

# Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems

Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath

Department of Computer Science  
Rutgers University  
Piscataway, NJ 08854-8019

{edpin, ricardob, vinicio, taliver}@cs.rutgers.edu

Technical Report DCS-TR-440, May 2001

## Abstract

In this paper we address power conservation for clusters of workstations or PCs, such as those that support a large number of research/teaching organizations and most Internet companies. Our approach is to develop systems that dynamically turn cluster nodes on – to be able to handle the load imposed on the system efficiently – and off – to save power under lighter load. The key component of our systems is an algorithm that makes load balancing and unbalancing decisions by considering both the total load imposed on the cluster and the power and performance implications of turning nodes off. The algorithm is implemented in three different ways: (1) at the application level for a cluster-based, locality-conscious network server; (2) at the operating system level for an operating system for clustered cycle servers; and (3) by application/operating system negotiation, again using the cluster-oriented operating system and a negotiation API. Our experimental results are very favorable, showing that our systems conserve both power and energy in comparison to traditional systems, which do not adapt their configurations.

## 1 Introduction

Power and energy consumption have always been critical concerns for laptop and hand-held devices, as these devices generally run on batteries and are not connected to the electrical power grid. The amount of power consumed by such a device determines the required battery capacity and the temperature of the device components, whereas the energy consumed ultimately determines the battery life. As a result, a tremendous amount of research has been directed towards low-power and low-energy design and conservation (e.g. [16, 33, 22, 11, 13]).

In contrast with this line of research, in this paper we propose a *new research direction*, namely power and energy conservation for clusters of workstations or PCs, such as those that support a large number of research and teaching organizations and most Internet companies. Power consumption is an important concern in the context of clusters as it directly influences the cooling requirements of each cluster node and of the overall cluster ensemble. Our own clus-

ter of high-performance workstations is an example of these per-node requirements. Each of our single-processor, dual-disk nodes uses 7 fans. Each node consumes approximately 78 Watts during normal operation under our group's cluster workloads. Interestingly, a non-trivial fraction of this power consumption, approximately 9 Watts (or 12%), is consumed by the fans themselves. Higher performance nodes consume an even larger fraction of their total power consumption just in per-node cooling hardware.

The cooling requirements of cluster ensembles can also be substantial. In fact, a medium to large number of high-performance nodes racked closely together in the same room, as is usually the case with clusters, requires a significant investment in cooling, both in terms of sophisticated racks and heavy-duty air conditioning systems. At each Google site, for instance, there are 40 racks, each of which with 80 PCs, for a total power consumption of nothing less than 180 kWatts! Cooling such a large installation is an expensive challenge. Even worse, *the investment in cooling systems is becoming an ever larger fraction of the cost of clusters, as off-the-shelf hardware prices constantly decrease*. Given a fixed budget, one dollar spent on air conditioning is one dollar not spent in hardware that can actually produce better service or higher throughput, such as cluster nodes, disks, etc.

Taking a broader perspective, the power requirements of clusters have become a major issue for several states, such as California, New York, and Massachusetts. The New York Times reported that the planned 46 data centers for the New York city metropolitan area alone have asked for a minimum draw of 500 megawatts of power, which is enough to power 500,000 households [29]! The inability of these states to meet such enormous power requirements has already resulted in data center projects being cancelled or delayed [29].

Besides power consumption, energy consumption is also important for large, high-performance clusters. Both the computational and the air conditioning infrastructures consume energy. This energy consumption is reflected in the electricity bill, which can be significant for a large cluster in a heavily air-conditioned room. In fact, the electricity bills for large cluster systems are likely to soar in certain states (especially California) in the near future, if charges per Wh indeed become a function of overall consumption. Research and teaching organizations supported by large clusters, in particular, may have

a hard time finding the funds for covering energy costs. Again, the news services provide anecdotal confirmation of how critical energy consumption is for large computer systems. An article from ComputerWorld [9] discusses last year’s surge in energy consumption in the heart of Silicon Valley and includes a quote from a spokesman for Silicon Valley Power to the effect that a single large data center can consume more energy than the largest manufacturing plant his company serves. A Google site, for instance, consumes no less than 130 MWh per month just with the cluster infrastructure.

Our approach to conserving power/energy is to develop systems that can leverage the widespread replication of resources in clusters. This paper presents our first steps in this new research direction. In particular, we develop systems that can dynamically turn cluster nodes on – to be able to handle the load imposed on the system efficiently – and off – to save power under lighter load. This research is inspired by previous work in cluster-wide load balancing (e.g. [2, 14, 24, 26, 8, 27, 4]). When performing load balancing, the goal is to evenly spread the work over the available cluster resources in such a way that idle nodes can be used and performance can be promoted. The inverse of the load balancing operation concentrates work in fewer nodes, idling other nodes that can be turned off. This *load concentration* or *unbalancing operation* saves the power consumed by the powered-down nodes, but can degrade the performance of the remaining nodes and potentially increase their power consumption. Thus, load concentration involves an interesting *performance vs. power tradeoff*. Our systems use load concentration as a first-class technique, rather than as a remedial technique like in systems that harvest idle workstations (e.g. [2, 24]) or as a management technique for manually excluding a cluster node.

The key component of our systems is an algorithm that makes load balancing and concentration decisions by considering both the total load imposed on the cluster and the power and performance of different cluster configurations. In more detail, the algorithm periodically considers whether nodes should be added to or removed from the cluster, based on the expected performance and power consumption that would result, and decides how the existing load should be re-distributed in case of a configuration change. To be able to understand the implications of our algorithm, we implemented it for two popular types of cluster-based systems: a locality-conscious network server and a load balancing distributed operating system (OS) for clustered cycle servers. The implementations were performed in three ways: (1) at the application level for the network server; (2) at the OS level for the cycle server; and (3) by application/OS negotiation for the cycle server, extending the load balancing OS with a simple negotiation API.

*Even though we target power conservation primarily, our experimental results show that our secondary goal of saving energy is achieved as well.* Our initial results show that our modified network server can reduce the total power consumption of the cluster by as much as 86% and the energy consumption by 43% in comparison to the original server running on a static cluster configuration with 8 nodes. The modified OS can reduce power consumption by as much as 86% for a synthetic workload, while attempting to keep performance degradation below 20%, again in comparison to the original system on a static 8-node cluster. The energy savings it accrues in this case is 32%. Finally, our application/OS interaction experiments on 4 cluster nodes show that we can save as much as 25% power and 29% energy, while respecting performance degradation directions provided by the applications.

As mentioned above, the previous work on power and energy is concentrated on laptop computers and embedded or hand-held devices. As far as we are aware, our work is the first to study power and energy conservation for clusters in detail. A very recent paper [7] also suggests this research direction, but does not include an evaluation of their proposed energy-conscious switch. In addition, most of the previous work (e.g. [11, 22, 7]) is focused on reducing consumption by moving resources to standby or idle states, which are states that are not as power-hungry as the active state, but still consume power. In this paper we take the extreme approach of turning whole nodes completely off. Our algorithm and systems apply novel strategies for distributing load in clusters, in particular they apply load concentration as a first-class technique. In terms of application/OS interaction, two previous papers [25, 13] have proposed such interaction for energy conservation. We propose yet another type of interaction.

