Energy Conservation in Heterogeneous Server Clusters*

Taliver Heath
Dept. of Computer Science
Rutgers University
Piscataway, NJ
taliver@cs.rutgers.edu

Bruno Diniz
Dept. of Computer Science
Federal Univ. of Minas Gerais
Belo Horizonte, Brazil
diniz@dcc.ufmg.br

Enrique V. Carrera
Dept. of Computer Science
Univ. San Francisco of Quito
Quito, Ecuador
vinicioc@usfq.edu.ec

Wagner Meira Jr.
Dept. of Computer Science
Federal Univ. of Minas Gerais
Belo Horizonte, Brazil
meira@dcc.ufmg.br

Ricardo Bianchini
Dept. of Computer Science
Rutgers University
Piscataway, NJ
ricardob@cs.rutgers.edu

ABSTRACT

The previous research on cluster-based servers has focused on homogeneous systems. However, real-life clusters are almost invariably heterogeneous in terms of the performance, capacity, and power consumption of their hardware components. In this paper, we argue that designing efficient servers for heterogeneous clusters requires defining an efficiency metric, modeling the different types of nodes with respect to the metric, and searching for request distributions that optimize the metric. To concretely illustrate this process, we design a cooperative Web server for a heterogeneous cluster that uses modeling and optimization to minimize the energy consumed per request. Our experimental results for a cluster comprised of traditional and blade nodes show that our server can consume 42% less energy than an energy-oblivious server, with only a negligible loss in throughput. The results also show that our server conserves 45% more energy than an energy-conscious server that was previously proposed for homogeneous clusters.

Categories and Subject Descriptors

D.4 [**Operating systems**]: Organization and Design

General Terms

Design, experimentation, measurement

Keywords

Energy conservation, server clusters, heterogeneity

*This research has been supported by NSF under grants #EIA-0224428, #CCR-0100798, and #CCR-0238182 (CAREER award), and CNPq/Brazil under grants #680.024/01-8 and #380.134/97-7.

PPoPP'05, June 15–17, 2005, Chicago, Illinois, USA.

1. INTRODUCTION

Most of the previous research on cluster-based servers (or simply "server clusters") has focused on request distribution for improved performance (e.g., [3, 4, 9, 21]) and dynamic cluster reconfiguration for energy conservation without performance degradation [10, 15, 23, 24]. Because these works focused solely on homogeneous clusters, they found essentially that the cluster configuration should be the smallest (in number of nodes) needed to satisfy the current offered load, whereas requests should be evenly distributed across the nodes modulo locality considerations.

However, real-life server clusters are almost invariably heterogeneous in terms of the performance, capacity, and power consumption of their hardware components. For example, the Teoma/AskJeeves search engine is supported by a highly heterogeneous server cluster with thousands of nodes. In fact, the different services involved in the search engine, such as the indexing and Web services, are themselves supported by heterogeneous nodes. The heterogeneity comes from nodes with different processor and network interface speeds, as well as different numbers of processors and memory sizes [25].

The reason for the heterogeneity of real-life server clusters is simple and at least three-fold: (1) failed or misbehaving components are usually replaced with different (more powerful) ones, as cost/performance ratios for off-the-shelf components keep falling; (2) any necessary increases in performance or capacity, due to expected increases in offered load, are also usually made with more powerful components than those of the existing cluster; and (3) traditional, PC-style nodes are slowly being replaced by collections of single-board "blade" nodes to save physical data center space and ease management. The combination of traditional and blade nodes makes for highly heterogeneous clusters, since some blade systems exploit laptop technology to consume significantly less energy than traditional computers. In essence, clusters are only homogeneous (if at all) when first installed.

Heterogeneity raises the problem of how to distribute the clients' requests to the different cluster nodes for best performance. Furthermore, heterogeneity must be considered in cluster reconfiguration for energy conservation, raising the additional problem of how to configure the cluster for an appropriate tradeoff between energy conservation and performance. None of the previous approaches to request distribution and cluster configuration is ideal for heterogeneous systems, since they are oblivious to the characteristics of the different types of nodes and/or requests, can under-utilize resources, or do not consider energy consumption explicitly.

In this paper, we design a server cluster that can adjust its configuration and request distribution to optimize power, energy, throughput, latency, or some combination of these metrics. The particular optimization function can be defined by the system administrator; for this paper, we select the ratio of cluster-wide power consumption and throughput, so that our system can consume the lowest energy per request at each point in time.

Unfortunately, designing such a server is a non-trivial task when nodes are highly heterogeneous. To tackle this design task, we develop analytical models that use information about the expected load on the cluster to predict the overall throughput and power consumption, as a function of the request distribution. Using the models and an iterative optimization algorithm, we can evaluate a large space of configurations and distributions to find the one that minimizes the power/throughput ratio for each level of offered load intensity. Our approach is general and can implement all previous distribution and reconfiguration approaches.

As a proof-of-concept implementation, we apply our models to a Web server cluster serving both static and dynamic content. The servers cooperate to implement the request distribution found by the optimization algorithm. Since the optimization step is typically time-consuming, we run it off-line and store the best configuration and request distribution found for each load intensity on the node that runs a master process. Periodically, the servers send their load information to the master, which then computes the total load imposed on the system. With this information, the master looks up the best request distribution and configuration for the current load and commands the nodes to adjust accordingly.

Our validation experiments running on a cluster of blade and traditional nodes show that the models are accurate for a wide range of distributions; modeled throughputs are within 6% of the actual measurements, whereas modeled powers are within 1.3% of the measured results. The experimental results with our model-based server running on the heterogeneous cluster show that we can consume 42% less energy than a traditional, energy-oblivious server with only a 0.35% loss in throughput. The results also show that our server conserves 45% more energy than an energy-conscious server proposed for homogeneous clusters [23].

Based on our results, we conclude that servers need to self-configure intelligently on heterogeneous clusters for an ideal tradeoff between energy and performance.

