

## Chapter 1

# DYNAMIC CLUSTER RECONFIGURATION FOR POWER AND PERFORMANCE \*

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

*Department of Computer Science*

*Rutgers University*

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

### 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 cluster reconfiguration decisions by considering the total load imposed on the system and the power and performance implications of changing the current configuration. The algorithm is implemented in two common cluster-based systems: a network server and 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.

**Keywords:** Load balancing, load concentration, power and energy conservation.

### 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

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

design and conservation (e.g. [Halfhill, 2000; Weiser et al., 1994; Lebeck et al., 2000; Douglass and Krishnan, 1995; Flinn and Satyanarayanan, 1999]).

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 most Internet companies and a large number of research and teaching organizations. 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. [Barak and La'adan, 1998; Ghormley et al., 1998; Litzkow and Solomon, 1992; Pinheiro and Bianchini, 1999; Cisco, 2000; Bestavros et al., 1998]). 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 uses a control-theoretic approach to determine whether nodes should be added to or removed from the cluster, 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 in two popular types of cluster-based systems: a network server and an 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 [Pinheiro et al., 2001a], 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 71% and the energy consumption by 45% in comparison to the original server running on a static cluster configuration. The modified OS can reduce power consumption by as much as 88% for a synthetic workload, while attempting to avoid any performance degradation, again in comparison to the original system on a static cluster. The energy savings it accrues in this case is 32%. When a 20% performance degradation is acceptable, our system conserves 88% of the power and 40% of the energy consumed by the static system.

The remainder of this paper is organized as follows. The next section discusses our motivation. 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.

## 1. 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 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 [RLX Technologies, 2001] 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, at the height of the economic boom in the United States. 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.

## 2. Cluster Configuration and Load Distribution

### 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 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. 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.)

It is clear then that a key component of our algorithm is the demand for resources at each point in time. However, the decisions made by the algorithm must not be solely based on instantaneous demands to avoid reconfigurations triggered by transient load variations. The algorithm should also take into account the past history of demands and the speed of change in demands. *Control theory* provides a formal and well-understood approach to considering these properties. Thus, we use a Proportional-Integral-Differential (PID) feedback controller for each resource as the basis for our algorithm's decisions. The controller with the largest output (in absolute value) is used to determine the ideal cluster configuration at each point in time. The formula that describes the output,  $o(t)$ , of each controller is:

$$o(t) = k_p e(t) + k_i \sum_0^t e(t) + k_d \Delta e(t)$$

Each controller calculates the current excess demand (with respect to the current cluster configuration) for a resource, accumulates excess demand over time, and computes the rate of change in excess demands. These are the proportional, integral, and differential components of the controller, respectively. Each of these components is weighted with a tunable constant, which should reflect how much importance we want to give to each component. In our experiments, we used  $k_p = 0.7$ ,  $k_i = 0.15$ , and  $k_d = 0.15$ . Furthermore, we saturate the integral component at the resource capacity of a single node, i.e. 100% plus the acceptable performance degradation. (These constants and saturation value were chosen after some experimentation with our systems.) The output of the controller is the sum of the weighted components. The controller is executed every 10 seconds.

To guarantee stability, our algorithm computes excess demands with respect to the arithmetic mean of the resource capacities of  $N$  and  $N-1$  nodes for a configuration with  $N$  nodes. Moreover, the algorithm only triggers a reconfiguration if the absolute value of the controller's output is greater than half of the resource capacity of a single node plus 10% of this value. To decide how many nodes to add or remove, the algorithm divides the output of the controller minus half of the resource capacity of a node by the resource capacity of a node. For instance, for

a 5-node configuration and a 20% acceptable performance degradation, excess demands would be computed with respect to 540% (the average of  $5 \times 120$  and  $4 \times 120$ ). In this scenario, a controller output of -65% means that the current configuration should not be altered ( $65 < 66$ ). An output of -300% should trigger the removal of two nodes ( $\lceil (|-300| - 60)/120 \rceil = 2$ ).

We refer to the acceptable degradation and the minimum time between reconfigurations as the **degrad** and **elapse** parameters of our algorithm. The **degrad** parameter 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 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 acceptable 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 on the system for which the algorithm is implemented, so we leave the description of these decisions for the next subsection.

