Coordinated Management of Power Usage and Runtime Performance

Malgorzata Steinder, Ian Whalley, James E. Hanson and Jeffrey O. Kephart
IBM Thomas J. Watson Research Center
Hawthorne, NY 10532
Email: {steinder,inw,jehanson,kephart}@us.ibm.com

Abstract—
With the continued growth of computing power and reduction in physical size of enterprise servers, the need for actively managing electrical power usage in large datacenters is becoming ever more pressing. By far the greatest savings in electrical power can be effected by dynamically consolidating workload onto the minimum number of servers needed at a given time and powering off the remainder. However, simple schemes for achieving this goal fail to cope with the complexities of realistic usage scenarios. In this paper we present a combined power-and performance-management system that builds on a state-of-the-art performance manager to achieve significant power savings without unacceptable loss of performance. In our system, the degree to which performance may be traded off against power is itself adjustable using a small number of easily-understood parameters, permitting administrators in different facilities to select the optimal tradeoff for their needs. We characterize the power saved, the effects of the tradeoff between power and performance, and the changes in behavior as the tradeoff parameters are adjusted, both in simulation and in a sample deployment of the real system.

I. Introduction

Energy consumption within data centers has become a major concern for businesses and governments worldwide. As server power densities continue to increase, the cost of the energy used to power and cool a server during its lifetime is becoming a significant fraction of the cost of the hardware itself.

Given the widely-recognized value of reducing power consumption in data centers, a large variety of power-saving measures have been proposed, some of which are being implemented and marketed. Chips have been equipped with low-power states that are manipulable by firmware or software (e.g. by the operating system or middleware), and low-power servers that exploit these states are being designed. At the facilities level, more efficient power supplies and cooling systems are being developed and marketed. Kephart et al. [1] and Chase et al. [2] have previously argued that workload consolidation with powering off spare server machines is the most effective way to conserve electrical energy. Our study of power usage curves as a function of CPU usage performed on IBM blade center hardware confirms this argument. For example, we found out that on an IBM LS20 dual-core AMD Opteron blade as much as 80% of server power usage is incurred in idle state. Only 20% may be controlled by other techniques while the server is powered on. In a much more extensive study Fan et al. [3] found a similar result.

In this paper, we describe an application placement controller (APC) that consolidates workload to achieve substantial power savings. It is an augmentation of an existing commercial APC that manages systems to specified performance objectives. We address a key challenge: how to place applications so as to meet combined power and performance objectives. One straightforward approach to addressing the tradeoff is to give blanket priority to performance by consolidating workload onto the minimum number of machines sufficient to serve it, and turning off the unused machines. However, much greater energy savings are possible if we allow application performance to be somewhat degraded. In the system we describe here, an application's performance is measured relative to a service level agreement (SLA), which permits us in principle to reduce the amount of computing resources allocated to the applications—thereby saving power at the expense of performance—to the point where the SLA goals are just barely being met. However, this approach is too inflexible. Even if service contracts specify SLAs of applications, the service provider should be able to decide whether to always meet the SLAs based on their value, penalties, and the cost of running the datacenter (of which electrical power usage is an important component). Therefore, in this paper we take on the more involved problem of modeling a tradeoff between power and performance and designing a controller that optimizes application placement and server usage so as to achieve an optimal tradeoff. It bears emphasizing that we are doing this by extending an existing performance management system [4], thereby incorporating the state-of-the-art constrained performance optimization techniques it employs [5].

The remainder of this paper is organized as follows. Section II describes a formal model of the system and gives our proposed solution the power-performance coordination problem. Section III presents experimental results and simulation data characterizing how our solution performs in practice. Next is a discussion of related work, followed by conclusions and a summary of future work.

II. System and Algorithms

In this section we first describe our system, and then describe how we have augmented an placement controller to simultaneously manage power and performance.

978-1-4244-2066-7/08/$25.00 ©2008 IEEE

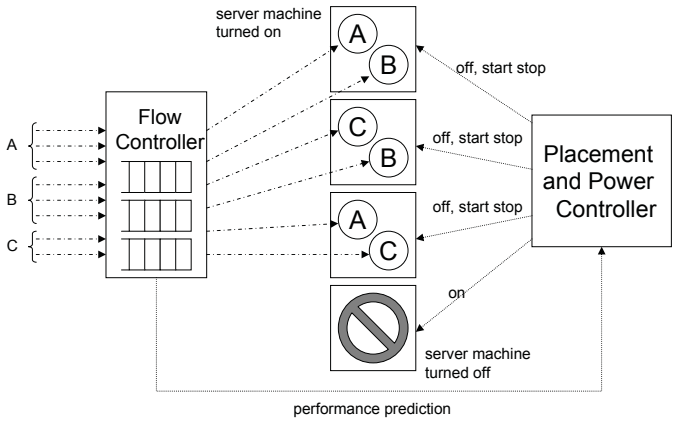

Fig. 1. System diagram

A. System description

We consider a cluster of server machines servicing requests to web applications, as shown in Figure 1. Each application may be replicated for high-availability and performance to multiple server machines. The set of all replicas of a given application (here called the *instances* of the application) constitutes its application cluster. Application clusters may arbitrarily overlap on physical machines.