Based on our experimental results, we believe that our algorithm and systems should be useful for organizations and companies that rely on large clusters of workstations or PCs, in that they can revert the dollars to be spent in air conditioning and electricity to dollars to be spent on additional computational hardware.

The remainder of this paper is organized as follows. Section 2 describes our cluster configuration and load distribution algorithm and its different implementations. Section 3 describes our experimental set-up and the methodology used. Section 4 discusses our experimental results. Section 5 discusses the related work. Finally, section 6 concludes the paper and mentions our future work.

## 2 The Cluster Configuration and Load Distribution Algorithm

### 2.1 Overview

**Power vs. performance.** We consider the tradeoff between power and two types of performance, namely throughput and execution time performance. Throughput is the key issue for systems such as modern network servers, in which the goal is to service as many requests as possible; the latency of each request at the server is usually a small fraction of the overall latency of wide-area client-server communication. Execution time is key for systems such as cycle servers, as users may object to significant delays in the execution of their jobs.

The cluster configuration and load distribution algorithm we propose is based on the overall trends shown in figure 1. The figure plots the behavior of power consumption and performance (in this particular case, throughput performance) for an imaginary fixed workload, as a function of the number of cluster nodes. The power consumed by a cluster decreases as we decrease the number of nodes, whereas throughput suffers beyond the point where resources achieve maximum utilization.

Our algorithm conceptually moves around the graph in figure 1, evaluating the “best” move at each time. More specifically, for each cluster configuration and currently offered load, the algorithm decides whether to add (turn on) or remove (turn off) nodes, according to the expected performance and power implications of the decision.

For simplicity, the algorithm assumes that the cluster is comprised of homogeneous machines. Furthermore, the algorithm assumes that the removal of a node does not cripple the file system. In practice this assumption is often verified, since the responsibility for serving file

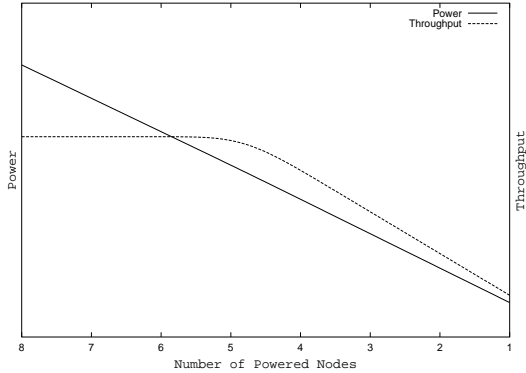


Figure 1: Power and throughput performance trends.

system requests can easily be assigned to one or more servers that do not belong to the cluster.

**Addition/removal decision.** In order to make node addition or removal decisions, the algorithm requires the ability to predict the performance and the power consumption of different cluster configurations. Exact power consumption predictions are not straightforward, although it might seem that way superficially. The problem is that it is difficult to predict the power to be consumed by a node after it receives some arbitrary load. Conversely, it is difficult to predict the power to be consumed by a node after some of its load is moved elsewhere.

Nevertheless, exact power consumption predictions are not really necessary for the algorithm to achieve its main goal, namely to conserve power. The reason for this is that each of our cluster nodes consumes approximately 70 Watts when idle and approximately 94 Watts when all resources, i.e. CPU, caches, memory, network interface, and disk, are stretched to the maximum. These measurements mean that: (a) there is a relatively small difference in power consumption between an idle node and a fully utilized node; and (b) the penalty for keeping a node powered on is high, even if it is idle. Thus, we find in practice that turning a node off always saves power, even if its load has to be moved to one or more other nodes. Thus, our algorithm always decreases the number of nodes, provided that the expected performance of applications is acceptable.

Performance predictions can also be difficult to make. We predict performance by keeping track of the *demand* for (not the utilization of) resources on all cluster nodes. With this information, our algorithm can estimate the performance degradation that can be imposed on a node when new load is sent to it. There is a caveat here, though. A degradation prediction is made based on past resource demand history of the load to be moved, so the prediction does not consider possible demand changes over time. In particular, the initial settle-down period during which the caches are warmed up with the new load is disregarded; we are more interested in steady-state performance.

A throughput prediction can easily be made based on the resource demand information. To see how this works, let us consider the throughput of a cluster-based network server. Suppose a scenario with 3 cluster nodes, each of which with demands for disk of 80%, 30%, and 20% of their nominal bandwidth. By adding up all of these disk demands (and disregarding other resources to simplify the example), we find that the server could run with no throughput degradation on 2 nodes ( $130 < 200$ ) and with a 30% throughput degradation on 1 node ( $130 - 100 = 30$ ). Our algorithm should decide to remove

one of the nodes or even two nodes, if a 30% throughput degradation is acceptable.

Execution time predictions are much more complex, as they depend heavily on the specific characteristics of the applications and on the amount *and timing* of the demand imposed on the different resources. Therefore, we have to settle for optimistic execution time predictions based on the demand for resources. The predictions are optimistic because they assume that the use of resources is fully pipelined and overlapped. To see how this works, let us consider the execution time performance of applications running on a cluster of cycle servers. Suppose a scenario with 2 cluster nodes with demands for their disks of 80% and 40% of the disks' nominal bandwidth. Our optimistic prediction strategy says that these applications could run with a *minimum* 20% execution time degradation on 1 node ( $120 - 100 = 20$ ). Our algorithm should decide to remove one of the nodes, if a 20% execution time degradation is acceptable.

The acceptable performance degradation can be specified by the cluster administrator or by each application (i.e. user), as we discuss in the next section. Ideally, the algorithm could also try to guarantee a maximum performance degradation. This is clearly not possible for execution time performance, but is conceivable for throughput performance. However, even in the case of throughput, such a strong guarantee cannot be made, given that the load on the cluster may increase faster than the system can react to such increase. Rather, we use our performance degradation parameter to *trigger* actions that can reduce or eliminate any degradation.

**Load (re-)distribution decision.** After an addition or removal decision is made, the load may have to be re-distributed. If the decision is to add one or more nodes, the algorithm must determine what part of the current load should be sent to the added nodes. Obviously, the load to be migrated should come from nodes undergoing excessive demand for resources.

If the decision is to remove one or more nodes, the algorithm must determine which nodes should be removed and, if necessary, where to send the load currently assigned to the soon-to-be-removed nodes. Obviously, the algorithm should give preference to lightly loaded victim nodes and destination nodes that would not undergo excessive demand for resources after receiving the new load.