## 2.2 Implementations

Our algorithm has been implemented with minor variations in two different environments: (1) at the application level for a network server that runs alone on a cluster; and (2) at the system level for a 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 de-

mand 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.

**Power-aware cluster-based network server.** We modified PRESS [Carrera and Bianchini, 2001], 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.

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(s) 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.

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 that is greater than 0%. In fact, in our experiments we assume that the resource capacity of a single node is either 70% or 85% of its actual capacity, i.e. we study **degrad** parameters of -30% and -15%. These values provide some slack

to compensate for the time it takes for a node to be booted, approximately 100 seconds. We set the default value of the `elapse` parameter to 120 seconds. Given that the interval between demand computations is 10 seconds, this setting for `elapse` allows the server two observations of the state of a reconfigured cluster before another reconfiguration is permitted.

**Power-aware OS for clusters.** We modified Nomad [Pinheiro and Bianchini, 1999], a Linux-based single-system-image 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 observed locally at a node remains higher than a threshold for 5 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 30 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.

Again, 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, the demands are amortized using the same formula used by the WWW server. For our experiments,  $\alpha = 0.8$  and the interval between demand computations is 10 seconds. In case of the

OS implementation of our algorithm, we are interested in execution time performance.

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. 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 not effected if only 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 (about 100 seconds), it might take a while before the demand for resources becomes acceptable again, after a long-lasting surge of activity. We experiment with two values for `degrad`: 0% and 20%. The `elapse` parameter is set to 150 seconds. This setting allows the system time to reconfigure, migrate applications to balance the load, and re-evaluate the resource demands before another reconfiguration is allowed.

### 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 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.

**Network server experiments.** Besides the main cluster, we use another 14 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 according to a trace of the accesses to the World Cup '98 site from 12pm on 07/12 to 12pm on 07/14. The trace includes the day of the championship match. To shorten the duration of the experiments, we accelerated the trace 20 times.

**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 [IOzone, 2000]; 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. To shorten the length of the experiments, we generate significant changes in offered demand in very little time.

## 4. Experimental Results

**Power-aware cluster-based network server.** We describe two experiments with our server. In the first experiment, the parameters for the cluster reconfiguration algorithm are set to guarantee quick reaction to fluctuations in load, so that we can tackle significant increases in load without performance degradation. More specifically, the algorithm tries to keep 30% spare resources for any cluster configuration. Figure 1.1 presents the evolution of the cluster configuration and demands for each resource as a function of time in seconds for this experiment. The demand for 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 (140 minutes). We started the experiment with a two-node configuration. The traffic directed to the server initially increases

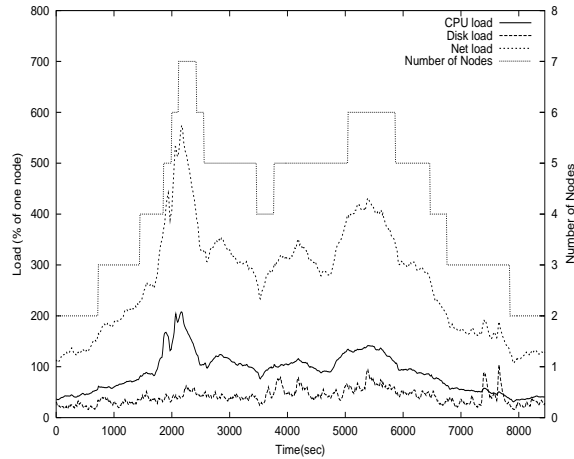


Figure 1.1. Cluster evolution and resource demands for the WWW server. `elapse` = 120 seconds; `degrad` = -30%.

