Workload-based power management for parallel computer systems

D. J. Bradley
R. E. Harper
S. W. Hunter

This paper describes and evaluates predictive power management algorithms that we have developed to minimize energy consumption and unmet demand in parallel computer systems. The algorithms are evaluated using workload data obtained from production servers from several applications, showing that energy savings of 20% or more can readily be achieved, with a small degree of unmet demand and acceptable reliability, availability, and serviceability (RAS) impact. The implementation of these algorithms in IBM system management software and the possibilities for future work are discussed.

1. Introduction

Concern over the power consumption of computer systems is no longer limited to long-lived embedded applications such as spacecraft and aerospace systems, but is spreading to the general-purpose computer market. Power consumption and dissipation constraints are jeopardizing the ability of the IT industry to support the business demands of present-day workloads, especially in the areas of electronic commerce and Web hosting. Given existing physical infrastructure, it can be difficult to get power into and heat out of such large systems, while power shortages such as those recently experienced in California impose further, unpredictable constraints.

This is in part because server complex size is growing dramatically in response to skyrocketing electronic commerce and Web hosting computational needs, and in part because processor power consumption is also growing, to well over 100 watts per processor for some present-day processors. Modern-day server complexes for electronic commerce and Web hosting constitute literally thousands of servers operated in parallel, consuming thousands of square feet of computer room space and many kilowatts of power. In some cases there is simply no additional power and cooling capacity in a given physical environment to support increases in capacity.

Lower-power processors are now becoming available from the industry, but it is safe to say that a new market-acceptable price–power–performance equilibrium has yet to be demonstrated in the server space, and in fact the performance characteristics of lower-power processors may limit their ultimate penetration into this space. Moreover, processor power consumption, while significant, does not account for all the power consumed by a modern server. Memory controllers, I/O adapters, disk drives, memory, and other devices consume a non-negligible fraction of server power cannot be disregarded. Finally, incorporation of advanced power management functionality into server hardware architectures, while proceeding because of market demand, must always be balanced against the reality of minimizing base server cost in an extremely price-competitive market.

Fortunately, electronic commerce and Web-serving workloads have certain characteristics that make them amenable to predictive system management techniques that can effectively reduce power consumption for a broad class of server architectures.[1] They usually support periodic or otherwise variable workloads, with the peak workload being substantially higher than the minimum or

[1] Other appropriately implemented and managed workloads may also have many of these characteristics.

©**Copyright** 2003 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

0018-8646/03/$5.00 © 2003 IBM

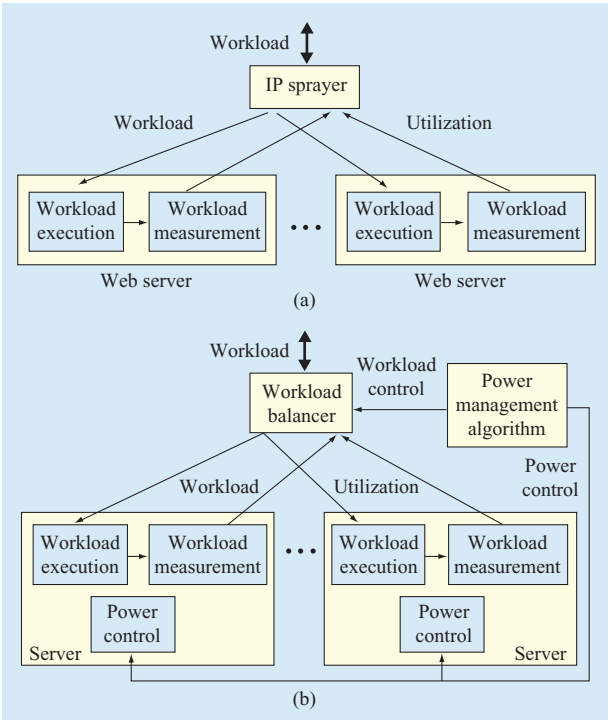

Figure 1

IP address sprayer: (a) Schema; (b) power management.

average workload. The dynamic range of the workload is often a factor of 10; that is, the peak workload can be ten times the minimum workload. Furthermore, such stampedes can cause the transition from low to high workload (and vice versa) to be abrupt. These workload attributes imply that keeping all of the servers powered on all of the time might be a waste of power, yet the possibility of failing to meet an unanticipated surge of demand must be minimized.