The details of how to select victim nodes and of how to migrate load around the cluster depends heavily on the system for which the algorithm is implemented, so we do not specify this aspect of the algorithm in detail. The next section describes different ways of handling load distribution.

**General form.** In its most general form, our algorithm can be described as in figure 2. Node removal is acceptable if the expected performance degradation to any application on any node is smaller than a certain threshold, *degrad*, and the time since the last reconfiguration is larger than another threshold, *elapse*. Node addition is necessary if the current degradation is at least *degrad* and the time since the last reconfiguration is larger than *elapse*.

## 2.2 Implementations

Our algorithm has been implemented with minor variations in 3 different environments: (1) at the application level for a locality-conscious network server that runs alone on a cluster; (2) at the OS level for an OS for clustered cycle servers; and (3) by application/OS negotiation, again using the cluster-oriented OS and a simple nego-

```

Periodically do
  if removal is acceptable
    choose nodes (victims) with low demand to be turned off
    if necessary, determine nodes to receive load of victims
      and ask victims to migrate their load out
    ask victims to turn themselves off
  else
    if addition is necessary
      turn on new nodes
    if necessary, determine load to be sent to added nodes and
      ask nodes to share their load with added nodes

```

Figure 2: Pseudo-code for cluster configuration and load distribution algorithm.

tiation API. We discuss each of these implementations below.

In all implementations, the algorithm is run by a master node (node 0), which is a regular node except that it receives periodic resource demand messages from all other nodes and it cannot be turned off. We chose centralized implementations of the algorithm due to their simplicity and the fact that load messages can be infrequent. For fault tolerance, a distributed implementation would be best, but that is beyond the scope of this paper.

Given that reconfiguration operations are time-consuming, the implementations of our algorithm are conservative and only remove or add a single node at a time. More aggressive implementations can be produced by simply changing a runtime parameter.

**Power-aware cluster-based network server.** We modified PRESS [5], a cluster-based, event-driven WWW server to implement our algorithm completely at the application level. The server is based on the observation that serving a request from any memory cache, even a remote cache, is substantially more efficient than serving it from a disk, even a local disk. Essentially, the server distributes HTTP requests across nodes based on cache locality and load balancing considerations, so that files are unlikely to be read from disk if there is a cached copy somewhere in the cluster. (Note that the HTTP request distribution algorithm performed by the server has nothing to do with the cluster configuration and load distribution algorithm we propose in this paper.)

Any node of the cluster can receive a client HTTP request, serve the request itself, or forward it to another node. In case of forwarding, the requested content is sent back to the client through the cluster node that initially accepted the request. To be able to intelligently distribute the HTTP requests it receives, each node needs locality and load information about all other nodes. Locality information takes the form of the names of the files that are brought into the caches, whereas load information is the number of open connections handled by the nodes. These data are disseminated throughout the cluster via periodic broadcasts. To avoid excessive overheads, internal cluster communication is implemented with the Virtual Interface Architecture and the frequency of broadcasts is controlled. Greater details about the server can be found in [5].

We implemented the cluster configuration and load distribution algorithm in the server making all nodes periodically inform the master node about their CPU, disk, and network interface demands. The CPU demand is computed by reading information from the `/proc` directory, whereas network and disk demands are computed based on internal server information. To smooth out short bursts of activity,

each of these demands is exponentially amortized over time using the following formula:  $\alpha \times old\_demand + (1 - \alpha) \times current\_demand$ . For our experiments,  $\alpha = 0.8$  and the interval between demand computations is 10 seconds. In case of the server, we are interested in throughput performance.

Note that at the application level it is impossible to determine the demand for network interface (due to buffering in the kernel) and CPU (due to the fact that the server is single-process) resources, so our server cannot deal with a throughput degradation (degrad) greater than 0%. We experiment with two values for `elapse`: 200 and 300 seconds.

With information from all nodes, the master runs the cluster configuration and load distribution algorithm described in the previous section. If a removal decision is made, the master determines the maximum demand for any resource at each node and picks the node with the lowest of the maximum demands as the victim. For the WWW server, it is not necessary to migrate load from a node to be excluded from the cluster. The load can be naturally redistributed among the remaining nodes, by the server's own HTTP request distribution algorithm and/or a load balancing front-end. Similarly, the addition of a new node to the cluster does not require migrating any load from other nodes to it. A node addition is deemed necessary if any resource of any node is more highly demanded than a threshold, 90% in our experiments. Setting the threshold at 90% rather than at 100% provides some slack to compensate for the time it takes for a node to be rebooted, approximately 100 seconds.

**Power-aware OS for clusters.** We modified Nomad [26], a Linux-based distributed OS for clusters of uni and/or multiprocessor cycle servers. For the purposes of this paper, the most important characteristics of the OS are that (a) it has a shared file system; (b) it starts each application on the most lightly loaded node of the cluster at the moment; and (c) it performs dynamic checkpointing and migration of whole applications (with all its processes and state, including open file descriptors, static libraries, data, stack, registers and the like) between cluster nodes to balance load. Resource demand is computed for each node in the OS, by checking the resource queues every second. Whenever the average CPU demand, the memory consumption, or the I/O demand by a node remains higher than a threshold for 10 seconds, the OS considers the node to be undergoing excessive demand and attempts to migrate some of its load out to a more lightly-loaded node with respect to the heavily demanded resource.

To avoid excessive migration activity, the migration of an application can only happen if a few conditions are verified. First, an application can only be migrated if it has already executed at least as long as the average cost of a migration at the current host node. Second, a node that has just migrated an application elsewhere will not migrate another one until a period of stabilization, currently set to 80 seconds, has elapsed. Third, no incoming migration will be accepted by a node that has been either the source or the destination of a migration during the stabilization period. Finally, the OS was designed for clustered cycle servers, i.e. time-shared execution of sequential applications on uniprocessor nodes and of parallel applications on multiprocessor nodes, so applications that do not conform to these restrictions cannot be migrated by the system. Greater details about the base OS can be found in [26].

Like for the WWW server, we implemented the cluster configuration and load distribution algorithm in the OS making all nodes pe-

riodically inform the master node about their CPU, memory, and I/O demands. The CPU demand and the memory consumption are computed by reading information from `/proc`, whereas I/O demands are determined by instrumenting read and write system calls and getting swap information from `/proc`. To smooth out short bursts of activity, again the demands are amortized using the same formula as before. For our experiments,  $\alpha = 0.8$  and the interval between demand computations is 1 second. In case of the OS implementation of our algorithm, we are interested in execution time performance. The `degrad` parameter in our experiments is 20%. We experiment with two values for `elapse`: 90 and 180 seconds.

