcheti. Workload characterization of a personalized web site and its implications for dynamic content caching. In Proc. of the 7th Int'l Workshop on Web Caching and Content Distribution, 2002.

[10] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: a mechanism for resource management in cluster-based network servers. In Proc. of the ACM SIGMETRICS, 2000.

[11] G. Banga, P. Druschel, and J. Mogul. Resource containers: a new facility for resource management in server systems. In Proc. of the 3rd OSDI, 1999.

[12] A. Bouch, A. Kuchinsky, and N. Bhatti. Quality is in the eye of the beholder: meeting users' requirements for Internet quality of service. HP Labs Tech Report HPL-2000-4, 2000.

[13] R. Rajamony and M. Elnozahy. Measuring client-perceived response times on the WWW. In Proc. of the 3rd USENIX USITS, 2001.

[14] D. Olshefski, J. Nieh, and D. Agrawal. Inferring client response time at the Web server. In Proc. of the ACM SIGMETRICS, 2002.

[15] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In Proc. of OSDI, 1994.

[16] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, and R. Rajamony. The case for power management in Web servers. In Power-Aware Computing (Robert Graybill and Rami Melhem, editors). Kluwer/Plenum series in Computer Science, 2002.

[17] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In Proc. of ISPASS, 2003.

[18] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. Tech Report DCS-TR-440, Dept. of Computer Science, Rutgers Univ., 2001.

[19] M. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for Web servers. In Proc. of the 4th USENIX USITS, 2003.

[20] T. Abdelzaher and N. Bhatti. Adaptive content delivery for Web server QoS. In Proc. of Int'l Workshop on QoS, 1999.

[21] J. Martin and A. Nilsson. On service level agreements for IP networks. In Proc. of INFOCOM, 2002.

[22] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Oceano -- SLA based management of a computing utility. In Proc. of IFIP/IEEE IM, 2001.

[23] K. McCloghrie and M. Rose. Management information base for network management of TCP/IP-based internets. RFC 1213, 1991.