The remainder of the paper is organized as follows. The next section details our motivation using a few simple examples. Section 3 describes our modeling and optimization approach. Section 4 describes our model-based server and how we use our analytical framework to guide the decisions the system makes. Section 5 presents our methodology and experimental results. Section 6 discusses the related work. Finally, section 7 draws our conclusions.

2. MOTIVATION

In this section, we motivate the need for a model-based approach to designing performance and energy-efficient server clusters. We organize the section around the key questions involved in request distribution and cluster configuration. Throughout the section, we use a simple example to demonstrate why our design approach is appropriate. For clarity, we disregard several overheads.

Type	Metric	Resource 1	Resource 2
A	Max Throughput	3200 units/s	20000 units/s
B	Max Throughput	800 units/s	50000 units/s
A	Max Power	120 Watts	5 Watts
B	Max Power	25 Watts	10 Watts

Table 1: Example throughputs and powers per node type. The values are representative of two real systems, assuming resource 1 is the CPU and resource 2 is the disk: node A represents a Fujitsu RX 100 Monoprocessor system (3.2 GHz Pentium 4 and 7200 rpm IDE disk), whereas node B represents one of our own machines (800 MHz Pentium 3 and 2 10K rpm SCSI disks).

Type	Resource 1	Resource 2	Fraction of Requests
α	1	100	0.87
β	20	1	0.13

Table 2: Example resource needs and fraction of requests per request type. Requests of type α access files, whereas requests of type β execute small CGI scripts.

Question 1: How should we distribute requests to optimize throughput? The most common approach to request distribution is to have content-oblivious front-end devices that distribute the clients' requests across the server cluster using a policy such as round-robin, weighted round-robin, or least-connections. The two latter approaches recognize that the cluster nodes may be heterogeneous. However, even these two approaches distribute requests based solely on the relative performance of the bottleneck resource (e.g., the CPU or the disk subsystem) at each node. The same resources at other nodes may be severely under-utilized. For the most efficient resource usage (and, thus, highest throughput), their request distribution would have to consider the different request types and their approximate resource requirements. Systems that use content-aware front-ends can consider request types, but typically only do so to segregate requests into separate cluster partitions, again under-utilizing resources.

We argue that the best (in terms of throughput and generality) approach to request distribution in heterogeneous server clusters involves taking the characteristics of different node and request types into account explicitly. Using this information, *the set of individual cluster resources can be pooled together and scheduled at a fine grain, at the request level.* The request distribution can be effected either by (1) content-aware front-ends that implement a possibly different request distribution for each request type or (2) content-oblivious front-ends that implement a single request distribution scheme with re-distribution by the server nodes themselves. We favor approach (2) because it avoids the lower performance of content-aware front-end devices, which need to accept TCP connections and inspect requests before they can be forwarded to back-end nodes. We illustrate approach (2) with the following example.

The top part of table 1 lists the throughput of two resources of two types of nodes. The resources are generic, so let us assume that the throughput is in "units"/second. Think of resource 1 as the CPU and resource 2 as the disk, for example. In this case, the throughput of resource 1 could be measured in instructions/second and the throughput of resource 2 could be measured in KBytes/second. Let us further assume that there are two types of requests, α and β, each of which is responsible for a fraction of the request stream. The resource needs of the different request types are described in table 2. So, for example, each request of type α requires 1 unit of resource 1 and 100 units of resource 2. You can think of type α as requests for a large file and type β as requests for a small CGI script; both file and script are replicated at the two nodes.

For a system with a content-oblivious, round-robin front-end and one node of each type, we can only get a maximum throughput of 460 requests/second, since the performance of nodes A and B is limited by resource 2 and 1, respectively. If we now have node B send all requests of type β that it receives to node A, and node A send 43.1% requests of type α that it receives to node B, we can improve the cluster throughput to 803 requests/second. This shows how intra-cluster cooperation has the potential for better resource utilization in heterogeneous systems.

Question 2: How should we distribute requests and configure the cluster to optimize power/throughput? The request distribution approach we just discussed seeks to optimize throughput, but disregards power and energy altogether. Previous work has considered request distribution optimizing both throughput and energy [10, 15, 23, 24], but in the context of homogeneous clusters. Their approach is to reconfigure the cluster to the smallest size (by turning some of the nodes off) needed to satisfy the current offered load and distribute the requests evenly across the active nodes. Because all nodes and resources are treated as consuming the same amount of power, this approach can obviously be inefficient for heterogeneous systems.

Our own cluster of traditional and blade nodes is a good example of power heterogeneity. Each traditional node consumes from 70 to 94 Watts, whereas each blade consumes from 40 to 46 Watts, depending on utilization. To complicate decisions further, the first blade to be turned on consumes an extra 150 Watts, as it incurs the overhead of three (underloaded) power supplies, three fans, and other infrastructure shared by the other blades.

Another simple approach that may lead to inefficiencies is to reconfigure the cluster under light load by turning off nodes with decreasing "power efficiency", i.e. maximum power/maximum throughput ratios. Properly ordering nodes with respect to power efficiency is difficult for two reasons: (1) nodes can exhibit different relative orderings depending on the characteristics of the workload (particularly, the resource requirements and the load intensity); and (2) the ordering may depend on the rest of the configuration, as in the case of our own system (having a blade on is more power efficient than having a traditional node on, unless it is the first blade). Finally, even when it is possible to determine a proper node ordering, an even distribution of requests across the active nodes can under-utilize resources as we argued above.

We can eliminate the possibility of inefficiencies by extending our approach to consider the power characteristics of the different types of nodes (and resources) explicitly. We can see this by going back to the example of tables 1 and 2.