Each application is accessed by a certain number of client sessions, which varies over time. Session requests arrive at an L7 proxy router that provides flow-control. The flow controller places incoming requests in queues and dispatches them from the queues so as to prevent overload on the backend server machines. The dispatching is done based on a weighted-fair round robin scheduling discipline. The dispatching weights are controlled based on application SLAs, which are defined in terms of average response time goals. The flow controller monitors and profiles the incoming request flows while estimating their average service time on each backend server, response time, number of client sessions, client think times, and CPU work factor [6]. Based on these data, the flow controller models application response time as a function of CPU speed allocation. The model is used to calculate the optimal division of server CPU capacity among applications, which translates into optimal dispatching weights for the dispatcher. The flow controller is limited by the current placement of application instances, which introduces constraints on the amount of CPU capacity that may be used by each application. The design of the controller is described in detail in [7] and [8].

The placement of applications may be controlled by starting and stopping individual instances of that application. Application placement may be changed dynamically based on workload intensity and application SLAs. In the past, we have implemented a controller that periodically evaluates the placement of applications and modifies it to better optimize the allocation of resources. To achieve this goal it collaborates with the flow controller: the flow controller provides the placement controller with application performance information. The placement controller places applications according to the same optimality criteria as those used by the flow controller.

When placing application instances, the placement controller strives to meet CPU and memory capacity constraints as well as various other constraints such as allocation restrictions, collocation restrictions, affinity constraints, minimum and maximum number of instances for each application, etc. The placement controller that achieves these objectives has been introduced in [5].

Even though they jointly solve the same optimization problem, the flow and placement controllers are separate entities working on different time scales. The flow controller readjusts queue dispatching weights every 15-30 seconds, which ensures rapid response to workload intensity changes. Placement is readjusted every several to tens of minutes, as placement changes are typically heavy-weight and time consuming.

In this paper, we extend the placement controller with the ability to consolidate application instances on a subset of available server machines so as to permit turning off the remaining machines.

B. Formal system model

To model the system, we start with a set of server machines (referred to as *nodes*) $\mathcal{N} = \{n_1, \ldots, n_N\}$. At any time, a node n_i is either powered on or off. We denote the set of nodes that are powered on by $\mathcal{N}^{\mathrm{on}}$. Each node n has CPU capacity Ω_n and memory capacity Γ_n.

We also have a set of applications $\mathcal{M} = \{m_1, \ldots, m_M\}$. The *placement matrix* P describes the way instances are distributed across nodes: $P_{mn} = i$ means that application m has i instances running on node n. For this paper, we only consider the case $i \in \{0, 1\}$. Obviously, when $P_{mn} = 0$ for all m, then node n may be turned off to save power, i.e. it may be excluded from $\mathcal{N}^{\mathrm{on}}$.

With a given placement P, each application instance is allocated a portion of the memory and CPU resources of the node on which it is running. As was noted above, the placement must obey a variety of constraints and policies that are unrelated to performance goals. Fortunately, these complications have no effect on the power-vs.-performance tradeoff with which we are concerned. In the remainder of this paper, we will therefore focus almost exclusively on the CPU allocation. The amount of CPU resource allocated to the instance of application m running on node n is denoted by ω_{mn}. We will denote by L the CPU allocation matrix giving ω_{mn} for all m and n. Clearly we have $0 \leq \omega_{mn} \leq \Omega_n$ and $P_{mn} = 0$ implies $\omega_{mn} = 0$. It is also useful to form partial sums over applications ($\omega_n^{\mathrm{node}} = \sum_m \omega_{mn}$) and over nodes ($\omega_m^{\mathrm{app}} = \sum_n \omega_{mn}$). In order to place application instances, both P and L must be computed, but from the perspective of the placement controller they are tightly linked. Henceforth, we will use L to describe application placement, as the CPU allocation has the more direct influence on power consumption.

Based on our observations of several models of Intel- and AMD-based servers, we model the electrical power usage Π

of a running node as a linear function:

$$\Pi_n(\omega_n) = p_{0,n} + p_{1,n}\omega_n^{\text{node}} \quad (1)$$

where $p_{0,n}$, the *idle-power term*, is electrical power used by node n if it is powered on but idle. We can express the total electrical power usage as a function of CPU allocation as follows:

$$\Pi(L) = \sum_{n \in \mathcal{N}^{\text{on}}} \left[p_{0,n} + p_{1,n}\omega_n^{node} \right] \quad (2)$$

As was noted above, in practice the idle-power terms dominate the CPU-dependent terms by a factor of $3-5$ or more, even when a node is running at capacity (so $\omega_n = \Omega_n$).

We define application performance vector in terms of response time as follows: $d_m = \frac{\tau_m - \text{RT}_m}{\tau_m}$, where τ_m represents the response time goal defined in the SLA for application m, and RT_m is the measured response time. Thus the performance for an application is 0 when the SLA is just being met, and 1 when the response time is perfect, i.e. equal to 0. (In this paper, the performance is based upon response time, but in general it can be any performance metric, such as throughput.)

To express the tradeoff between application performance and electrical power usage, we introduce a system utility function $U(d, \Pi)$ that depends on both the performance vector d (in which component d_m represents the performance of application m) and the total power consumption Π. Following refs. [2], [1], we assume that the utility can be separated into a performance value portion $V(d)$ and an electrical power cost portion $C(\Pi)$; the net utility is simply $U = V - C$. Since both the performance and the power consumption are determined by the CPU allocation matrix, V, C and U are correspondingly functions of L.