With information from all nodes, the master can run our algorithm. If a removal decision is made, the master selects the nodes with the lowest demands for each resource as candidate victims. Obviously, the master never selects itself as a candidate victim. Unlike the WWW server, in the OS case the load of the victim must be migrated to other nodes, so the master selects the two nodes with the lightest load with respect to each resource (CPU, I/O and memory) and selects the source/destination pair that would lead to the lowest overall demand for resources. (Ideally, one would want a selection to be made from all possible combinations of nodes. This, however, would take time quadratic on the number of nodes, as opposed to constant time.) To simplify our prototype implementation, the destination node receives all applications that are running on the victim node. Any load imbalances are later corrected by the OS according to its load balancing policy.

In the modified OS, a node addition is deemed necessary when the execution time degradation is at least `degrad` and more than one application is responsible for the excessive resource demand. After a new node is turned on, the OS will start migrating applications to it, so that the load will be balanced again. Given that adding nodes takes a significant amount of time and that our system only adds one node at a time to the cluster, it might take a while before the demand for resources becomes acceptable again, after a long-lasting surge of activity.

**Application/OS negotiation for power.** To study opportunities for application/OS interactions in the context of power conservation, we modified the OS infrastructure we just discussed to allow applications to influence cluster configuration decisions. When considering the exclusion of a node, the OS determines the node (called source) to be excluded and the node (called destination) to receive the load of the excluded node. The OS then proposes the intended exclusion to all applications running on the source and destination nodes, telling them about their expected performance degradation. If even a single application rejects the degradation, the node cannot be excluded and nothing happens. In case all applications running on both source and destination nodes accept or simply do respond, then all applications running on the source node are migrated to the destination node and the now idle node (source) is turned off.

We modified the distributed OS to implement this type of negotiation. More specifically, an application can specify a handler to catch a signal sent by the OS which informs the application that the cluster reconfiguration is about to happen. The application, upon catching the signal, calls the OS to find out what is the execution time degradation it may suffer, if the cluster reconfiguration is allowed to take place. The application then decides whether or not it can tolerate such a degradation and either rejects the reconfiguration by calling a specific system call or accepts the reconfiguration by taking no action. The OS waits for replies from applications for a fixed time

```

/* Signal handler */
void signal_handler (int sig)
{
    int degradation;
    degradation = get_degradation(); /* system call */
    if (degradation > maximum_acceptable_degradation())
        dont_accept(); /* system call */
    /* else do nothing */
}

/* Sets signal handler */
signal(&signal_handler, SIGDEGRAD);

```

Figure 3: Signal-handling pseudo-code.

(currently set to 1 second) before making a final decision. (Note that no fixed delay is long enough to guarantee that all applications always have a chance to catch the signal. However, our 1-second delay should be sufficient in most scenarios. In fact, 100 CPU-intensive applications would be required for our current delay to be too short, since the quantum size is 10 milliseconds in this implementation, and the signal handler is the first code to run after a process acquires the CPU.)

Figure 3 illustrates the handler code to be included in applications. The function `maximum_acceptable_degradation()` is application-specific and might be fixed or dependent on application state, and thus change over time.

### 3 Methodology

To study the performance of our algorithm and systems, we performed experiments with a cluster of 8 PCs connected by a Fast Ethernet switch and a Gigaset switch. Each of the nodes contains an 800-MHz Pentium III processor, 512 MBytes of memory, two 7200 rpm disks (only one disk is actually used), and network interfaces for the switches. Shutting a node down takes approximately 45 seconds and bringing it back up takes approximately 100 seconds.

Our results compare the network sever and the two versions of the clustered cycle servers against the static configuration versions of these systems. *The static systems keep all their nodes powered on all the time, as is the common practice for network and cycle servers in the real world.*

Our power measurement and control infrastructure is depicted in figure 4. All machines are connected to a power strip that allows for remote control of the outlets. Machines can be turned on and off by sending commands to the IP address of the power strip. The total amount of power consumed by the cluster is then monitored by a multimeter connected to the power strip. The multimeter collects instantaneous power measurements 3-4 times per second and sends these measurement to another computer (not shown in the figure), which stores them in a log for later use. We obtain the power consumed by different cluster configurations by aligning the log and our systems' statistics. This alignment is simpler for us than for previous researchers [13] because the sampling frequency and the scale of the measurements are much coarser in our experiments. To compress these data and smooth out the curves, our figures show only 1 power measurement per as many as 9-10 seconds.

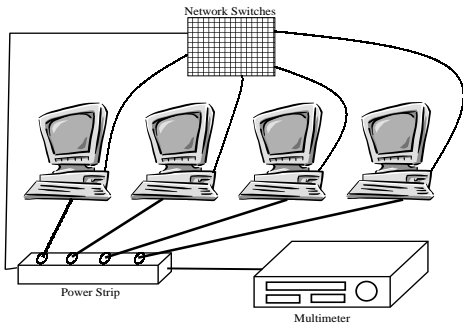


Figure 4: Power measurement and control infrastructure.

**WWW server experiments.** Besides the main cluster, we use another 12 Pentium-based machines to generate load for the modified WWW server. Simple clients running on these machines connect to the cluster using TCP over the Fast Ethernet switch. (The Gigaset switch is used in our WWW server experiments for intra-cluster communication.) For simplicity, we did not experiment with a front-end device that would hide the powering down of cluster nodes from clients. Instead, clients poll all servers every 10 seconds and can thus detect cluster reconfigurations and adapt their behavior accordingly. The clients send requests to the available nodes of the server in randomized fashion and reproduce the accesses made to the main server for the Computer Science Department of Rutgers University in the first 25 days of March 2000. To shorten the length of the experiments, we generate significant changes in offered demand in very little time.

**Distributed OS experiments.** The synthetic workload used for our modified OS experiments draws applications from a number of sources: all integer applications from the SPEC2000 benchmark, the Berkeley MPEG movie encoder, and two I/O benchmarks, IOcall and IOzone. IOcall is a benchmark to measure OS performance on I/O calls, especially file read system calls. IOzone is a file system benchmarking tool [19]; it generates and measures the performance of a variety of file operations. Applications are arbitrarily assigned to nodes and are run in arbitrary groups. Because the cluster size varies dynamically according to the resource demand imposed on it, we start with only one machine powered on (the master), which is responsible for launching all applications in the workload. Again, to shorten the length of the experiments, we generate significant changes in offered demand in very little time. The Gigaset network is not used for these experiments.

**Application/OS negotiation experiments.** In order to demonstrate the application/OS negotiation in a controlled manner, a new synthetic application was used. This application was set to consume roughly 65% of CPU time, no I/O and a negligible amount of memory. We set the cluster-wide parameter *degrad* to 40%. The idea is that two of these applications can share a node without triggering node additions, because  $65\% + 65\% = 130\%$  which is below the 140% threshold (40% degradation). This experiment uses only 4 nodes, all of them powered on initially, and 4 copies of the synthetic application, one on each node. We set two of the copies to refuse any execution time degradation and the other two to accept any degradation. The Gigaset network is not used in these experiments.

## 4 Experimental Results

In this section we present the results of our experiments with the network server and the two versions of the cycle server.

