# 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

## Abstract

In this paper we address power conservation for clusters of workstations or PCs. 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 two different ways: (1) at the application level for a cluster-based, locality-conscious network server; and (2) at the operating system level for an operating system for clustered cycle servers. Our experimental results are very favorable, showing that our systems conserve both power and energy in comparison to traditional systems.

## 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. Over the years, a large amount of research has been devoted to low-power and low-energy design and conservation (e.g. [17, 33, 22, 11, 13]).

In contrast with this line of research, in this paper we focus on 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. Our approach to conserving power and energy is to develop systems that can leverage the widespread replication of resources in clusters. 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, 15, 24, 26, 9, 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 exploit load concentration to conserve power. Their key component 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 two ways: (1) at the application level for the network server; and (2) at the OS level for the cycle server. In a previous technical report [27], we also considered implementations that rely on application/OS interaction.

Even though we target power conservation primarily, our experimental results show that our secondary goal of saving energy is achieved as well. Our results show that the modified network server can reduce the total power consumption 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%.

The remainder of this paper is organized as follows. The next section discusses our motivation. Section 3 describes our cluster configuration and load distribution algorithm and its different implementations. Section 4 describes our experimental set-up and the methodology used. Section 5 discusses our experimental results. Section 6 discusses the related work. Finally, section 7 concludes the paper and mentions our future work.

## 2 Motivation

Our motivation in pursuing this research is that large clusters consume significant amounts of power and energy. Power consumption is an important concern for clusters as it directly influences their cooling requirements. In fact, a medium to large number of

*This research has been supported by NSF under grant # CCR-9986046.

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. Besides cooling under normal operation, power consumption also influences the required investments in backup cooling and backup power-generation equipment for clusters that can never be unavailable, such as those of companies that provide services on the Internet. The recent trend towards ultra-dense clusters [29] will only worsen the cooling problem.

Taking a broader perspective, the power requirements of clusters have become a major issue for several states, such as California and New York. Even if these states make a tremendous investment in new power plants in the next several years, power conservation should still be an important goal in that most power-generation technologies (such as nuclear and coal-based generation) have a negative impact on the environment.

Energy consumption is also an important concern for clusters in that 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 and/or dense cluster in a heavily air-conditioned room. Research and teaching organizations, in particular, may find it difficult to cover high energy costs.

The **bottom line** is that to conserve the power and energy consumed by clusters eases deployment and installation, protects the environment, and can potentially save a lot of money. In fact, even when it is not possible to reduce the maximum power requirements of a cluster (i.e. it is not possible to cut down the one-time cost of cooling and backup power-generation systems), reducing the common-case power and energy consumption reduces the operational cost of these systems and the electricity cost.

# 3 The Algorithm

## 3.1 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 decides whether to add (turn on) or remove (turn off) nodes, according to the expected performance and power implications of the decision. Decisions are made dynamically for each cluster configuration and currently offered load.

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. This is a valid assumption, since: (1) in certain environments it is possible to replicate files at all nodes; and (2) when this is not the case, the file servers can transparently be run on machines that do not strictly belong to the cluster or that are not subject to the algorithm.

**Addition/removal decision.** 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. 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 on its current node, so the prediction does not consider demand changes due to unexpected future behavior or due to migrations. 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 at least; two nodes if a 30% 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 CPUs of 80% and 40%. Our optimistic prediction strategy says that these applications could run with a 20% execution time degradation on 1 node (120 - 100 = 20). Our algorithm should decide to remove one of the nodes, if a 20% degradation is acceptable. (In reality, 20% is a *lower bound* on the degradation.)