These important workloads have other promising characteristics from a power management perspective. They are highly parallel and relatively easy to balance. A typical Web-serving system has a large number of Web servers fronted by a load-balancing "IP sprayer," which provides a single Internet protocol (IP) address to the outside world and dispatches requests from the outside world to the many Web servers in the complex to balance the load among them [**Figure 1(a)**].

Simplistically, the IP (address) sprayer sends a given request to the server having the lowest utilization, and, in turn, the servers keep the IP sprayer updated with their utilization, response time, or other indication. Workload is merely routed around failed servers, and the users having transactions or sessions on a failed server can click again to have their request go to another server. This works quite well because a given request is locale-transparent: Assuming that all servers have access to the same back-end source of Web pages or database, as is common in practice, that request can be dispatched to any Web server in the complex. Finally, the requests are short-lived enough that if a given server is "condemned" and new workload is withheld from it, its utilization quickly falls, whereas if a new server is brought on-line, new workload can readily be dispatched to it and its utilization quickly rises. These workload attributes imply that it should be relatively straightforward to power individual systems off and on with minimal disruption to the overall application.

On the basis of these needs and observations, we have developed and evaluated an algorithm (called CoolRunnings) that exploits this environment to manage power on the basis of measured and predicted workload, such that both unmet demand and power consumption are minimized. It does this by

1. Measuring and characterizing the workload on all of the servers in a defined group.
2. Determining whether any servers need to be powered on or off in the near future by assessing the current capacity relative to the predicted capacity needs.
3. Manipulating the existing system and workload management functions to remove load from servers to be turned off.
4. Physically turning on or off (or switching into and out of standby or hibernation) designated servers using existing system management interfaces.

Thus, there is a measurement component, an algorithmic component, a workload management component, and a physical server power management component, as shown in **Figure 1(b)**.

The CoolRunnings algorithm is not intended to replace emerging power management techniques internal to the architecture of a server, such as processor clock and voltage modulation, nor workload-balancing techniques such as those provided by existing workload and system management infrastructures. It is intended to supplement them with computing facility-scale power management techniques that are capable of supporting a wide range of systems. CoolRunnings also differs from recently deployed IBM functions and products, notably Capacity Manager [1] and Software Rejuvenation [1]. Capacity Manager is designed to measure long-term (e.g., weeks and months) trends in resource utilization, and to instruct the customer to procure additional appropriate resources before system performance suffers. Software Rejuvenation is designed to detect long-term (weeks) resource exhaustion due to bugs such as memory leaks. All or part of the system is restarted at a convenient time (automatically or by the customer) before exhaustion of the resources causes an

unplanned outage. CoolRunnings shares some technologies from these products, notably the workload characterization and saturation definitions of Capacity Management, and some prediction concepts from Software Rejuvenation, but it is primarily intended to provide short-term (e.g., minutes to hours in advance) projections of workload, and power systems off or on appropriately to meet the objective function. Thus, CoolRunnings is interested only in projecting workload far enough forward in time to allow adequate resources to be powered on and readied for work when the workload requires them.

This paper describes some predictive power management algorithms we have developed that attempt to minimize energy and unmet demand. These algorithms are evaluated using historical workload data obtained from production servers from several different application domains, and we show that energy savings of 20% or more can be readily achieved with a very small amount of unmet demand. We then describe how this technology can be implemented in IBM system management software, and point to interesting possibilities for future work.

2. Related work

There has been much activity in the study of power conservation in computing devices. This can be accomplished at the single system image level [2–4], as well as at the multisystem level [2, 5–8]. The application of power management to Web applications [9] is especially interesting because there is often a widely varying workload [10], such that dynamic decisions in support of power management policies can be made through the use of feedback mechanisms [11]. This paper focuses on the algorithms for making those decisions on the basis of current workload conditions in a multiserver infrastructure. This technique could be especially useful when applied to servers with a blade form factor [12] in a common chassis.

3. Power management algorithms

Objective

The objective of our power management algorithm is to meet the offered workload at all times, subject to minimizing power consumption and not unduly increasing server failure rate or client disconnection rate due to excessive power cycling.

Specifically, the figures of merit that the algorithm attempts to minimize are the following:

1. The energy consumption of the server aggregate, normalized to the energy consumption when all servers are powered on. This is computed as the energy consumption per server (including all of its ancillary components such as disks, fans, and additional logic components times the number of servers that are powered on. We do not differentiate between the energy consumption of a completely utilized powered-on server and an unutilized powered-on server.
2. The unmet demand relative to integrated demand over a history window, suitably normalized to obtain a number between 0 and 1. As mentioned below, this can be extended to incorporate response time characteristics of met demand.
3. The number of power-on cycles incurred by each system, normalized to power-on/off cycles per day. This must be minimized in order to reduce reliability degradation due to power cycles.