**Power-aware cluster-based network server.** Figure 5 presents the evolution of the cluster configuration and demands for each resource as a function of time in seconds. The demand of each resource is plotted as a percentage of the nominal throughput of the same resource in one node. The figure shows that for this particular workload the network interface is the bottleneck resource throughout the whole execution of the experiment (1 hour and 20 minutes), followed closely by the disk. We started the experiment with a single-node configuration. As the traffic directed to the server increases, the disk and network demands increase and eventually trigger the addition of a new node. The load on the server keeps increasing, triggering the addition of several other nodes. The addition of a new node occurs once every 200 seconds (the `elapse` value). At about halfway through the experiment, the load on the cluster starts to subside. The server responds to this change in load by excluding cluster nodes, one at a time, again respecting the `elapse` parameter.

Figure 6 presents the power consumption of the whole cluster for two versions of the same experiment as a function of time. The lower curve (labeled “Dynamic Configuration”) represents the version in which we run the power-aware server, i.e. the cluster configuration is dynamically adapted to respond to variations in resource demand. The higher curve (labeled “Static Configuration”) represents a situation where we run the original server, i.e. the cluster configuration is fixed at 8 nodes. As can be seen in the figure, our modified WWW server can reduce power consumption significantly for most of the execution of our experiment. Power savings actually reach 86% when the resource demands require only a single node. Our energy savings are also significant. Calculating the area below the two curves, we find that the modified WWW server saves 43% in energy. An interesting observation in this figure is the power spike required during the booting of a node. However, the spikes quickly disappear and should not be a problem even if we want to add more than one node at a time.

Finally, it is important to make sure that throughput is not sacrificed excessively in favor of power and energy savings. Figure 7 shows the throughput of the server in requests serviced per second for the two versions mentioned above as a function of time. The figure shows that throughput only suffers significantly in comparison to the static system during the times in which nodes are being added to or removed from the system. During these times, each node of the server has to update its internal data structures and communication channels. A new node has to load its cache. All of these operations, especially the latter, induce overheads that are responsible for the lower throughput. Overall, the dynamic configuration services only 19% fewer requests than its static counterpart in this experiment.

Besides reconfiguration overheads, the other factor that may cause a significant loss in throughput is a mismatch between the value of the `elapse` parameter and the rate with which the workload changes. Comparing figures 7 and 8 we can see this clearly. Figure 8 again plots the throughput of two versions of the server, but this time the power-aware server uses `elapse` = 300 seconds. Even though our power and energy savings are roughly the same as in the previous experiment, this figure suggests that the load on the cluster increases too fast in the beginning of our experiment for the power-aware server to keep up. During this phase, the time it takes to add

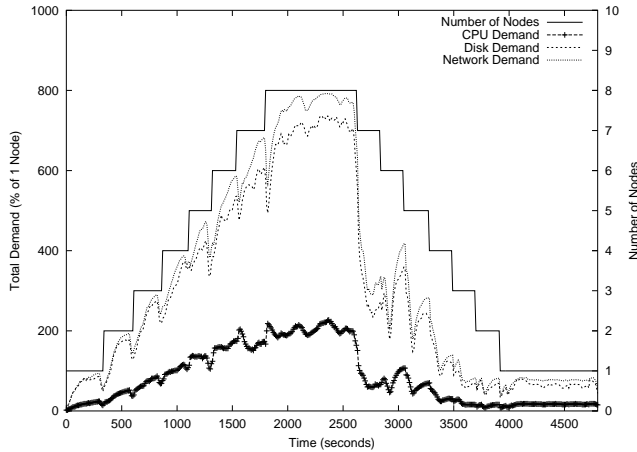


Figure 5: Cluster evolution and resource demands for the WWW server.  $\text{elapse} = 200$  seconds.

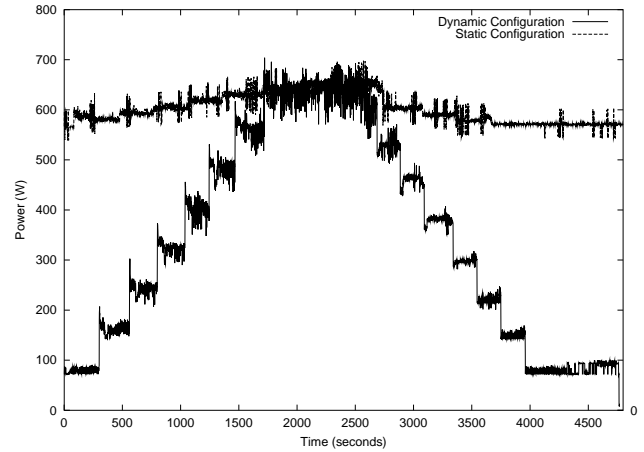


Figure 6: Power consumption for the WWW server under static and dynamic cluster configurations.  $\text{elapse} = 200$  seconds.

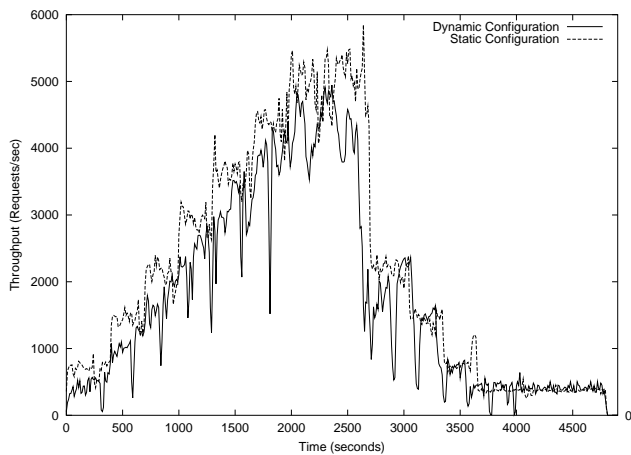


Figure 7: Throughput of the WWW server under static and dynamic cluster configurations.  $\text{elapse} = 200$  seconds.

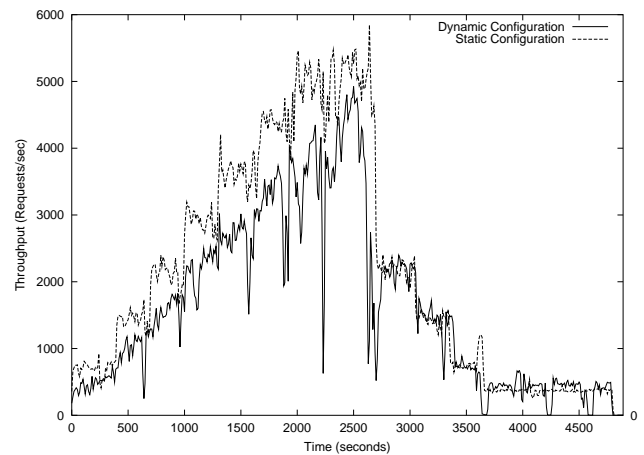


Figure 8: Throughput of the WWW server under static and dynamic cluster configurations.  $\text{elapse} = 300$  seconds.

new nodes becomes a problem and the dynamic system ends up servicing 27% fewer requests than the static system. Overall, the dynamic system services 23% fewer requests than the static system.