The acceptable performance degradation can be specified by the cluster administrator or by each application (i.e. user). Ideally, the algorithm could also try to guarantee a maximum performance degradation. This is clearly not possible for execution time perfor-

```
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 1: Pseudo-code for cluster configuration and load distribution algorithm.

mance, 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 depend heavily on the system for which the algorithm is implemented, so we leave the description of these decisions for the next subsection.

**General form.** In its most general form, our algorithm can be described as in figure 1. 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`.

## 3.2 Implementations

Our algorithm has been implemented with minor variations in two different environments: (1) at the application level for a locality-conscious network server that runs alone on a cluster; and (2) at the system level for an OS for clustered cycle servers.

In both 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. Since the cacheable files are static, each node stores a copy of all files on its local disk.

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 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 estimated time to migrate a process of its size. 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.

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

# 4 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 Giganet switch. Each of the nodes contains an 800-MHz Pentium III processor, 512 MBytes of memory, two 7200 rpm disks (only one disk is used in our experiments), and two network interfaces. Shutting a node down takes approximately 45 seconds and bringing it back up takes approximately 100 seconds.

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 nodes 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, 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. To compress these data and smooth out the curves, our figures show only 1 power measurement per as many as 9-10 seconds.

**WWW server experiments.** Besides the main cluster, we use another 12 Pentium-based machines to generate load for the modified WWW server. 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. Requests are directed to the cluster with a bell-shaped distribution. 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 [20]; 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. The offered demand conforms to a bell-shaped curve. Again, to shorten the length of the experiments, we generate significant changes in offered demand in very little time.

Note that several parameters for our implementations of the cluster configuration algorithm (section 3.2) were picked intuitively based on the characteristics of these "accelerated" workloads. Our future work includes experimenting with non-accelerated workloads, for which we will more carefully determine implementation parameters and will consider adjusting some of them dynamically.

# 5 Experimental Results

**Power-aware cluster-based network server.** Figure 2 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

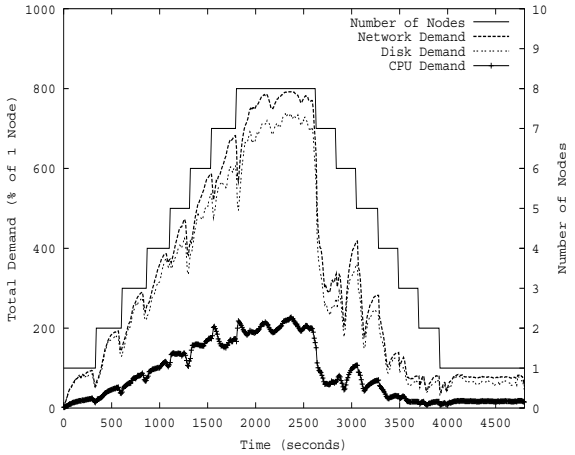


Figure 2: Cluster evolution and resource demands for the WWW server. `elapse` = 200 seconds.

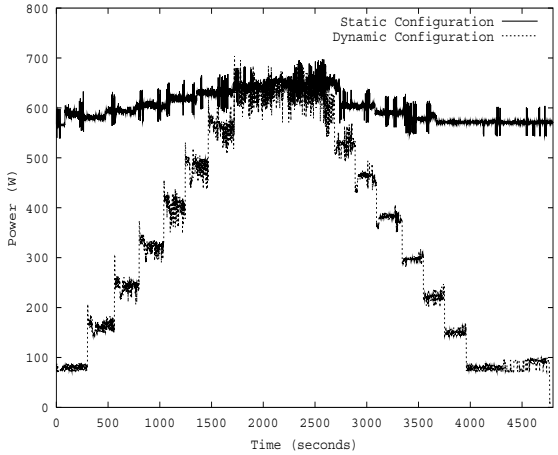


Figure 3: Power consumption for the WWW server under static and dynamic cluster configurations. `elapse` = 200 seconds.

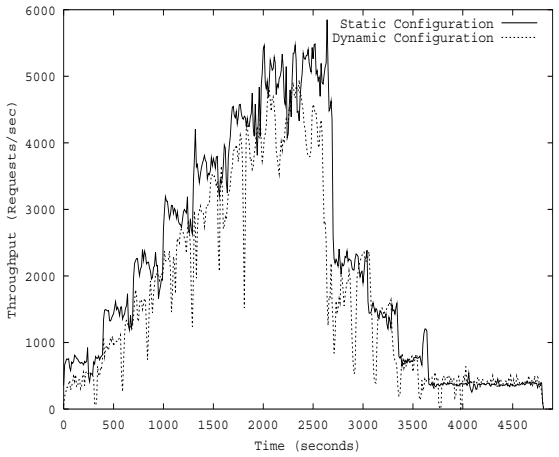


Figure 4: Throughput of the WWW server under static and dynamic cluster configurations. `elapse` = 200 seconds.

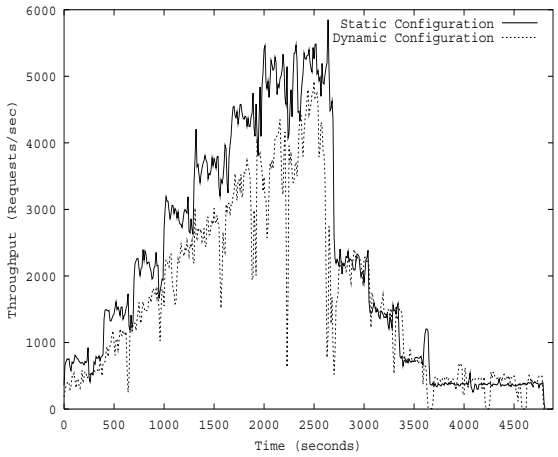


Figure 5: Throughput of the WWW server under static and dynamic cluster configurations. `elapse` = 300 seconds.

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 3 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. Thus, the load on the cooling infrastructure is also reduced by 43%.

Finally, it is important to make sure that throughput is not sacrificed excessively in favor of power and energy savings. Figure 4 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 19% fewer requests than its static counterpart in this experiment. This degradation is relatively small compared to the significant reductions in power and energy consumption obtained by the dynamic system.