In general, one can envision many different plausible functional forms for $V(d)$ and $C(\Pi)$; this is a matter for the business person or system administrator to decide. For purposes of this paper, we choose specific forms that experience suggests are practical. First, we assume that the electrical power cost is linear in the power consumption, and for simplicity set $C(\Pi) = \Pi$; any constant of proportionality can be absorbed into the value function. Second, we take the value function V to depend on d. The total value function is defined as a sum over application-specific value functions: $V(d) = \sum_m v_m(d_m)$. Specific forms for the functions $v_m(d_m)$ will be discussed in the next subsection.

It might seem that we can simply compute the CPU allocation L that optimizes $U(L) = V(d(L)) - \Pi(L)$. However, there are some complications that require us to take a more subtle approach that constrains our search to a subset of the full universe of possible L. Previous authors [7] have found that making allocations according to a utility function that sums over individual application value functions unduly favors the applications that are deemed more "important", often starving applications with lower value to the point where their SLAs are violated dramatically. The resulting system behavior can be hard to predict and analyze. Moreover, system administrators tend to expect "fair" resource allocation, in which all applications are doing approximately equally well in meeting their SLA goals, i.e. the performance values d_m are roughly the same. Fairness is achieved in the existing placement controller by choosing an allocation L^* according to a max-min optimization over L:

$$L^* = \arg\max_L \min_m d_m(L) \quad (3)$$

In order to combine the fairness achieved by Eq. (3) with the power-performance tradeoff that would be achieved by optimizing over $U(L)$, we separate the problem into two parts solved by two conceptually different entities operating on different timescales: a power controller that determines which nodes are to be turned on, and a placement controller that determines how the applications are to be placed on those nodes. The second of these is essentially the existing placement controller, which uses Eq. 3 to determine L^* given a fixed set of nodes. As will be described more fully at the end of section II, the power controller considers various possible settings of $\mathcal{N}^{\text{on}}$, querying the placement controller to determine what would be the resulting $L^*(\mathcal{N}^{\text{on}})$. The power controller then computes the net utility $U(L^*(\mathcal{N}^{\text{on}})) = V(d(L^*(\mathcal{N}^{\text{on}}))) - \Pi(L^*(\mathcal{N}^{\text{on}}))$, and selects $\mathcal{N}^{\text{on}*}$ to maximize $U(L^*(\mathcal{N}^{\text{on}*}))$. The resulting solution will in general yield a somewhat lower U than would have been attainable with no constraints on L, but as will be seen in Section III-A it yields a good power-performance tradeoff that also satisfies the fairness criterion.

C. Definition of performance value function

Now we describe in further detail the application value functions $v_m(d_m)$ that compose the total value function $V(d)$. We seek functions that promote the behavior that system administrators would desire and expect, and possess tunable parameters that provide flexible controls over their shape that reflect a range of power-performance tradeoffs in an understandable manner. Moreover, we seek functions that permit us to select the desired level of application performance and to control the rate with which the function value changes as the distance between an achieved performance level and the desired performance level increases. This rate of change determines the relative importance of application performance and electrical power savings.

Based on these considerations, we choose for this paper the following function:

$$v_m(d_m) = v_{m,1} + v_{m,0}(1 - (1 + d_{m,0} - d_m)^k) \quad (4)$$

The parameters of $v_m(d_m)$ can be interpreted as follows. The value of $d_{m,0}$ configures the desired level of application performance. For example, we use $d_{m,0} = 0$ when it is sufficient to only meet SLA goals. We use $d_{m,0} = 1$ when we want the system to offer the best possible performance and only consolidate unused cycles; intermediate values allow continuous tuning between these extremes. We can also use it to implement a safety zone to prevent SLA violations as a result of the unavoidable inaccuracies of profiling and modeling techniques used by our system. Parameter k ('rigidity'),

which is greater than or equal to 1, controls the importance of achieving $d_{m,0}$ relative to saving power. A low value of k permits the system to reduce physical machines usage in violation of $d_{m,0}$. A high value of k forbids such a tradeoff, as it makes value function essentially a step function. The value of $v_{m,1}$ simply controls the vertical offset of the value function and is given here for cosmetic reasons. Since we are only concerned with finding the allocation that achieves the optimal tradeoff, and not with the absolute value of that tradeoff, $v_{m,1}$ may be set to 0 with no loss of generality.

Parameter $v_{m,0}$ controls the absolute value of the value function, which must be dependent on workload intensity. Parameter $v_{m,0}$ also controls the first derivative of v_m. To select the right $v_{m,0}$ we consider the relationship between value and power functions. The electrical power usage is a piece-wise linear function with discontinuities that occur when to increase CPU allocation a new server must be turned on. The height of the discontinuity corresponds to the power cost of the added server in idle state, $p_{0,n}$. In continuous regions, the power function increases linearly with rate $p_{1,n}$. The system utility, which is the distance between value and power curves is maximized at a point ω_0 where the first derivative of the value function is equal to $p_{1,n}$, or at any value of CPU allocation where discontinuity occurs and which is less than ω_0. When $d_m(\omega_0) < d_{m,0}$, the system will never achieve $d_{m,0}$, which is the performance level desired by a user. Hence, we must choose $v_{m,0}$ that allows $d_m(\omega_0) >= d_{m,0}$. It is easy to show that to achieve this objective, we must use $v_{m,0}$ with minimum value defined as follows (where ω'_m is the derivative of ω_m with respect to d taken at $d_{m,0}$):

$$v_{m,0} = \frac{1}{k}\omega'_m(d_{m,0}) \max_n p_{1,n} \tag{5}$$

D. Power management algorithm

In this subsection we describe the power algorithm in further detail. Recall that it determines the subset of servers that must be powered on in order to maximize system utility, and interacts with a placement controller that has been described previously [5], [9].