Mismatches between the rate of workload change and cluster reconfigurations can be alleviated by either changing  $\text{elapse}$  (maybe dynamically) or allowing the addition or removal of more than one node at a time. We believe however that in practice a value of a few minutes for  $\text{elapse}$  and one addition/removal at a time should work just fine, since real network server workloads are likely to change more slowly than in our experiments.

**Power-aware OS for clusters.** Figure 9 presents the evolution of the cluster configuration and demands for each resource with  $\text{elapse} = 90$  seconds and  $\text{degrad} = 20\%$ , as a function of time. The experiment lasted 50 minutes. Either CPU or disk is the bottleneck resource during the experiment, whereas memory was never used to its maximum. The experiment starts with a single-node configuration. This node is responsible for starting all the applications in the workload. As new applications are started, CPU and I/O demands increase and eventually trigger the addition of a new node. When the new node is added by the master, the OS attempts to bal-

ance the load by migrating some applications to the new node. As the number of applications started by the workload increases, they trigger the addition of other nodes, one at a time. The OS is able to track the demand increases fairly well by increasing the size of the cluster. At about half way through the experiment, the demand for CPU becomes much higher than can be managed by an 8-node cluster. Right after this peak in demand however, some applications start to finish computing and the demand for resources drops quickly. The master responds to this change in load by excluding the now idle nodes, one at a time. Again, the system does a good job of tracking the decrease in resource demand.

Figure 10 presents the power consumption of the whole cluster for two versions of the same experiment as a function of time. The lower curve represents the version in which we run the base power-aware OS, i.e. the cluster configuration is adapted to respond to variations in resource demand. The higher curve represents a situation where we run the original distributed OS, i.e. the cluster configuration is fixed at 8 nodes. As can be seen in the figure, our base power-aware OS can reduce power consumption significantly for most of the execution time of the experiment. Power savings actually reach 86% when the resource demands require only a single node. Energy sav-

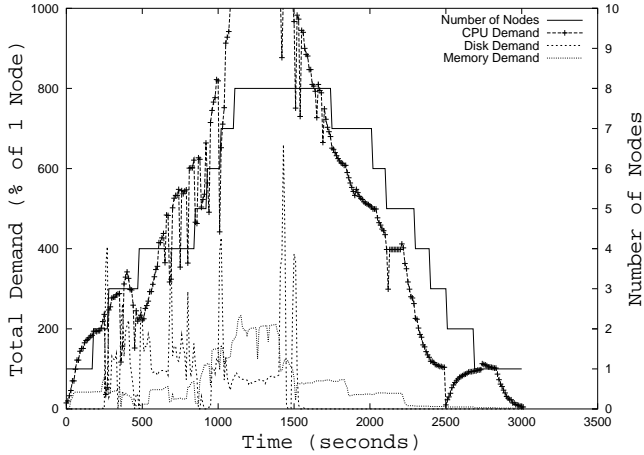


Figure 9: Cluster evolution and resource demands in the power-aware OS. `elapsed` = 90 seconds.

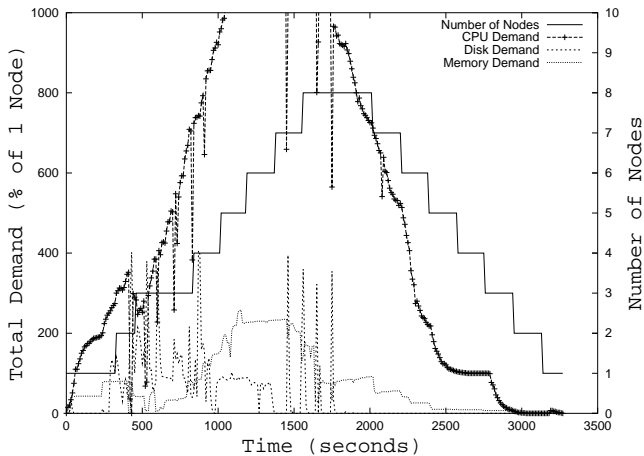


Figure 11: Cluster evolution and resource demands in the power-aware OS. `elapsed` = 180 seconds.

ings are also significant. The area below the two curves indicates that the power-aware OS saves 32% in energy for this workload.

It is interesting to note that the workload used in this experiment finishes earlier on the static configuration (around 33 minutes) than on the dynamic one (around 46 minutes). If we compare the energy consumed by the static configuration during the first 33 minutes of the experiment against that of the dynamic configuration for the whole experiment, we find that our energy savings are smaller but still significant, 20%. This comparison does not seem fair however, since real (i.e. static) cycle servers are never turned off.

We now turn to the execution time of the applications running on the cluster. The fact that the resource demands are frequently lower than the cluster capacity in figure 9 shows that applications experienced little degradation in this experiment. The main reason for that is that the low value for the `elapsed` parameter (90 seconds) allowed the OS to track the rapid changes in the amount of load imposed on the cluster. Figure 11 shows that a longer time between reconfigurations, 180 seconds, would cause applications to suffer much greater performance degradations. Even though performance becomes worse in this experiment, our savings in power and energy

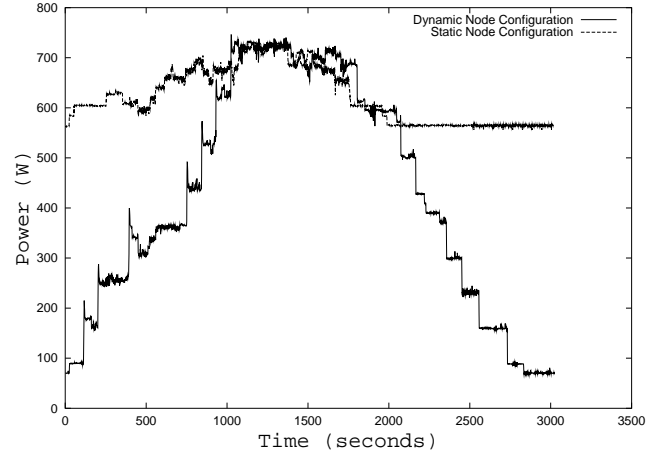


Figure 10: Power consumption for the power-aware OS under static and dynamic cluster configurations. `elapsed` = 90 seconds.

are almost exactly the same as before, 86% and 36%, respectively. Considering that applications finish sooner on the static configuration, our energy gains are 19%.

**Application/OS negotiation for power.** As mentioned before, we experimented with application/OS negotiation on only 4 nodes and 4 copies of the same synthetic applications, one on each node initially. Two copies are set to refuse any execution time degradation and the other two to accept any degradation. The values of `elapsed` and `degrad` are 3 minutes and 40%, respectively.