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 4 and 5 we can see this clearly. Figure 5 again plots the throughput of two versions of the server, but

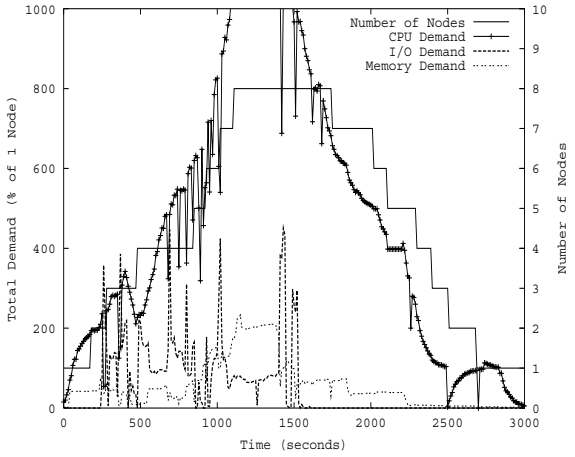


Figure 6: Cluster evolution and resource demands in the power-aware OS. `elapse` = 90 seconds.

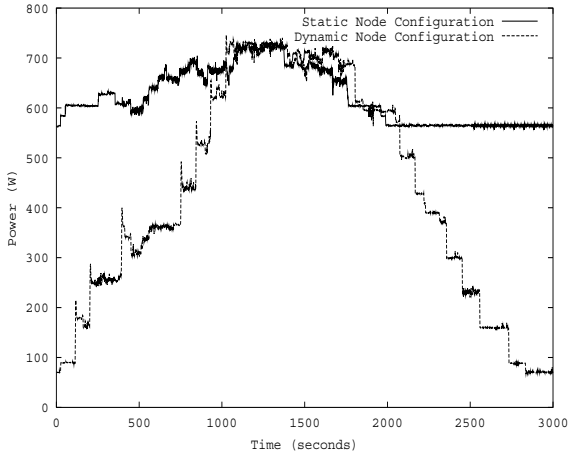


Figure 7: Power consumption for the power-aware OS under static and dynamic cluster configurations. `elapse` = 90 seconds.

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 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 `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 `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 6 presents the evolution of the cluster configuration and demands for each resource with `elapse` = 90 seconds and `degrad` = 20%, as a function of time. The experiment lasted 50 minutes. Either CPU or I/O 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 balance the load by migrating some applications to the new node. As the number of applications started 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 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 7 presents the power consumption of the whole cluster for two versions of the same experiment as a function of time. As can be seen in the figure, our power-aware OS can reduce power consumption significantly for most of the execution time of the experiment.

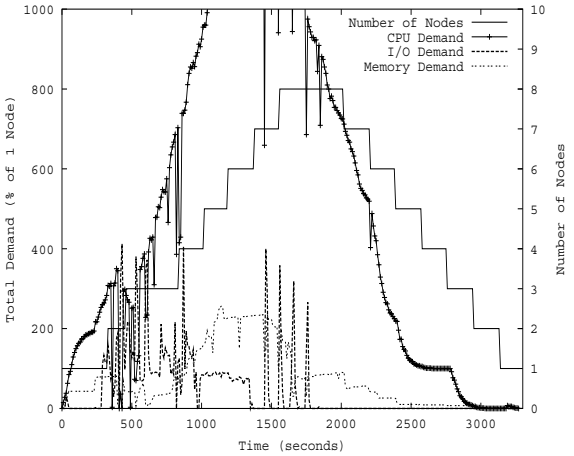


Figure 8: Cluster evolution and resource demands in the power-aware OS. `elapse` = 180 seconds.

Power savings actually reach 86% when the resource demands require only a single node. Energy savings 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 is not really fair however, since real, i.e. static, cycle servers are never turned off). In any case, it is clear that the load on the cooling infrastructure is reduced by at least 20% under the dynamic system.

We now turn to the execution time of the applications. The fact that the resource demands are frequently lower than the cluster capacity in figure 6 shows that applications experienced little degradation. The main reason is that the low value for the `elapse` parameter (90 seconds) allowed the OS to track the rapid changes in offered load. Figure 8 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 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%.

# 6 Related Work

**Single-processor systems.** Most of the previous work on conservation has been focused on laptop computers and embedded and handheld devices. Research on these devices has included optimizations for the processor (e.g. [33, 17, 19]), for the memory (e.g. [22, 32]), for the disk (e.g. [23, 11, 18]), and for offloading computation from them to non-battery-operated computers (e.g. [28, 21]).

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 aspects that the OS should take into account when running on batteries and what could be done to avoid using energy unnecessarily, like turning off unnecessary portions of the memory subsystem. In a later study, Lebeck *et al.* [22] further exploited 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 shutdowns can be determined by the system more accurately than by applications 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.

A few previous papers considered application/OS interactions intended to optimize for power and energy [25, 13].

**Clusters.** Some of the research mentioned above can be used to optimize each node of a cluster independently, so we can also benefit from it. However, our research is orthogonal to these contributions in the sense that we focus on cluster-wide power and energy conservation, i.e. conservation that considers all of the cluster resources and the load offered to the cluster as a whole.

This paper is a shorter and slightly revised version of our previous technical report [27]. Recently we found out that an upcoming paper [7] also deals with power and energy research for clusters. The paper tackles the general problem of resource allocation in hosting centers using market-based policies. In terms of power and energy conservation, the authors evaluate a resource allocation policy for a clustered WWW server that is similar to the cluster configuration algorithm we study here. The paper exploits ideas from a previous paper [8].