The bottom part of table 1 lists sample maximum power consumptions for each resource. Let us assume that the resource power consumption varies linearly with utilization. Under heavy load and the best request distribution we found for throughput, the power/throughput ratio would be 0.145 Joules/request (116 Watts/803 requests/second). However, given that resource 1 in node A is power-hungry, we can get a better ratio by reducing the number of requests sent from node B to node A. For example, if we change the above configuration so that machine B does not forward any requests, the ratio becomes 0.137 Joules/request (57 Watts/416 requests/second). This shows that optimizing for energy and performance is not straightforward.

3. ANALYTICAL MODELS

In this section, we describe the analytical models and optimization procedure that allow us to determine the best cluster configuration and request distribution for a heterogeneous Web server cluster. This machinery works for homogeneous clusters as well but is unnecessary for these systems, since configuration and distribution are well-understood issues for them.

As we have mentioned, to determine the best configuration and distribution for heterogeneous clusters, we need to consider all resources, their performance and power consumptions, at the same time. We cast this as an optimization problem: *find the request distribution (1) from clients to servers and (2) among servers, in such a way that the demand for each resource is not higher than its bandwidth, and we minimize a particular metric of interest.* The cluster configuration comes out from the request distribution; we can turn off nodes that are not sent any requests.

In this paper we define the metric of interest to be cluster-wide power consumption divided by throughput. Thus, the request distributions, the total power, and the maximum throughput for each cluster configuration are the *unknowns* in the models, whereas the resources and offered workload characteristics are the *inputs* to the models. In particular, the resource-related inputs are the bandwidth of each resource, the power consumed by each node when it is idle, and a linear factor describing how the power consumption changes as a function of resource utilization. The resource types we consider are processor, network interface (divided into incoming and outgoing bandwidth), disk, and a software resource representing the maximum number of sockets each server can concurrently open with its clients. Obviously, this software resource does not consume power. For simplicity, we assume that each node contains a single resource of each type. The workload-related inputs are the expected file popularity, the expected average size of requested files, the fraction of requests for static files in the workload, the characteristics of the dynamically generated content, and the resource consumptions per request type as well. These inputs can be determined from a sample trace of requests and an analysis of the programs that generate the dynamic content. Table 3 summarizes the unknowns and inputs of our models.

In the next few subsections, we describe our models for estimating throughput and power, as a function of the request distributions. We also describe how to instantiate the inputs to the models. Finally, we describe the optimization procedure that solves the equations that define the models.

	Description	Representation in models
Inputs	Bandwidth of each resource	C matrix
	Resource cost for performing a request locally	L matrix
	Resource cost for sending remote requests	S matrix
	Resource cost for accepting remote requests	A matrix
	Idle power of each node	B vector
	Power factor of each resource	M matrix
	Partition of request stream into types	F matrix
Unknowns	Request distributions	D vector and R matrix
	Maximum system throughput	$max\_thruput$
	Total system power	$overall\_power$

Table 3: Summary of model inputs and unknowns.

3.1 Modeling Request Distributions

To formalize the problem, we will represent the distributions using a vector and multiple matrices. We use a distribution vector, D, to represent the fraction of the requests coming from the clients that will be sent to a certain node. Thus, D_i is the fraction of the requests from clients that server i will receive.

Requests can be of different types as well. For example, a Web server can get requests for files that are in memory, files that are on disk, or dynamically generated content. We define a matrix F to describe the partitioning of the request stream into these request types for each node. Thus, F_i^t is the fraction of requests of type t that are directed to node i by the front-end devices.

When nodes cooperate, we must also take the intra-cluster request distribution into account. We represent this distribution with one matrix, R^t, per request type t. Each element R_{ij}^t represents the fraction of requests on node i of request type t that are serviced by node j. In a non-cooperative server environment, $\forall i R_{ii}^t = 1$, while all other terms would be 0.

The resource consumptions for each request type are represented with 3 other matrices. We represent the amount of resource r used by a request of type t that node i must expend if the request is serviced locally as L_i^{rt}. Even if node i sends the request to node j, it may still expend some resources, denoted by S_{ij}^{rt}. If node i instead receives a request from node j, the resource usage is represented as A_{ji}^{rt}.

The costs in the S, A, and L matrices are average numbers of bytes per request for all devices. The exceptions are the CPU device and the software resource. The CPU cost per request can be represented in average number of instructions, average number of memory bytes used, or some other relevant metric. The software resource costs can also be arbitrarily represented. For example, we can represent the cost per request of consuming a socket as the average number of instructions executed or the average time elapsed between the opening and closing of the socket.

3.2 Modeling Resource Utilization

To determine the utilization U_i^r (in bytes/second) of each resource r at each node i, we need to sum the cost due to the fraction of requests serviced locally, the cost of sending requests to other nodes, and the cost of servicing requests on behalf of other nodes. This sum then needs to be multiplied by the total number of requests being served per second (the current throughput of the server cluster, *thruput*). This is expressed as:

$$u_i^r = \sum_t^{types} \left(D_i F_i^t R_{ii}^t L_i^{rt} + D_i F_i^t \sum_j^{nodes} R_{ij}^t S_{ij}^{rt} + \sum_j^{nodes} D_j F_j^t R_{ji}^t A_{ji}^{rt} \right)$$

$$U_i^r = thruput \times u_i^r \tag{1}$$

3.3 Determining Max Throughput and Power

We can use U to determine the maximum throughput and overall power consumption of a configuration.

Maximum throughput. We find the maximum throughput achievable by the server cluster by determining the bottleneck resource(s) in the system. The bottleneck(s) will be such that $U_i^r = C_i^r$ under maximum throughput, where each element C_i^r is the capacity of resource type r on node i. Thus, we define the maximum throughput as:

$$max\_thruput = \min_{\forall r,i} \frac{C_i^r}{u_i^r} \tag{2}$$

Overall power. The power consumed by each *hardware* resource generally relates to the utilization of the resource. We use a linear model of resource power, such that the power consumed by each node i is:

$$P_i = B_i + \sum_r M_i^r \times \left(\frac{U_i^r}{C_i^r} \right) \tag{3}$$

where B_i is the base power consumed by node i when it is idle, and M_i^r is the measure of power of the resource r at full utilization.