The optimal solution to the power-performance tradeoff problem requires us to evaluate all subsets of nodes by calculating the optimal application placement that uses a given subset of nodes and evaluating the utility of the resultant placement. It is therefore necessary to rely on heuristics. In this paper, we use the following simple approach. We search the space of machine subsets starting from the subset of machines that are currently turned on. We perform the search in two directions: by adding and by removing machines from the set. We only evaluate one choice of a machine to be added or removed. Then, we proceed to add or remove more machines. We stop when the change does not increase the utility.

Since we evaluate only one machine as a candidate to add or remove, we must be careful in selecting it. Considerations that we take into account include the following.

- Application affinity to servers—it may be impossible to remove some application instances from a server due to potential loss of state or the cost of migration to a different server. We cannot remove a server that hosts an application that cannot be replaced.
- Application allocation restrictions—an application may be only runnable on a subset of server machines that match its requirements. When adding a node, we must select one that can run the lowest-performing application. When removing the node, we avoid selecting one that runs the lowest-performing application.
- Machine power efficiency—we prefer to add machines that are more power efficient, where power efficiency is defined as a ratio of machine power usage at maximum CPU utilization to its maximum CPU speed.

Considering that evaluating a subset involves solving the placement problem, which is known to be complex [5], [9], it is reasonable to constrain the search space to subsets whose cardinality differs from the cardinality of the currently running subset by not more than a configured number of machines. Besides reducing the complexity, this conservative approach helps prevent oscillations. The overall complexity of the power management algorithm is therefore equivalent to the complexity of the placement algorithm, and for the algorithm used in this paper [9] it is $\mathcal{O}(NM^2)$.

III. System Evaluation

In this section we evaluate the power management solution proposed in this paper through real system experiments and through simulation.

A. Experimental results

We have integrated our power management technique with IBM WebSphere Extended Deployment [4] management middleware by extending its Application Placement Controller component (APC) with the functionality of the power controller. Besides APC, IBM WebSphere Extended Deployment includes other workload management components: Application Request Flow Manager, Dynamic Workload Manager, and Work Profiler, which are responsible for overload protection, load balancing, and online request profiling, respectively. The accuracy and stability of each of these controllers affects the overall system performance making it difficult to evaluate any single controller in isolation. To reduce the impact of other controllers on the efficacy of power management technique, we isolate them as follows. We profile application requests offline to obtain CPU work factor and client think time for each application and configure the system to use thus computed values instead of online estimates. This change makes our system more responsive to workload intensity changes without the loss of stability than we would normally see in practice. As a result, in a relatively short time interval, we can evaluate the system behavior at a large range of workload intensities while permitting dramatic workload intensity variations. We continue to use the full functionality of the Application Request Flow Manager and Dynamic Workload Manager.

We evaluate the system in a cluster of 13 server machines. One machine serves as a management node where

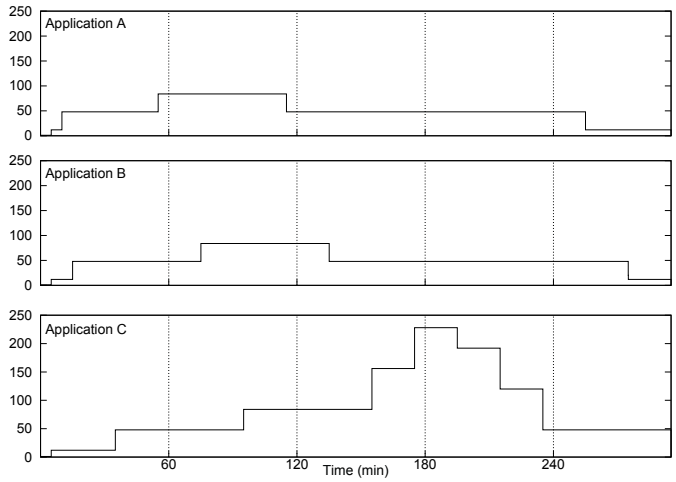

Fig. 2. Number of client sessions for applications A, B, and C as a function of time.

the management components reside. Another machine is used by the L7 Proxy node with overload protection functionality. The remaining 11 machines are used as compute nodes. The compute nodes are single-CPU 3.2GHz Intel Xeon servers with hyperthreading disabled and 4GB of memory.

We load the system using three synthetic applications (A, B, and C) which serve servlet requests that alternate between CPU intensive computation and sleeping. The sleep time emulates application access to a backend database. By controlling the amount of computation and the frequency and duration of sleep times, we can emulate applications with various degrees of computational intensity. The profiled values of the CPU work factor and service times on the server machines are 37.3 mega cycles and 220ms, respectively, for all three applications. We configure response time goals for applications A, B, and C to be 360ms, 600ms, and 840ms, respectively. Throughout all experiments we vary load by changing the number of concurrent client sessions according to pattern illustrated in Figure 2.