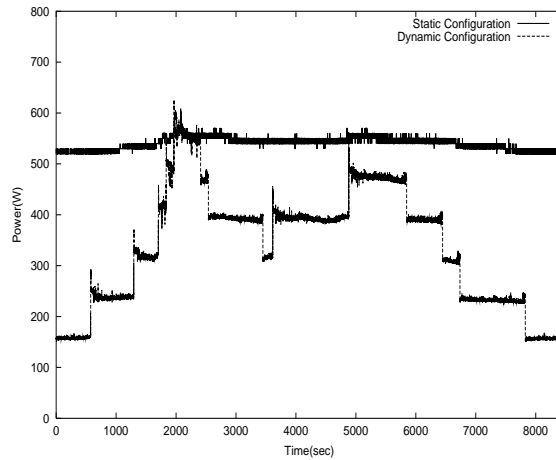


Figure 1.2. Power consumption for the WWW server under static and dynamic cluster configurations. `elapse` = 120 seconds; `degrad` = -30%.

slowly, triggering the addition of a node, before increasing substantially and triggering the addition of several new nodes in quick sequence. The traffic then subsides, until another period of high traffic occurs, which is followed by a substantial decline in traffic. Note that throughout the experiment the system reacts quickly to increases in traffic, because of the spare capacity it consistently retains. As a result of the spare capacity, the performance of the server is not affected by the dynamic reconfiguration of the cluster.

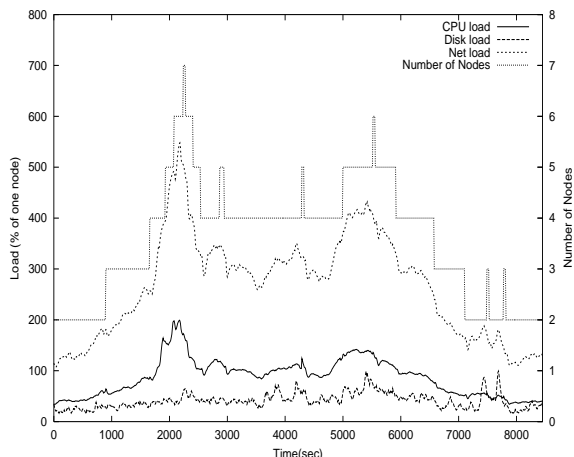


Figure 1.3. Cluster evolution and resource demands for the WWW server. `elapse` = 120 seconds; `degrad` = -15%.

Figure 1.2 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 7 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 71% when the resource demands require only two nodes. Our energy savings are also significant. Calculating the area below the two curves, we find that the power-aware server saves 38% in energy. Thus, the load on the cooling infrastructure over time is also reduced by 38%.

Even though these are significant gains, we can do better. The reason is that keeping spare capacity promotes performance at the cost of higher power and energy consumption. If we can estimate how fast the offered traffic can increase, we can reduce the spare capacity to the minimum required to avoid excessively long request latencies. For our trace, this minimum is 15%. Thus, figure 1.3 presents the evolution of the cluster configuration when the system attempts to retain this much spare capacity. Comparing figures 1.1 and 1.3 we can see that during most of the experiment the system now requires fewer active nodes to handle the offered load. In this case, the power and energy gains that

can be achieved in comparison to a static system with 7 nodes are 71% and 45%, respectively.

In general, it might not be possible to determine the maximum rate of workload change a priori. In these cases, mismatches between the rate of workload change and cluster reconfiguration delays can be alleviated by adjusting the `elapse` and/or `degrad` parameters dynamically. We believe however that in practice values of a few minutes for `elapse` and a few percent for `degrad` should work just fine, since real network server workloads usually change more slowly than in our experiments.

**Power-aware OS for clusters.** Figure 1.4 presents the evolution of the cluster configuration and demands for each resource with `elapse` = 150 seconds and `degrad` = 0%, as a function of time. The experiment lasted about 46 minutes. The CPU is always the bottleneck resource during the experiment. 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, the CPU demand increases and eventually triggers 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 1.5 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. Power savings actually reach 88% 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 35% in energy for this workload.

It is interesting to note that the workload used in this experiment finishes a little earlier on the static configuration (around 45 minutes) than on the dynamic one (around 46 minutes). If we compare the energy consumed by the static configuration during the first 45 minutes of the experiment against that of the dynamic configuration for the whole experiment, we find that our energy savings are only slightly smaller, 32%.