Finally, we define the power consumed by the server cluster, $overall\_power$, as the sum of the power consumed by all hardware resources in the system:

$$overall\_power = \sum_i P_i \tag{4}$$

3.4 Instantiating the Inputs to the Models

The C, L, S, A, B, and M matrices are inputs to our models. Determining the correct values for them is not always simple. To determine the C^r, L^r, S^r, and A^r matrices, we run microbenchmarks on two machines to exercise the resource with requests of different types and/or sizes and measure their performance.

C^{disk} is perhaps the hardest vector to instantiate. More specifically, our disk microbenchmark goes through several rounds of random reads of fixed size, going from 4-KByte accesses to 128-KByte accesses (the largest chunk our server will read at once from disk) in steps of 4 KBytes. We run this microbenchmark for each of the different disks in the heterogeneous cluster. Besides these data, we also need the average size of disk accesses on each different node. Unfortunately, the average size of disk accesses (and the memory cache hit ratios) is not readily available off-line. To approximate that size, we use information about the memory size and the expected file popularity coming from a representative request trace. In more detail, we rank the files in descending order of popularity and add the file sizes until the combined size exceed the memory size. Accesses to files that are more popular than this threshold are (optimistically) expected to be cache hits. With information about the files that cause misses, we can easily compute the average disk access size. We have successfully taken this same approach to approximating hit rates and disk access sizes in our previous Web server modeling work (e.g., [6, 9]).

Instantiating the B vector involves measuring the power consumed by each different node when it is completely idle.

Determining the M^r vectors is somewhat more involved. For each different node i we determine M_i^{disk}, M_i^{CPU}, and M_i^{net} by running several microbenchmarks, each of which exercises one of these three resources in different ways. For each microbenchmark, we record average disk, CPU, and network interface utilization statistics, as well as average overall power. The utilization data forms an $m \times 3$ matrix called E, where m is the number of microbenchmarks. The power data (actually, the subtraction of the power data by the base power of the node) forms an $m \times 1$ vector called W. We then compute a least-squares fit to determine the 1×3 vector X for which $EX = W$. The resulting X_1, X_2, and X_3 are M^{disk}, M^{CPU}, and M^{net}, respectively, for the node. This same approach has been used successfully by other groups [19].

3.5 Finding the Best Distributions

To optimize power consumption and throughput, we need to find the request distributions D and R^t for each type t (and consequently the best cluster configuration), such that $U_i^r \leq C_i^r\ \forall r, i$ and we minimize some metric that combines throughput and power. In this work, we decided to give equal emphasis to throughput and power, so our optimization procedure attempts to minimize $overall\_power/thruput$, under the constraint that $max\_thruput \geq thruput$. Thus, a decrease in power consumption (with fixed throughput) has the same effect as an equal increase in throughput (with fixed power). If the offered load ever becomes greater than the maximum achievable throughput by any configuration and distribution, our goal becomes to maximize $max\_thruput$ so that we lose the fewest requests possible.

We could attempt to solve the system of equations defined by our model directly. Unfortunately, this is a complex proposition, because to determine utilizations we need to multiply unknown distributions, making the problem nonlinear. Moreover, cluster power and throughput are functions of these distributions and vice-versa, so the problem is also recursive. These characteristics suggest that a numerical and iterative optimization technique, such as simulated annealing or genetic algorithms, is appropriate.

We use simulated annealing. The annealing works by trying to iteratively optimize a "current solution", i.e. values for the distributions D and R, starting from initial guess values for these matrices. The initial distributions are set such that the D_i are all the same and the R^t matrices specify no intra-cluster cooperation.

A candidate solution is generated by modifying two elements of the same (randomly chosen) vector/matrix at a time; one element is decreased by a randomly chosen amount, while another is increased by the same amount. The rows are then normalized. A candidate also has to satisfy the constraint that $U_i^r \leq C_i^r\ \forall r, i$.

Evaluating a candidate solution involves computing its $overall\_power/thruput$ measure and comparing it to the corresponding measure of the current solution. If the candidate solution produces a measure that is smaller than the current minimum, it becomes the new current solution. If it does not, it might still become the new current solution, but with a decreasing probability. After evaluating each candidate solution, a new one is generated and the process is repeated. The number of iterations is determined by a "temperature" parameter to the annealing algorithm. More details about simulated annealing can be found in [17].

Finally, we speed up the search process significantly by treating all nodes of the same type together. This makes the number of calculations proportional to the square of the number of node types, as opposed to the square of the actual number of nodes in the cluster.

4. MODEL-BASED COOPERATIVE SERVER

We developed a cooperative Web server that uses the results of our modeling and optimization approach to configure the cluster and distribute requests.

Requests are sent to the individual Web servers according to the D vector computed by our model and optimization procedure. Each server either serves a requests it receives locally or forwards it to another server over a persistent TCP connection. Requests for static content are always served locally. The R matrix determines where to serve the dynamic-content requests. Once a forwarded request completes, the reply is sent back to the server that received the original request. This server then replies to the client.

Each server sends its delivered load information to a master process every 10 seconds. The master process accumulates this information and smooths it by computing an Exponentially Weighted Moving Average (EWMA) of the form $avg\_load = \alpha \times current\_load + (1-\alpha) \times previous\_load$. Our EWMA uses an α value of 0.4. Periodically, the master decides whether the request distribution (and configuration) needs to be changed. We refer to the fixed time in between decisions as the *reconfiguration interval*. The master's decision is based on its prediction of what the average load will be in the end of the next reconfiguration interval. The prediction uses a first order derivative of the current average load and the last average load of the previous interval, i.e. $predicted\_load = avg\_load + (avg\_load - last\_avg\_load)$. To avoid undershooting and losing requests, the system never predicts load that is lower than the current value. Based on the predicted load, the master can determine the best request distributions, as predicted by our model and optimization procedure. If the request distribution needs to be changed, the master process commands the servers and/or the front-end devices to adjust accordingly.