We perform four runs in total. The first run (the 'no-power-control' run) has power optimization disabled, and performs only performance management—all nodes are always on, and placement decisions do not take power into account. The other three runs have power optimization enabled, and are run at various settings of performance goal d_0 and rigidity k. We first set $d_0 = 0$ and $k = 300$. With this choice of parameters we expect the system to always satisfy SLA goals but to make no effort to exceed them. In the second run, we use $d_0 = 1$ and $k = 300$, which instructs the system to always maximize performance. Any power savings with this choice of d_0 and k result from consolidating unused cycles. Finally, we set $d_0 = 1$ and $k = 20$, which makes the system attempt to maximize the performance, but under high workload intensity permits it to depart from the best performance by about 10% if this results in turning off a machine.

In Figure 3 we show a time series of the response times for applications A, B and C over time in the four runs with response times averaged over 1 minute intervals. The dotted horizontal lines in Figure 3 show the response time goals for the three applications, as discussed above. We expect that for $d_0 = 0$ and $k = 300$, the average response times should be at, or slightly below, 360ms, 600ms, and 840ms for applications A, B, and C. In the experiment, we observe some departures from this expectation. First, throughout the run, we observe occasional high spikes, which correspond to the increases in workload intensity and last as long as it takes for the system to react to the change (usually about 1-2 minutes). Second, throughout the run, and particularly between 140 and 240 minutes after the beginning of the experiment, the response time for applications is slightly above the configured goal. This happens as a result of a coarse granularity of allocation used by the flow controller, which is unable to effectively utilize CPU capacity that is smaller than what is needed by a full request. In our scenario, an average request requires 37.3 mega cycles and is served by a server within 220 ms. Thus, its CPU demand is about 170MHz, which is roughly 5% of the overall node CPU capacity. Thus, in the worst case, the flow controller may be underutilizing the system by as much as 5%, which results in unnecessary queuing at the L7 proxy and deteriorates the response time. This is clearly a flaw of our management system, which may be addressed in several ways, for example by (1) improving the granularity of management in the flow controller or (2) modifying the placement controller to detect the ratio of CPU capacity that the flow controller is able to utilize and overprovision the system accordingly.

In the second run, for $d_0 = 1$ and $k = 300$, we expect the system to always maximize performance. This means that the response time for all applications should stay at about 220ms (the service time) throughout the entire run. While we observe similar discrepancies as in the first run, overall the response time stays very close to the desired 220ms.

Finally, for $d_0 = 1$ and $k = 20$, we expect the system to achieve a response time of 220ms, while occasionally departing from this level of performance under high workload intensity. The evidence of such departures may be observed in Figure 3 at time intervals 45-55, 160-175, and 220-235 minutes after the beginning of the experiment.

Figure 4 shows a time series of the number of nodes turned on in the three runs in which power management was enabled (in the no-power-control run, all nodes are always on). It may be easily observed that for $d_0 = 0$ and $k = 300$ we are using up to 3 fewer nodes than for $d_0 = 1$ and $k = 300$ and $d_0 = 1$ and $k = 20$. The differences between $d_0 = 1$ and $k = 300$ and $d_0 = 1$ and $k = 20$ show up in points marked by arrows (a), (b), and (c). Arrow (a) shows a case in which with $k = 20$ we are able to serve 48 client sessions for each application using 4 servers, while for $k = 300$, 5 servers must be started. We see similar 1 server differences at points (b) and (c). Overall, as workload increases, higher rigidity makes the system turn on a node sooner and turn it off later than a lower rigidity.

Figure 5 shows application placement on nodes as a function of time observed in the run with $d_0 = 0$ and $k = 300$ (for

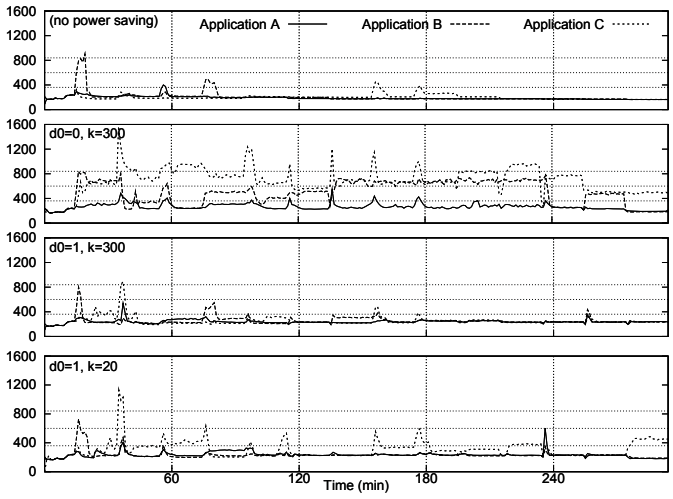

Fig. 3. Observed response times of applications A, B, and C as a function of time for various settings of d_0 and k.