Figure 12 presents the evolution of the total demand for resources in the same way as for the previous experiments. During the execution, two applications are migrated together and the vacated node is turned off, while the other two are kept separately on different nodes as per their requirement of no performance degradation. As applications start to finish, the demand for CPU is reduced and two nodes can be turned off.

Figure 13 shows the power consumed by the workload as it executes and finishes. We compare the versions of the same experiment: (1) one version is our application/OS negotiation system (labeled “Application Controlled”); (2) one version in which we run the same OS but all 4 applications accept any degradation (labeled “No Application Control”); and (3) one version in which the applications are run on the static system (labeled “Static Configuration”). In version (2), two applications are migrated to two other nodes, so their original nodes are turned off. In version (3), the load remains balanced throughout the execution, so there are no application migrations.

The results of this comparison show that our application-controlled system can consume up to 75% less power than a static system, while saving 29% in terms of energy against that system. The workload does finish earlier on the static system, though. If we take this fact into consideration, we find that our energy savings are only of 21%. Comparing the two dynamic configurations, as one would expect, we find that the restrictions imposed by two of the applications reduce the power and energy gains that can be achieved by a few percent.

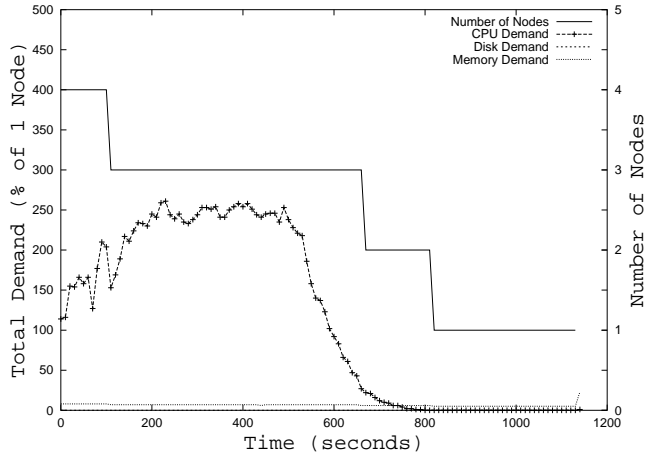


Figure 12: Cluster evolution and resource demands in the power-aware OS with application/OS negotiation.

## 5 Related Work

As mentioned in the introduction to this paper, a recent paper [7] also suggests the study of power and energy issues for clusters of workstations or PCs. In that paper the authors propose an energy-conscious switch for WWW servers, but does not evaluate the idea. Our paper takes a broader approach, proposing, implementing, and evaluating three different types of power-aware systems for clusters. In particular, we consider application, OS, and application/OS-level implementations of power-awareness in clusters. As discussed below, our work is related to other previous works in different ways.

**Laptops and other mobile devices.** Most of the previous work on power and energy conservation has been focused on laptop computers and embedded and hand-held devices. Research on these devices has included optimizations for the processor (e.g. [33, 16, 18]), for the memory (e.g. [22, 32]), for the disk (e.g. [23, 11, 17]), for transcoding the content they receive (e.g. [13, 10, 6]), and for off-loading computation from them (e.g. [28, 21]). All of these studies considered single-processor battery-operated devices. We are not aware of any work, besides ours, involving power and energy optimizations for clusters.

**System-level techniques.** The OS has been the target of power and energy research as well (e.g. [33, 30, 22, 3, 13]). Vahdat *et al.* [30] suggest several aspects that the OS should take into account when running on batteries and what could be done to avoid using energy unnecessarily, like performing computation remotely and turning off unnecessary devices (including portions of the memory subsystem). In a later study, Lebeck *et al.* [22] further exploit the memory subsystem by directing memory accesses to certain memory banks and turning off the unused banks.

In [3], Benini *et al.* suggest that the OS should monitor resource usage so that shutdown operations can be determined by the OS more accurately than by applications alone or hardware alone.

Flinn *et al.* [13] developed a user-level middleware to filter and transcode data that applications fetch. Transcoding changes data quality in order for applications to use the minimum amount of energy when processing it. Vahdat *et al.* [6] and De Lara *et al.* [10] also concerned themselves with transcoding.

**Application and application/OS techniques.** We are not aware

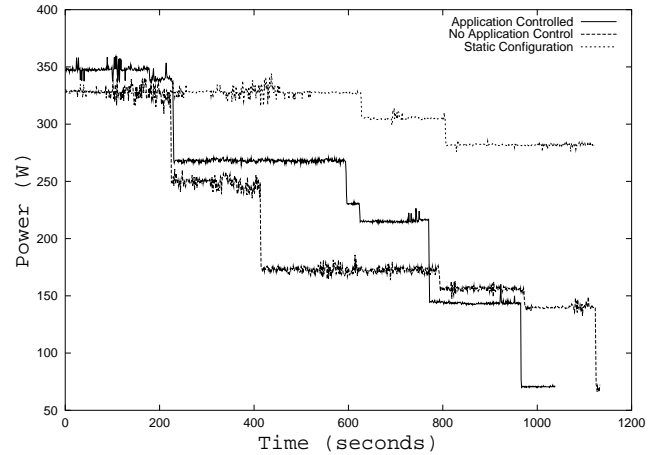


Figure 13: Power consumption for the power-aware OS with application/OS negotiation under static and dynamic configurations.

of any power or energy optimization works that do not rely on some level of runtime/OS or hardware support; our power-aware WWW server seems to be the first example of building power conservation completely into the application itself.

A few previous papers considered application/OS interactions intended to optimize for power and energy. Lu *et al.* [25] implement a technique in which applications communicate to the OS their resource demands. When no application wants to use a certain device, the device is put in sleep mode immediately.

Flinn *et al.* [13] also considered application/OS interaction for energy conservation. In their scheme, the kernel upcalls applications to tell them to adapt their energy usage. The decision of whether or not to issue such upcalls is based on estimates of the remaining battery energy and of the future energy demands.

Our paper studied a different form of application/OS interaction for conserving power; one that involves a negotiation phase between the two sides. Several papers have been published about application/OS interactions for purposes other than conserving power and/or energy.

**Load balancing and configuration of clusters.** Several groups have done research on load balancing and cluster configuration, but always without regard for power or energy issues.

The load balancing literature is gigantic (e.g. [2, 14, 24, 12, 20]). The goal of these systems is either to balance load across multiple machines or to harvest the cycles of idle machines. Most of the focus of our work is on load concentration, the inverse of the load balancing operation. Other systems use load concentration, but only as a remedial (e.g. [2, 24]) or management technique.

A few projects deal with cluster reconfiguration, besides the systems that harvest idle machines. One such ongoing research project is Oceano at IBM [1]. In their work, they have three tiers of computers. The first is a front-end system that monitors workload. The third tier is composed of large database servers that are statically assigned to customers. In the second tier, many machines reply to generated requests. These machines can be shifted between customer domains dynamically to answer to higher loads. This work could easily be extended to include powering down unused systems.