Because our optimization procedure is time-consuming, the best distributions for each amount of load are computed off-line. We store a large pre-computed table of best distributions (D and R arrays) and throughputs entries, $\{thruput, best\_distribution\}$, on the local disk of the node running the master process. The master finds the best distributions for a certain load by looking up the entry listing the lowest throughput that is higher than the load.

5. EXPERIMENTAL RESULTS

In this section, we describe our experimental methodology, validate the models, and compare different server systems.

5.1 Methodology

Cluster hardware. Our cluster is comprised of 4 Linux-based PCs (each with an 800-MHz Pentium III processor, local memory, two SCSI disks, and a Fast Ethernet interface) and 4 Linux-based blade servers (each with a 1.2-GHz Celeron processor, local memory, an IDE disk, and a Fast Ethernet interface) from Nexcom [20]. The two sets of machines are connected by a Fast Ethernet switch.

The PCs (herein also called traditional nodes) consume about 70 Watts when idle and about 94 Watts when fully utilized. In contrast, each blade consumes about 40 Watts when idle and 46 Watts when fully utilized. However, the base power consumption of the blade chassis is substantially higher, 150 Watts, due to three power supplies, the power backplane, three fans, and the KVM (keyboard-video-mouse) controller.

All PCs and the blade system are connected to a power strip that allows for remote control of its outlets. The systems can be turned on and off by sending commands to the IP address of the power strip. The blades can be controlled independently as well, by sending commands to the KVM controller. Shutting a node down takes 45 seconds (traditional nodes) and 21 seconds (blades); bringing any node back up takes about 80 seconds.

The total amount of power consumed by the cluster is monitored by a multimeter connected to the power strip. The multimeter collects instantaneous power measurements (several thousand per second) and sends these measurements 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.

Server software. We experiment with three servers: (1) a conventional, energy-oblivious server; (2) an energy-conscious server that was developed for homogeneous clusters; and (3) our energy-conscious, model-based Web server for heterogeneous clusters. The conventional server, called "Energy-Oblivious", is similar to Flash [22] and involves no cooperation between nodes.

The energy-conscious server for homogeneous clusters, "Adaptive", is based on the same code as Energy-Oblivious, but includes a master process that collects load information from the servers and decides on the minimum cluster configuration that can satisfy the offered load. The master uses a PID feedback controller to determine how to change the configuration. When there are enough spare resources, the master forces a node to turn off. When more resources are needed, the master turns a node on. Because the server assumes a homogeneous cluster, the master randomly selects which nodes to turn on and off. In addition, requests are distributed evenly across the active nodes. More details about Adaptive can be found in [23].

Our model-based cooperative Web server, "Model Adaptive", is based on the same code as Adaptive, but includes dynamic-content request forwarding and uses our modeling and optimization machinery, as described in the previous section. For a fair comparison, the frequency of load information exchanges and reconfiguration decisions in our experiments is kept the same in both adaptive systems.

The master process in the adaptive systems remains blocked most of the time, so it can run on any active node without a noticeable increase in energy consumption. For simplicity, we run it alone on a 9th node.

Clients. Besides our main cluster, we use 7 x86-based machines to generate load for the servers. These clients connect to the cluster using TCP over the Fast Ethernet switch. In our experiments with the Energy-Oblivious and Adaptive servers, the client requests are distributed across cluster nodes in one of two ways: (1) randomly to mimic a large number of users and a Round-Robin DNS policy; or (2) according to a Least-Connections policy that continuously tries to balance the load by sending each request to the node with the smallest number of open connections at the time. We refer to these approaches as "RR" and "LC", respectively. Model Adaptive distributes the client requests according to the D vector (and forwards requests internally according to R).

For simplicity, we did not use a front-end device that would enforce the RR, LC, and D distributions. Instead, the client themselves distribute their requests according to the policies. Note that, although the clients do not coordinate their requests in the LC policy, the load is still properly balanced, since the clients' local views approximate the nodes' behaviors in steady state. In our Model Adaptive experiments, when the request distribution needs to change, the master process sends the D vector to each client over pre-established socket connections. The clients send requests to the available nodes in randomly, but obey the vector.

The clients issue their requests according to a trace of the accesses to the World Cup '98 site from June 23rd to June 24th, 1998 (WC'98). We run two types of experiments with this trace: validations of our models and self-configuration experiments. In our model validation experiments, the clients disregard the timing information in the trace and issue new requests as soon as possible. In our self-configuration experiments, the clients take the timings of the trace into account, but accelerate the replay of the trace 20 times to shorten the experiments to 7500 seconds. Regardless of the type of experiment, requests that are not serviced within 10 seconds are considered lost.

We also modified the trace in two other ways. The first modification replaces 30% of the static requests issued with dynamic requests to simulate a CGI load. For simplicity, we used a single CGI script that does nothing else but produce a short reply. The accesses to this script drastically reduce the throughput of our server cluster, so we also attenuated the trace by a factor of 50. (This was done so that we could still observe the load changes in the trace while keeping the run time to approximately 2 hours.) Under these assumptions, the CPU and the software resource become the main bottlenecks in the system.

Given our accelerated trace, we set the reconfiguration interval of the adaptive servers to a minimum of 120 seconds

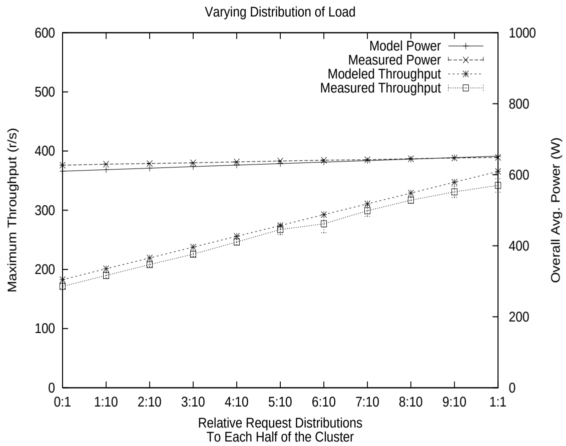

Figure 1: Modeling and experimental results for throughput and power, as a function of D.