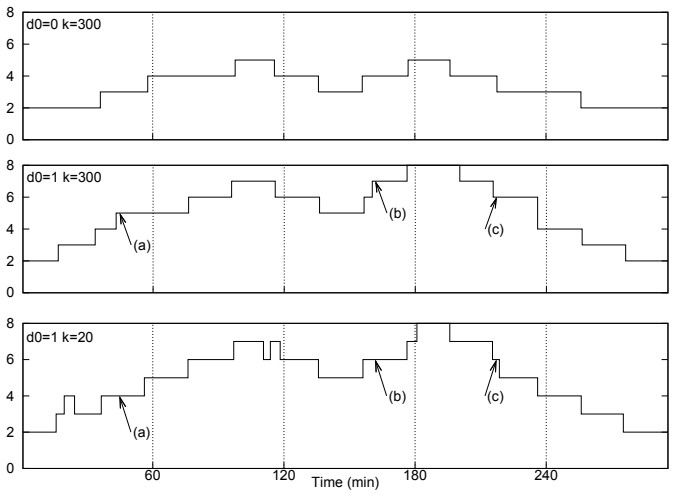

Fig. 4. Number of nodes in use as a function of time for various settings of d_0 and k.

clarity, only the nodes used in this run are shown). The figure permits us to make an interesting observation that our system, in response to a workload change for a particular application frequently responds by increasing the number of instances of other applications. Such a situation occurs, for example, in minute 35, when load for application C is increased and system responds by starting new instances of applications A and B on node 4. This seemingly odd behavior demonstrates a strength of our system, which is its ability to model and calculate how CPU power may be shifted among applications on a server machine. Another interesting scenario occurs in minute 115 where in response to decreased workload of A, we eliminate node 2 from the set of running servers. To accomplish this task, the system moves application C, which is collocated with A on node 2, to node 3.

Figure 6 shows the overall power savings (as a percentage)

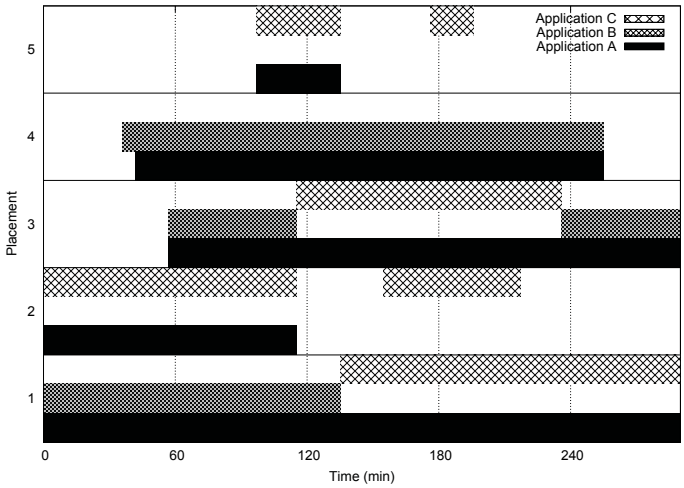

Fig. 5. Application placement as a function of time for $d_0 = 0$ and $k = 300$.

as a function of time in each run of the experiment. The power saving at a given time is defined as difference between the amount of power used at that time in the current run, and that used at that time in the no-power-control run—the higher the number, the more power is being saved. In this experiment, we defined the power consumption of the machines as being 80W when idle, and 100W when at 100% CPU consumption (that is to say, $p_0 = 80$ and $\Pi_{\max} = 100$ for all nodes). Between those two points, the power consumption increases linearly with CPU percentage. These numbers were measured on IBM LS20 blades—whilst this experiment is not conducted on those blades, the power consumption curve used is known to be realistic, as it comes from real machines.

In the no-power-control run, all 11 machines were on throughout, so the minimum possible power consumption (when all nodes are idle) in that run is 880W. In all runs, the maximum possible power consumption (when all nodes are on and running at 100% CPU utilisation) is 1100W. Figure 6 shows that the greatest power saving is obtained when $d_0 = 0$ and $k = 300$—this is expected, as it is in that run that the system is permitted to sacrifice the most performance for power. Figure 3 shows the performance side of this tradeoff, and was described above.

B. Simulations

To further characterize the way in which the tuneable parameters of v affect system behavior, we performed a series of simulations in which we calculated power savings and performance degradation across a wide range of k and d_0.

The simulation ran the same implementation of the placement algorithm that was used in the real experiments in a simulated environment of 20 nodes and 20 applications, each with its own time-varying workload, with a simulated flow controller and APC that carried out the placement algorithm's decisions exactly. We searched the space of k and d_0 by choosing $d_0 \in \{0, 0.1, 0.2, \ldots, 1\}$ and $log_2(k) \in \{0, 0.5, 1, \ldots, 9\}$. For each pair (d_0, k), two runs were made: one in which

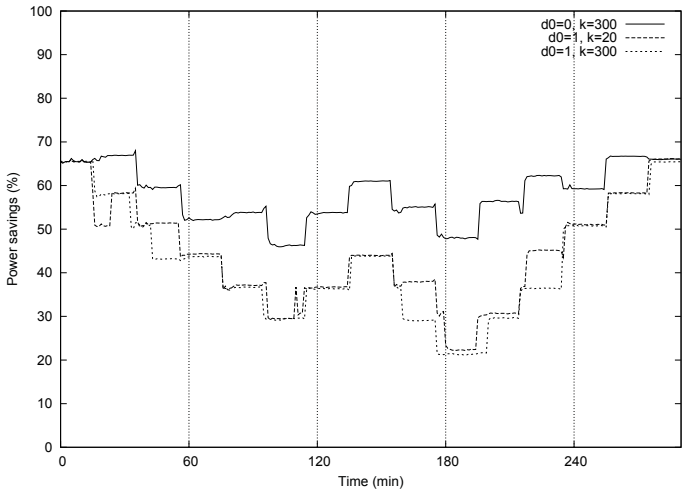

Fig. 6. Power savings, over the no-power-control run, as a function of time for various settings of d_0 and k.