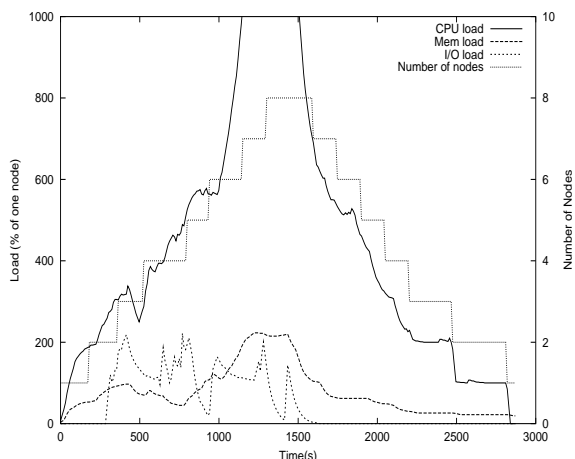


Figure 1.4. Cluster evolution and resource demands in the power-aware OS. `elapsed` = 150 seconds; `degraded` = 0%.

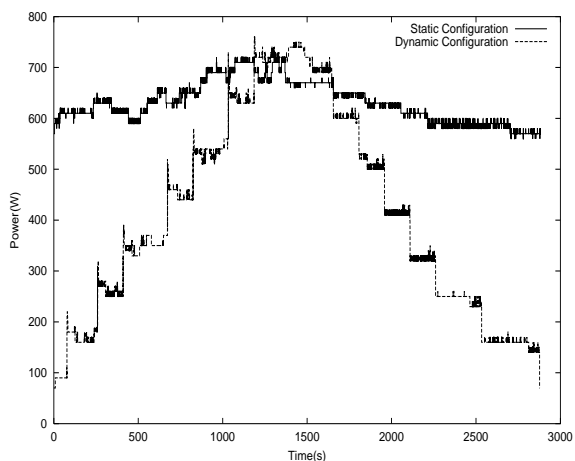


Figure 1.5. Power consumption for the power-aware OS under static and dynamic cluster configurations. `elapsed` = 150 seconds; `degraded` = 0%.

(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 32% under the dynamic system.

To investigate the tradeoff between performance and power, we also performed experiments in which our intended performance degradation is 20%. We kept `elapsed` at 150 seconds. Figure 1.6 illustrates the evolution of the cluster configuration in this case. As one would expect,

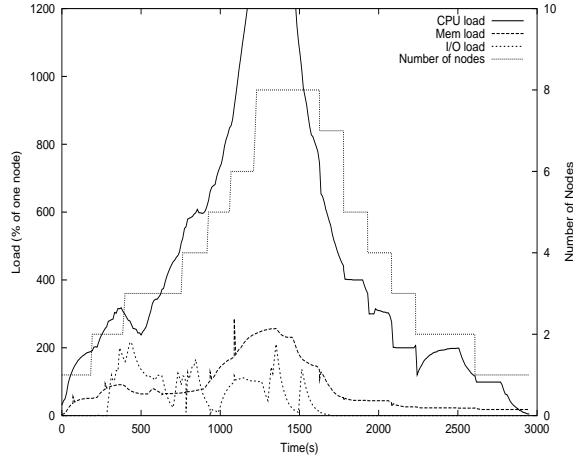


Figure 1.6. Cluster evolution and resource demands in the power-aware OS. `elapse` = 150 seconds; `degrad` = 20%.

allowing for some performance degradation has the effect of slowing down the addition of new nodes and speeding up the removal of unnecessary nodes. As a result, the system decides to jump directly from 6 to 8 nodes when ramping up and to jump directly from 7 to 5 nodes when down-sizing. Overall, our system conserves 88% and 42% of the power and energy consumed by its static counterpart in this experiment. If we consider that the workload finishes 2 minutes later on the dynamic than on the static configuration, the energy gains are of 40%.

## 5. Related Work

Most of the previous work on 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. [Weiser et al., 1994; Halfhill, 2000; Hsu et al., 2000]), for the memory (e.g. [Lebeck et al., 2000; Vijaykrishnan et al., 2000; Delaluz et al., 2001]), for the disk (e.g. [Li et al., 1994; Douglass and Krishnan, 1995; Helmbold et al., 1996]), and for offloading computation from them to non-battery-operated computers (e.g. [Rudenko et al., 1998; Kremer et al., 2000]).

Some of this research 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.