The Ensemble system [31] and the ND tool [15] can also adapt their node configurations. In fact, the ND tool was used to change the

configuration of cluster-based network servers. Even though these projects do not consider power and energy issues, they lend themselves naturally to our approach.

## 6 Conclusions and Future Work

In this paper we addressed power conservation for clusters of workstations or PCs. In this context, we proposed a cluster configuration and load distribution algorithm and applied it under three different scenarios. Our experiments showed that it is possible to conserve significant power and energy in the context of clusters.

This paper reports on the first few steps in power and energy conservation for clusters. In the near future, we plan to extend our algorithm to consider energy as well as power tradeoffs for the systems in which throughput is always the performance measure of interest. Our current inability to predict execution time degradations accurately prevents us from doing the same for other systems. We also intend to investigate a more detailed model of the power and energy consumption as a function of the cluster configuration and the offered load. After this work is completed, we plan to focus on other power and energy optimizations for cluster-based systems, and other types of application/OS interactions for power and energy conservation.

## Acknowledgements

We would like to thank Carla Ellis, Brett Fleisch, and Liviu Iftode for comments on the topic of this research. We also thank Uli Kremer, Mike Hsiao, and the rest of the people in the Programming Languages reading group, who helped us improve the quality of this paper. Finally, we would like to thank Uli Kremer for letting us use the power measurement infrastructure of the Energy Efficiency and Low-Power (EEL) lab at Rutgers.

## References

- [1] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, and B. Rochwerger. Oceanic - SLA Based Management of a Computing Utility. In *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, 2001.
- [2] Amnon Barak and Oren La'adan. The MOSIX Multicomputer Operating System for High Performance Cluster Computing. *Journal of Future Generation Computer Systems*, 13(4-5):361–372, March 1998.
- [3] Luca Benini, Alessandro Bogliolo, Stefano Cavallucci, and Bruno Riccó. Monitoring system activity for OS-directed dynamic power management. In *Proceedings of ISPLED 98*, pages 185–190, 1998.
- [4] A. Bestavros, M. Crovella, J. Liu, and D. Martin. Distributed Packet Rewriting and its Application to Scalable Server Architectures. In *Proceedings of the International Conference on Network Protocols*, October 1998.
- [5] E. V. Carrera and R. Bianchini. Efficiency vs. portability in cluster-based network servers. In *Proceedings of the 8th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, June 2001.
- [6] Surender Chandra, Carla Schlatter Ellis, and Amin Vahdat. Differentiated multimedia web services using quality aware transcoding. In *Proceedings of INFOCOM 2000 - Nineteenth Annual Joint Conference of the IEEE Computer And Communications Societies*, 2000.
- [7] J. Chase and R. Doyle. Balance of Power: Energy Management for Server Clusters. <http://www.cs.duke.edu/ari/publications/publications.html>, January 2001.
- [8] Cisco LocalDirector. <http://www.cisco.com/>, 2000.
- [9] ComputerWorld. Net blamed as crisis roils california. January 2001. <http://www.idg.net/go.cgi?id=426336>.
- [10] E. de Lara, D. Wallach, and W. Zwaenepoel. Puppeteer: Component-based adaptation for mobile computing. In *Proceedings of the 3rd Usenix Symposium on Internet Technologies and Systems*, March 2001.
- [11] Fred Douglass and P. Krishnan. Adaptive disk spin-down policies for mobile computers. *Computing Systems*, 8(4):381–413, 1995.
- [12] Fred Douglass and J. Ousterhout. Transparent Process Migration: Design and Alternatives and the Sprite Implementation. *Software: Practice and Experience*, 21(8):757–785, August 1991.
- [13] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the 17th Symposium on Operating Systems Principles*, pages 48–63, 1999.
- [14] D. Ghormley, D. Petrou, S. Rodrigues, A. Vahdat, and T. Anderson. GLUnix: a Global Layer Unix for a Network of Workstations. *Software: Practice and Experience*, February 1998.
- [15] G. Goldszmidt and G. Hunt. Scaling Internet Services by Dynamic Allocation of Connections. In *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management*, pages 171–184, 1999.
- [16] T. Halfhill. Transmeta breaks the x86 low-power barrier. In *Microprocessor Report*, February 2000.
- [17] David P. Helmbold, Darrell D. E. Long, and Bruce Sherrod. A dynamic disk spin-down technique for mobile computing. In *Proceedings of the 2nd International Conference on Mobile Computing (MOBICOM96)*, pages 130–142, 1996.
- [18] C-H. Hsu, U. Kremer, and M. Hsiao. Compiler-directed dynamic frequency and voltage scaling. In *Proceedings of the Workshop on Power-Aware Computer Systems*, November 2000.
- [19] IOzone. IOzone filesystem benchmark. November 2000. <http://www.iozone.org>.
- [20] Yousef A. Kalidi, José M. Barnabéu, Vlada Matena, Ken Shirriff, and Moti Thadani. Solaris MC: A Multi Computer OS. In *Proceedings of 1996 USENIX Conference*, January 1996.
- [21] U. Kremer, J. Hicks, and J. Rehg. Compiler-directed remote task execution for power management. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, October 2000.

- [22] Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, and Carla S. Ellis. Power aware page allocation. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, pages 105–116, 2000.
- [23] Kester Li, Roger Kumpf, Paul Horton, and Thomas Anderson. A quantitative analysis of disk drive power management in portable computers. In *Proceedings of the 1994 Winter USENIX Conference*, pages 279–291, 1994.
- [24] Michael J. Litzkow and Marvin Solomon. Supporting Checkpoint and Process Migration Outside the UNIX Kernel. In *Usenix Conference Proceedings*, pages 283–290, San Francisco, CA, Jan 1992.
- [25] Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Operating-system directed power reduction. In *Proceedings of ISLPED 00*, 2000.
- [26] E. Pinheiro and R. Bianchini. Nomad: A scalable operating system for clusters of uni and multiprocessors. In *Proceedings of the 1st IEEE International Workshop on Cluster Computing*, December 1999.
- [27] Resonate Central Dispatch. <http://www.resonateinc.com/>, 1999.
- [28] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26, 1998.
- [29] The New York Times. There’s money in housing internet servers. April 2001. <http://www.internetweek.com/story/INW20010427S0010>.
- [30] A. Vahdat, A. Lebeck, and C. Ellis. Every joule is precious: The case for revisiting operating system design for energy efficiency. September 2000.
- [31] Robbert Van Renesse, Ken Birman, Mark Hayden, Alexey Vaysburd, and David Karr. Building adaptive systems using Ensemble. *Software Practice and Experience*, 28(9):963–979, 1998.
- [32] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 95–106, 2000.
- [33] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st Symposium on Operating System Design and Implementation*, 1994.