power-savings was taken into account, and one in which it was not. Each run consisted of a sequence of placements calculated for a sequence of 32 different workloads that varied in a statistically reasonable, but reproducible, way. For each such placement, simulated power usage and application performance levels were measured. The power usage was averaged over the sequence of placements, and the utility values were averaged over the placements and the applications to get two aggregate quantities per run: the average power usage level, and the average application performance level. By taking the difference in these aggregates for pairs of runs in which one run used the power-aware placement and the other did not, we were able to measure the average power savings and performance degradation for the given k and d_0.

Figure 7 shows the simulation results. The top part shows the power savings for different values of d_0 as a function of $log_2(k)$. The bottom part shows the corresponding change in performance. As the curves show, for any selected value of d_0, the change in overall behavior as a function of k is smoothly varying and monotonic; similarly, adjustments to d_0 cause a smooth, monotonic change in behavior. Thus k and d_0 are useful control parameters: administrators can adjust them to find desired tradeoff points, and the system's response to changes is both smooth and predictable.

IV. Related Work

The three principal approaches to power management are dynamic voltage Scaling (DVS), dynamic frequency scaling (DFS), and server consolidation.

DVS permits a quadratic reduction of power usage by lowering processor voltage at times of low CPU utilization. Bohrer et al. [10] have shown that this technique can reduce the CPU energy consumption by as much as 30%. Elnozahy et al. [11] have proposed a scheme that allows Web requests to be held in a queue for a short period of time at times of low workload intensity, and dispatched from the queue as a batch.

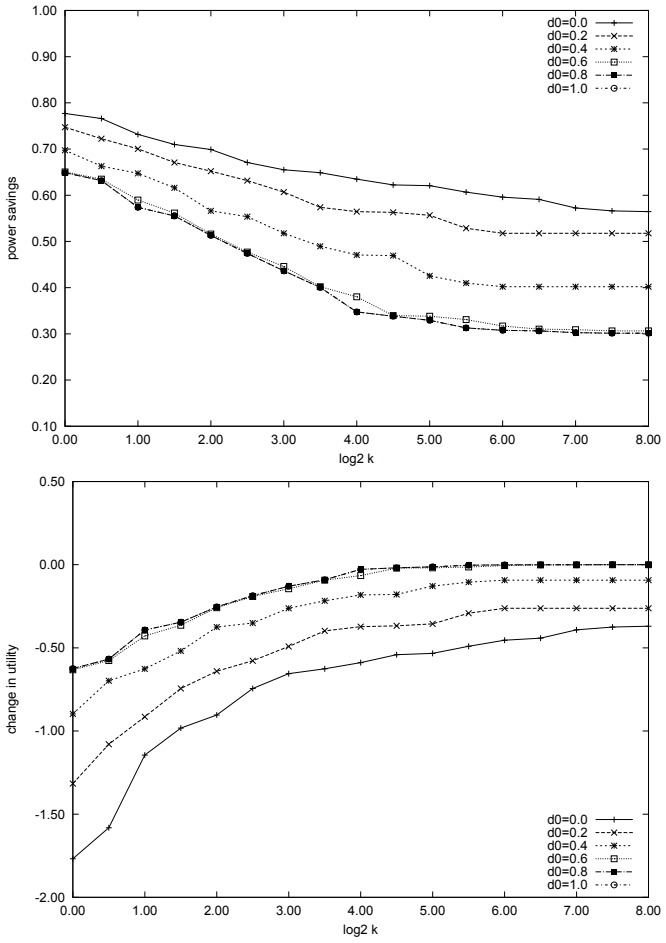

Fig. 7. Power savings and performance degradation as functions of $log_2(k)$, for selected d_0.

This permits a CPU to remain at low power usage for longer periods of time and thus offer further energy savings. Pillai et al. [12] studied DVS algorithms that offer real-time deadline guarantees. These approaches focus on a single server system. Wang et al. [13] propose a control-theoretic technique to adjust server voltage rates in a cluster of server machines, while permitting request buffering at the entrance to the system.

DFS reduces clock frequency, permitting the CPU to consume less power. Kephart et al. [1] use it in a scheme that trades off Web application performance and power usage based on a prescribed utility function that expresses monetary value of achieving a certain quality of service and the cost of power needed to provide CPU capacity necessary to achieve this level of QoS. The proposed feed back controller modifies CPU frequency setting and achieves 10% reduction of power consumption for a small degradation of application performance.

Server consolidation has become a particularly atractive option with the advent of virtualization technologies that permit live migration of arbitrary workloads. Chase et al. [2] propose an economic model that finds an optimal allocation of servers to application clusters. The model associates the cost of power usage with each server thus representing the

tradeoff between power and performance. Our approach differs from theirs in several ways. First, we allow finer-grained resource allocation to application clusters as, in our technique, application clusters overlap on a set of physical machines. This makes it necessary to not only decide the number of servers that are powered on, but also the placement on applications on servers that are on. Second, while they consider only CPU resources, our problem is also constrained by memory capacity and a variety of operational constraints. Finally, they assume the cluster of servers to be homogeneous, while our approach permits heterogeneous server clusters.