We originally proposed the ideas and systems described here in [Pinheiro et al., 2001a]. A shorter and revised version of that paper appears in [Pinheiro et al., 2001b]. This paper extends our original work in several ways: (a) our original cluster configuration and load distribution algorithm did not consider the past behavior and speed of change of the offered workload when making its decisions; (b) our original cluster reconfiguration decisions were limited to adding or removing a single node at a time; and (c) our original evaluation of the power-aware server assumed a synthetic workload. Extensions (a) and (b) increased our ability to track and quickly adjust to variations in offered load, whereas extension (c) allows us to demonstrate the usefulness of our approach in realistic scenarios.

Two recent papers [Chase et al., 2001; Elnozahy et al., 2002] also deal with power and energy research for clusters. Chase *et al.* [Chase et al., 2001] tackled the general problem of resource allocation in hosting centers using market-based policies. In terms of power and energy conservation, they evaluated a resource allocation policy for a clustered WWW server that is similar to the cluster configuration algorithm we study here. Elnozahy *et al.* [Elnozahy et al., 2002] evaluated different combinations of cluster reconfiguration and dynamic voltage scaling for clusters. They showed that the benefits of our technique can be increased by coupling it with coordinated (cluster-wide) voltage scaling.

As aforementioned, load concentration is inspired by previous work in cluster-wide load balancing (e.g. [Barak and La'adan, 1998; Ghormley et al., 1998; Litzkow and Solomon, 1992; Douglass and Ousterhout, 1991; Pinheiro and Bianchini, 1999; Cisco, 2000; Bestavros et al., 1998]). Some systems do use some form of load concentration, but only as a remedial technique like in systems that harvest idle workstations (e.g. [Barak and La'adan, 1998; Litzkow and Solomon, 1992]) 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.

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. [Rudenko et al., 1998; Kremer et al., 2000]). 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.

A few other projects deal with cluster reconfiguration (e.g. [Fox et al., 1997; Appleby et al., 2001; Van Renesse et al., 1998; Goldszmidt and Hunt, 1999]). Even though these projects do not consider power and en-

ergy issues, they lend themselves nicely to the powering down of unused systems.

In terms of our algorithm, the most closely related work is that of Skadron *et al.* [Skadron et al., 2002]. They proposed the use of control-theoretic techniques for dynamic thermal management of microprocessors. We apply similar techniques to cluster reconfiguration for power and performance.

## 6. Conclusions

In this paper we addressed power and energy conservation for clusters. We proposed a control-theoretic cluster configuration and load distribution algorithm and applied it under two different scenarios. Our experiments showed that it is indeed possible to conserve significant power and energy in the context of clusters. Based on our experimental results, we conclude that exploiting periods of light load can provide tremendous gains for organizations and companies that rely on large clusters of servers.

## 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

- Appleby, K., Fakhouri, S., Fong, L., Goldszmidt, G., Kalantar, M., Krishnakumar, S., Pazel, D., Pershing, J., and Rochwerger, B. (2001). Oceano - SLA Based Management of a Computing Utility. In *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*.
- Barak, A. and La'adan, O. (1998). The MOSIX Multicomputer Operating System for High Performance Cluster Computing. *Journal of Future Generation Computer Systems*, 13(4-5):361–372.
- Bestavros, A., Crovella, M., Liu, J., and Martin, D. (1998). Distributed Packet Rewriting and its Application to Scalable Server Architectures. In *Proceedings of the International Conference on Network Protocols*.
- Carrera, E. V. and Bianchini, R. (June 2001). Efficiency vs. portability in cluster-based network servers. In *Proceedings of the 8th ACM SIG-*