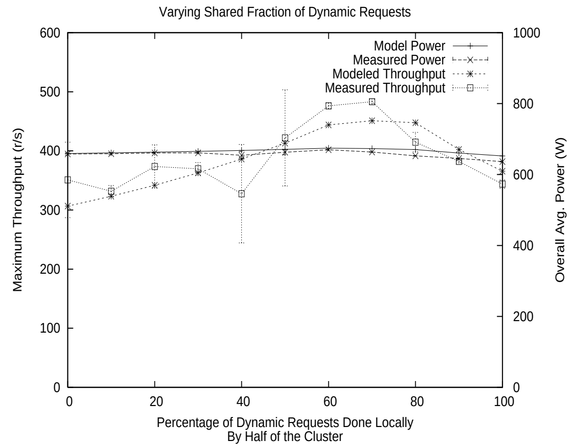

Figure 2: Modeling and experimental results for throughput and power, as a function of R.

between changes. This relatively short interval allows time for a rebooted node to settle and the servers to react quickly to variations in offered load. In practice, the reconfiguration interval can be substantially longer however, since real-life variations in load intensity occur over periods of tens of minutes, i.e. much more slowly than in our experiments. For WC'98, for example, we could have an interval of 2400 seconds in real time. In fact, we expect the energy and time overheads of reconfigurations to be small in practice.

5.2 Validation of the Models

Figures 1 and 2 show our validation results for the WC'98 trace running on our 8-node heterogeneous cluster, as a function of changes in the D and R distributions, respectively. Both figures show modeled and measured server cluster throughput (in requests/second) and power consumption (in Watts). Each point in the figures is an average of two runs; the vertical bars show the ranges of values we observed in the different runs. In figure 1, each point on the X-axis represents a different weighting of the distributions of the requests sent from the clients to the servers. For example, at “2:10”, 2 requests are sent to the traditional nodes for every 10 requests that are sent to the blades. In this case, the R matrix determines no cooperation between nodes.

In figure 2, each point on the X-axis represents the fraction of dynamic requests in the WC'98 trace that the blades execute locally; the others are sent to the traditional nodes. For example, at X=0, 100% of CGI requests received by the blades are sent to the traditional nodes. The requests coming from the clients are distributed to all nodes evenly.

These figures demonstrate that our models are very accurate for WC'98. The modeled throughput has an average error of 6% as compared to the measured results, with a maximum error of 18%. We can also see that varying request distributions has a significant effect on throughput, but only a minor effect on power. Power does not vary noticeably for two reasons: (1) all nodes are active (and highly utilized) throughout the experiments; and (2) resource utilization has a small effect on power, since most of the power consumed by the nodes and their resources is fixed, i.e. the base power. This leads to a small average and maximum error of 1.3% and 2.7%, respectively.

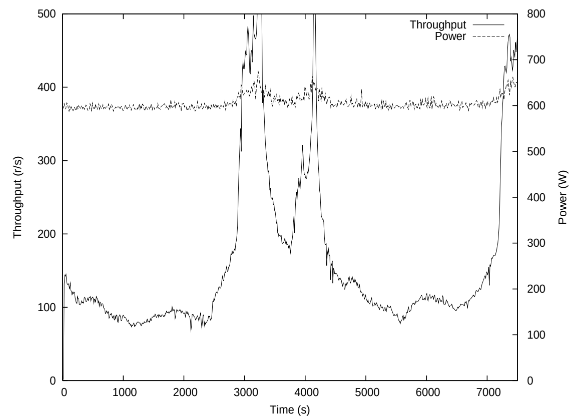

Figure 3: Throughput and power of Energy-Oblivious-LC.

5.3 Comparing Server Systems

Figures 3 to 5 show throughput and power for Energy-Oblivious-LC, Adaptive-LC, and Model Adaptive, as a function of time.

Let us discuss energy first. Figure 3 shows that Energy-Oblivious-LC consumes roughly the same amount of power throughout the experiment; non-trivial variations only occur during the three load peaks. In contrast, figures 4 and 5 demonstrate that the Adaptive-LC and Model Adaptive systems can nicely adjust the cluster configuration, according to the offered load. For instance, during the load valleys, only 2 or 3 nodes are required to serve the offered load; the other nodes can be turned off. As a result of the reconfiguration, the two systems accrue substantial energy savings.

Note though that Adaptive-LC leads to substantially higher power consumption than Model Adaptive during the load valleys. As a result, Model Adaptive consumes 42% less energy than Energy-Oblivious during this experiment, whereas Adaptive-LC only consumes 29% less energy. Comparing the amount of energy saved by Adaptive-LC (1.30 MJ) and Model Adaptive (1.89 MJ) directly, we find that the latter system conserves 45% more energy than the former.

The reason for the inefficient behavior of Adaptive is that it treats a heterogeneous system as if it were homogeneous.

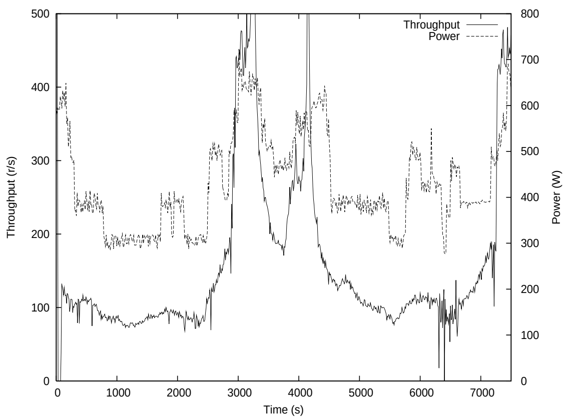

Figure 4: Throughput and power of Adaptive-LC.