Elnozahy et al. [14] study several policies of energy conservation including server consolidation and a hybrid approach involving server consolidation and DVS. The decision whether to power a server on or off is made based on the operating frequency of currently running servers. The approach evaluates an impact of these decisions on application performance. As in [2] a homogeneous server cluster is assumed.

Chen et al. [15] discuss a cluster of Web application servers in which power usage is minimized by both DVS and consolidation, subject to the constraint that application response times meet SLA goals. This approach can ensure SLA satisfaction, but does not permit a dynamical tradoff between power savings and performance level.

Bobroff et al. [16] introduce an algorithm to consolidate workloads running inside virtual machines. They minimize the set of running machines in a heterogeneous server cluster. An application is equated with a virtual machine, whereas we deal with applications that may, and usually need, to be spread across multiple physical or virtual machines. Application demands are given in the form of direct resource requirements and no explicit model of application performance is present. Likewise, there is no explicit notion of power cost that would help differentiate among machines with different energy consumption curves.

Tsai et al. [17] propose a capacity planning technique that estimates the CPU requirement for a Web application based on network and OS measurements. This technique is applied to change the number of running servers based on workload intensity in a cluster of homogeneous machines. The approach focuses on a single application and thus does not have to address the problem of packing applications on running servers while maintaining their various operational constraints.

V. Conclusions

In this paper we have shown how power-savings may be incorporated into advanced, SLA-based performance-management systems by means of an approximate optimization over a synthetic utility function. The combined power- and performance-management algorithm preserves fairness of allocation while simultaneously reducing power usage in a safe and flexible way. It also presents administrators with a small number of control parameters that can be used to achieve a "correct" tradeoff between maintaining performance levels and saving power. We have shown by experiment and simulation that our approach is likely to work in real-world deployments and that the control parameters are an effective way of shaping system behavior.

Future work includes further testing of the system, especially in complex situations involving policy-based constraints and memory allocation problems; integration with external components to handle the actual turning on and off of computers; consideration of policies for machine lifecycle issues; improving the scalability of the algorithms; and combining this work with support for other workload types.

Acknowledgments

The authors would like to thank David Carrera, for creating the simulator upon which that used for the simulations in Section III-B is based.

References

[1] J. O. Kephart, H. Chan, R. Das, D. W. Levine, G. Tesauro, and F. R. an C. Lefurgy, "Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs," in *IEEE Intl. Conf. on Autonomic Computing*, Jun. 2006, pp. 145–154.

[2] J. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *ACM Symposium on Operating Systems Principles*, 2001, pp. 103–116.

[3] X. Fan, W. Weber, and L. Barroso, "Power provisioning for a warehouse-sized computer," in *Proc. 34th Intl. Symposium on Computer Architecture (ISCA '07)*, 2007, pp. 13–23.

[4] "WebSphere Extended Deployment." [Online]. Available: http://www-306.ibm.com/software/webservers/appserv/extend/

[5] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic placement for clustered web applications," in *World Wide Web Conference*, Edinburgh, Scotland, May 2006.

[6] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi, "Dynamic estimation of cpu demand of web traffic," in *VALUETOOLS*, Pisa, Italy, Oct. 2006.

[7] G. Pacifici, M. Spreitzer, A. Tantawi, , and A. Youssef, "Performance management for cluster based web services," *Journal of Network and Systems Management*, vol. 23, no. 12, 2005.

[8] G. Pacifici, W. Segmuller, M. Spreitzer, M. Steinder, A. Tantawi, and A. Youssef, "Managing the response time for multi-tiered web applications," IBM, Tech. Rep. Tech. Rep. RC 23651, 2005.

[9] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *World Wide Web Conference*, Banff, Alberta, Canada, May 2007.

[10] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The case for power management in web servers," in *Power Aware Computing*, Graybill and Melhem, Eds. Kluwer Academic Publications, 2002.

[11] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy conservation policies for server clusters," in *4*th *USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, Mar. 2003.

[12] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *ACM Symposium on Operating systems Principles*, 2001, pp. 89–102.

[13] M. Wang, N. Kandasamy, A. Guea, and M. Kam, "Adaptive performance control of computing systems via distributed cooperative control: Application to power management in computing clusters," in *IEEE Intl. Conf. on Autonomic Computing*, Jun. 2006, pp. 165–174.

[14] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in 2^{nd} *Workshop on Power Aware Computing Systems (in conjunction with HPCA)*, Feb. 2002.

[15] Y. Chen, A. Das, and W. Qin, "Managing server energy and opeational costs in hosting centers," in *Proc. Intl. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS'05)*, 2007, pp. 303–314.

[16] N. Bobroff, A. Kochut, and K. Beatty, "Dynamic placement of virtual machines for managing SLA violations," in *Integrated Network Management*, Munich, Germany, May 2007, pp. 119 – 128.

[17] C.-H. Tsai, K. G. Shin, J. Reumann, and S. Singhal, "Online web cluster capacity estimation and its application to energy conservation," *IEEE Transactional on Parallel and distributed Systems*, vol. 18, no. 7, pp. 932–945, 2007.