As aforementioned, load concentration is inspired by previous work in cluster-wide load balancing (e.g. [2, 15, 24, 12, 26, 9, 4]). Some systems do use some form of load concentration, but only 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. We use load concentration as a first-class technique for conserving power and energy in clusters.

A few other projects deal with cluster reconfiguration (e.g. [14, 1, 31, 16]). Even though these projects do not consider power and energy issues, they lend themselves nicely to the powering down of unused systems.

The technique that is closest in spirit to load concentration for power and energy is offloading computation from a battery-operated device to a remote non-battery-operated computer (e.g. [28, 21]). However, load concentration as described here involves different challenges and tradeoffs, mainly because the load on the cluster and the effect of applying the technique must be determined before any action can be taken.

# 7 Conclusions and Future Work

In this paper we addressed power conservation for clusters. In this context, we proposed a simple cluster configuration and load distribution algorithm and applied it under two different scenarios. Our experiments showed that it is possible to conserve significant power and energy in the context of clusters. Based on our experimental results, we conclude that our algorithm and systems should be useful for organizations and companies that rely on large clusters of servers.

This paper reported on preliminary work. In the near future, we plan to improve and extend our work in several ways. First, we will extend our algorithm and implementations to transition multiple devices between multiple sleep states, rather than just turning entire nodes on and off. This will give us finer control of power and performance. Second, we will extend our algorithm and implementations to consider load migration and state transition costs (in terms of both power and performance) explicitly. By considering these costs, we expect to be able to predict the effect of algorithm decisions more accurately, as well as increase our ability to conserve power without degrading performance. Third, we will extend our algorithm and implementations to explicitly consider energy as well as power tradeoffs. Fourth, we will investigate a more detailed model of the power and energy consumption as a function of the cluster configuration and the offered load. Finally, we plan to experiment with real, non-accelerated workloads.

## Acknowledgements

We would like to thank Carla Ellis, Brett Fleisch, and Liviu Iftode for comments on the topic of this research. We also thank the anonymous referees, 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. Oceano - 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] Surendar 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, D. Anderson, P. Thackar, A. Vahdat, and R. Boyle. Managing energy and server resources in hosting centers. In *To appear in Proceedings of the 18th Symposium on Operating Systems Principles*, October 2001.

[8] J. Chase and R. Doyle. Balance of Power: Energy Management for Server Clusters. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, May 2001.

[9] Cisco LocalDirector. http://www.cisco.com/, 2000.

[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 Douglis and P. Krishnan. Adaptive disk spin-down policies for mobile computers. *Computing Systems*, 8(4):381–413, 1995.

[12] Fred Douglis 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] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier. Cluster-Based Scalable Network Services. In *Proceedings of the International Symposium on Operating Systems Principles*, pages 78–91, October 1997.

[15] 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.

[16] 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.

[17] T. Halfhill. Transmeta breaks the x86 low-power barrier. In *Microprocessor Report*, February 2000.

[18] 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.

[19] 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.

[20] IOzone. Iozone filesystem benchmark. November 2000. http://www.iozone.org.

[21] U. Kremer, J. Hicks, and J. Regh. 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] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. Technical Report DCS-TR-440, Department of Computer Science, Rutgers University, May 2001.

[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] RLX Technologies. Serverblade. June 2001. http://www.rlxtechnologies.com/.

[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.