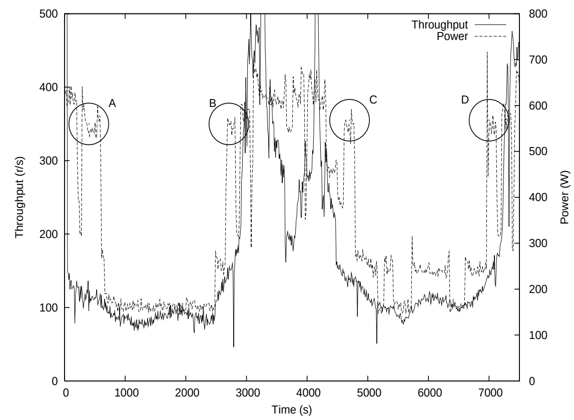

Figure 5: Throughput and power of Model Adaptive.

For instance, it treats a single-node blade system with an idle power consumption of 190 Watts the same as a traditional node that consumes only 70 Watts when idle. In effect, Adaptive selects the nodes to be part of the cluster configuration randomly, using feedback control to achieve the required throughput. In this experiment, the 3-node configuration of Adaptive-LC during the load valleys includes only one blade and, hence, incurs the unamortized fixed power consumption of the entire blade system.

The transitions between configurations are also markedly different between Adaptive-LC and Model Adaptive. Adaptive-LC changes the cluster configuration one node at a time. In contrast, Model Adaptive may decide to change the configuration completely, by turning several nodes off and several nodes on. These transitions are marked with letters A, B, C, and D in figure 5. The high energy consumed at these points results from having to turn the new nodes on, before the nodes in the current configuration can be turned off. At point A, for example, Model Adaptive needs to turn on 3 traditional nodes before turning off the 4 blades that comprise the current configuration.

Figure 6 shows the complete list of cluster configurations that Model Adaptive goes through, as a function of time. The stacked symbols illustrate the actual configurations, with each "+" representing a traditional node and each "X" representing a blade node. The vertical lines illustrate the times when the transitions are performed. When two consecutive stacks are the same, the only change is in the distribution of requests (D and R).

The most interesting observation from this figure is that several of the transitions only affected the request distribution. More specifically, as the offered load approaches the maximum throughput achievable by a configuration, the system tends to reduce the amount of inter-node cooperation since the bottleneck components will be highly utilized locally at all nodes. When the offered load is decreasing, the system tends to increase cooperation by shifting requests to the components that are most power-efficient.

In terms of performance, we see in table 4 that Adaptive-LC drops more than twice as many requests as Model Adaptive, due to the overhead of reconfigurations. Nevertheless, both systems drop a negligible percentage of the requests. In contrast, Adaptive-RR drops more than 30% of the requests due to the reconfigurations, making it effectively useless. For this reason, we do not show figures for Adaptive-RR.

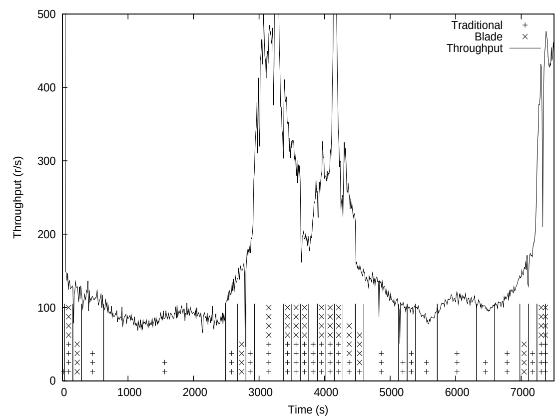

Figure 6: Throughput and configuration of Model Adaptive.

Discussion. Model Adaptive behaves well in terms of energy conservation and performance. The energy savings it achieves are mostly due to selecting the best cluster configuration for each load intensity level. However, significant configuration changes can consume substantial energy and decrease savings. In our experiments, the impact of the energy associated with these transitions is actually magnified, as we accelerate the trace and thus have less time to amortize the transition overheads.

Cooperation does not provide substantial gains in our experiments because the power consumption of our cluster nodes is dominated by their base powers. In contrast, the base power of more recent machines is a substantially smaller fraction of their maximum power consumption. We expect this trend to continue in future systems, especially as they become more power-aware.

6. RELATED WORK

Energy conservation research for server clusters. A few recent papers [8, 10, 15, 23, 24] deal with energy conservation for server clusters. Pinheiro *et al.* [23] and Chase *et al.* [10] concurrently proposed cluster reconfiguration to conserve energy. Elnozahy *et al.* [15] evaluated different combinations of cluster reconfiguration and dynamic voltage scaling. Rajamani and Lefurgy [24] studied how to improve the cluster reconfiguration technique by using spare servers

System	Energy (MJ)	Requests Serviced	Requests Lost	Drop Rate (%)
Energy-Oblivious-RR	4.54	1264424	4117	0.32
Energy-Oblivious-LC	4.54	1267475	0	0.00
Adaptive-RR	2.65	859320	408651	32.23
Adaptive-LC	3.24	1256091	11436	0.90
Model	2.65	1262736	4417	0.35

Table 4: Summary of energy consumption and performance degradation for WC'98 trace.

and history information about peak server loads. Finally, Elnozahy *et al.* [14] considered dynamic voltage scaling and request batching in Web servers. A survey of power and energy research for servers can be found in [7].

All of these previous works have been focused solely on conserving energy in homogeneous clusters. An early version of this paper [16] introduced our approach to dealing with heterogeneous clusters. This paper extends the early work by proposing more sophisticated models, the model-based cooperative server, and experimenting with workloads that include dynamic-content requests.