- PLAN Symposium on Principles and Practice of Parallel Programming.*
- Chase, J., Anderson, D., Thacker, P., Vahdat, A., and Boyle, R. (October 2001). Managing energy and server resources in hosting centers. In *Proceedings of the 18th Symposium on Operating Systems Principles*. Cisco (2000). Cisco LocalDirector. <http://www.cisco.com/>.
- Delaluz, V., Kandemir, M., Vijaykrishnan, N., Sivasubramaniam, A., and Irwin, M. J. (January 2001). Dram energy management using software and hardware directed power mode control. In *Proceedings of the International Symposium on High-Performance Computer Architecture*.
- Douglis, F. and Krishnan, P. (1995). Adaptive disk spin-down policies for mobile computers. *Computing Systems*, 8(4):381–413.
- Douglis, F. and Ousterhout, J. (1991). Transparent Process Migration: Design and Alternatives and the Sprite Implementation. *Software: Practice and Experience*, 21(8):757–785.
- Elnozahy, E. N., Kistler, M., and Rajamony, R. (February 2002). Energy-Efficient Server Clusters. In *Proceedings of the 2nd Workshop on Power-Aware Computing Systems*.
- Flinn, J. and Satyanarayanan, M. (1999). Energy-aware adaptation for mobile applications. In *Proceedings of the 17th Symposium on Operating Systems Principles*, pages 48–63.
- Fox, A., Gribble, S., Chawathe, Y., Brewer, E., and Gauthier, P. (1997). Cluster-Based Scalable Network Services. In *Proceedings of the International Symposium on Operating Systems Principles*, pages 78–91.
- Ghormley, D., Petrou, D., Rodrigues, S., Vahdat, A., and Anderson, T. (1998). GLUnix: a Global Layer Unix for a Network of Workstations. *Software: Practice and Experience*.
- Goldszmidt, G. and Hunt, G. (1999). Scaling Internet Services by Dynamic Allocation of Connections. In *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management*, pages 171–184.
- Halfhill, T. (February 2000). Transmeta breaks the x86 low-power barrier. In *Microprocessor Report*.
- Helmbold, D. P., Long, D. D. E., and Sherrod, B. (1996). A dynamic disk spin-down technique for mobile computing. In *Proceedings of the 2nd International Conference on Mobile Computing (MOBICOM96)*, pages 130–142.
- Hsu, C.-H., Kremer, U., and Hsiao, M. (November 2000). Compiler-directed dynamic frequency and voltage scaling. In *Proceedings of the Workshop on Power-Aware Computer Systems*.
- IOzone (November 2000). Iozone filesystem benchmark. <http://www.iozone.org>.

- Kremer, U., Hicks, J., and Regh, J. (October 2000). Compiler-directed remote task execution for power management. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*.
- Lebeck, A. R., Fan, X., Zeng, H., and Ellis, C. S. (2000). 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.
- Li, K., Kumpf, R., Horton, P., and Anderson, T. (1994). A quantitative analysis of disk drive power management in portable computers. In *Proceedings of the 1994 Winter USENIX Conference*, pages 279–291.
- Litzkow, M. J. and Solomon, M. (1992). Supporting Checkpoint and Process Migration Outside the UNIX Kernel. In *Usenix Conference Proceedings*, pages 283–290, San Francisco, CA.
- Pinheiro, E. and Bianchini, R. (December 1999). Nomad: A scalable operating system for clusters of uni and multiprocessors. In *Proceedings of the 1st IEEE International Workshop on Cluster Computing*.
- Pinheiro, E., Bianchini, R., Carrera, E. V., and Heath, T. (2001a). Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. Technical Report DCS-TR-440, Department of Computer Science, Rutgers University.
- Pinheiro, E., Bianchini, R., Carrera, E. V., and Heath, T. (2001b). Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. In *Proceedings of the International Workshop on Compilers and Operating Systems for Low Power*.
- RLX Technologies (June 2001). Serverblade. <http://www.rlxtechnologies.com/>.
- Rudenko, A., Reiher, P., Popek, G. J., and Kuenning, G. H. (1998). Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26.
- Skadron, K., Stan, M., and Abdelzaher, T. (February 2002). Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management. In *Proceedings of the International Symposium on High-Performance Computer Architecture*.
- Van Renesse, R., Birman, K., Hayden, M., Vaysburd, A., and Karr, D. (1998). Building adaptive systems using Ensemble. *Software Practice and Experience*, 28(9):963–979.
- Vijaykrishnan, N., Kandemir, M., Irwin, M. J., Kim, H. S., and Ye, W. (2000). Energy-driven integrated hardware-software optimizations using simplepower. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 95–106.
- Weiser, M., Welch, B., Demers, A., and Shenker, S. (1994). Scheduling for reduced cpu energy. In *Proceedings of the 1st Symposium on Operating System Design and Implementation*.