In a different environment, Kumar *et al.* [18] considered conserving chip-multiprocessor energy by relying on heterogeneous cores. Their approach has a similar flavor to cluster reconfiguration in that, depending on processor load (and performance requirements), a different core may execute each application or even each phase of a single application.

Modeling and optimization for servers. Carrera and Bianchini [6, 9] have successfully modeled the throughput of Web server clusters. Aron *et al.* [2] enforced resource shares in shared hosting platforms using models and optimization. Doyle *et al.* [13] have proposed a model-based approach to adjusting resource allocations again in shared hosting platforms. Hippodrome [1] applies modeling and optimization to assign load to units of a storage system. Our work is the first to use modeling and optimization to conserve energy in servers.

Request distribution in server clusters. Several request distribution strategies for homogeneous server clusters have been proposed, e.g. [11, 4, 21, 9]. One study [12] considered request distribution for distributed heterogeneous servers. Their approach was to assign a different TTL (time-to-live) to each DNS reply, according to the capacity of the selected node and/or the request rate of the source domain of the DNS request. Our approach is to distribute requests intra-cluster (without help from DNS) for energy conservation, as well as performance.

Load balancing in heterogeneous systems. A few papers do address job/task balancing/sharing in heterogeneous systems, e.g. [26, 5]. The key differences between these studies and ours are: (1) they typically focus on coarse-grain job/task scheduling, rather than on servers and request distribution; and (2) their goal is usually to improve running time, rather than increase throughput or conserve energy.

7. CONCLUSIONS

In this paper we developed a model-based cooperative Web server for heterogeneous clusters. The server is based on modeling and optimization of configurations, request distributions, throughput and power. Our experimental results demonstrated that (1) our modeling is accurate and (2) our server conserves more energy than the previously proposed system on a heterogeneous cluster, with a negligible effect on throughput.

Based on these results, we conclude that Web servers need to self-configure intelligently on heterogeneous clusters for higher energy savings. We also conclude that the style of modeling that allows our system to self-configure should be more widely applied in the systems community, given that most real systems do exhibit varying degrees of heterogeneity. In fact, our modeling framework may even be used in building systems that are heterogeneous by design.

We are currently extending our server implementation to deal with brownouts, i.e. periods during which the power budget is constrained. During these periods, our goal is to maximize throughput under the constrained power budget. The new version of the server will then have two modes of operation: `normal mode` which optimizes power/throughput making sure that the offered load is satisfied, and `constrained mode` which optimizes throughput within the available power budget. The new version will nicely leverage our modeling and optimization infrastructures.

In our future work, we plan to develop a tool to automate the process of instantiating our models. We also plan to exploit our modeling infrastructure to investigate whether servers clusters should be designed heterogeneous.

Acknowledgements

We would like to thank Gustavo Gama, Dorgival Guedes, and Eduardo Pinheiro for their comments on the topic of this paper.

8. REFERENCES

[1] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: Running Circles Around Storage Administration. In *Proceedings of the Conference on File and Storage Technologies*, January 2002.

[2] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster Reserves: A Mechanism for Resource Management in Cluster-Based Network Servers. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, June 2000.

[3] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable Content-Aware Request Distribution in Cluster-Based Network Servers. In *Proceedings of USENIX'00 Technical Conference*, June 2000.

[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] A. Bevilacqua. A Dynamic Load Balancing Method on a Heterogeneous Cluster of Workstations. *Informatica*, 23(1):49–56, March 1999.

[6] R. Bianchini and E. V. Carrera. Analytical and Experimental Evaluation of Cluster-Based WWW Servers. *World Wide Web journal*, 3(4), December 2000.

[7] R. Bianchini and R. Rajamony. Power and Energy Management for Server Systems. *IEEE Computer*, 37(11), November 2004.

[8] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. The Case for Power Management in Web Servers. In Graybill and Melhem, editors, *Power-Aware Computing*. Kluwer Academic Publishers, January 2002.

[9] 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.

[10] J. Chase, D. Anderson, P. Thackar, A. Vahdat, and R. Boyle. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the 18th Symposium on Operating Systems Principles*, October 2001.

[11] Cisco LocalDirector. http://www.cisco.com/.

[12] M. Colajanni, V. Cardellini, and P. S. Yu. Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, May 1998.

[13] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. M. Vahdat. Model-Based Resource Provisioning in a Web Service Utility. In *Proceddings of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.

[14] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy Conservation Policies for Web Servers. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.

[15] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy-Efficient Server Clusters. In *Proceedings of the 2nd Workshop on Power-Aware Computing Systems*, February 2002.

[16] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini. Self-Configuring Heterogeneous Server Clusters. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, September 2003.

[17] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[18] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *Proceedings of the 36th International Symposium on Microarchitecture*, December 2003.

[19] M. Martonosi, D. Brooks, and P. Bose. Power-Performance Modeling and Validation. In *Tutorial given at the International Conference on Measurement and Modeling of Computer Systems*, June 2001.

[20] Nexcom International. http://www.nexcom.com.tw/.

[21] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems*, October 1998.

[22] V. Pai, P. Druschel, and W. Zwaenepoel. Flash: An Efficient and Portable Web Server. In *Proceedings of USENIX'99 Technical Conference*, June 1999.

[23] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Dynamic Cluster Reconfiguration for Power and Performance. In L. Benini, M. Kandemir, and J. Ramanujam, editors, *Compilers and Operating Systems for Low Power*. Kluwer Academic Publishers, August 2003. Earlier version published as "Load Balancing and Unbalancing for Power and Performance" in Proceedings of the International Workshop on Compilers and Operating Systems for Low Power, September 2001.

[24] K. Rajamani and C. Lefurgy. On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, March 2003.

[25] Tao Yang. Personal communication. October 2003.

[26] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software - Practice and Experience*, 23(12), 1993.