Workload characterization

A server has several resources whose consumption renders it unable to accommodate additional load. On the basis of experience with server workloads, we use the following metrics for utilization:

- CPU utilization.
- Physical memory utilization.
- Local area network adapter bandwidth utilization.
- Disk bandwidth utilization.

These parameters are readily measured[2] and are the same as those used to characterize server workload in the IBM Capacity Management product.

The system workload traces that we had available for this study always contained CPU utilization, and sometimes contained proxy information for these other parameters. We chose to use CPU utilization alone to test our algorithm, recognizing that other resources may ultimately constrain performance. Incorporating additional performance parameters into our algorithm is straightforward. If the utilization of any resource is greater than a threshold, additional capacity should be added; in our case, if any servers are powered off, enough should be powered on to reduce all utilizations on all servers to below that threshold. If all utilizations are lower than this threshold (correcting for hysteresis), and assuming that there is adequate capacity in the resulting server group to absorb the load of at least one server without any resource on any server being over-utilized, at least one server can be powered off.

For our work, we defined the saturation threshold for any resource to be 70%, as in the IBM Capacity Management product. This figure reflects the realities of the performance characteristics of industry-standard servers, which suffer rapidly degrading performance as any resource utilization approaches this value. Also, below this

[2] In the Microsoft** Windows** operating system, the parameters can be derived from the performance counters. In Linux**, they can be derived from data residing in the /proc directory structure.

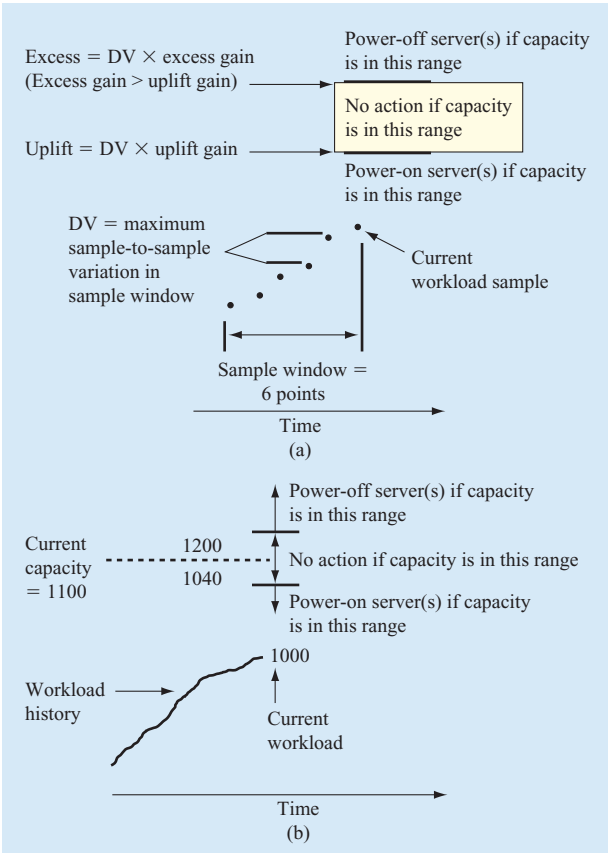

Figure 2

(a) Short-term gain algorithm; (b) algorithm demonstration.

utilization, nonlinearities due to queuing are not so large that approximating response time from utilization is too misleading. Thresholds can be increased or decreased on the basis of the load curve of a given system and application or the degree of conservatism of a system administrator, and different resources can be given different thresholds.

If enough is known about the system, the measured performance parameters can be combined into a fairly simple closed-form queuing model, and the projected response time can be incorporated into the objective function. In this case, the algorithm would attempt to minimize the unmet demand, plus the response time associated with the met demand. In some architectures, the response times can be measured directly, as in [5].

Gain-based algorithm

The gain-based version of the CoolRunnings algorithm attempts to estimate a capacity envelope at a time in the near future, and power servers on or off to maintain that capacity envelope at a desired amount (but not too far) above the projected workload. The entire capacity envelope must always be above the current utilization; otherwise, unmet demand will result. The projection time is equal to the time required to power up a server and ready it for work.

The lower limit of the capacity envelope, that is, the minimum amount of capacity deemed necessary given the current workload, is projected by taking the current workload and adding to it an "uplift" that is based on the maximum sample-to-sample deviation that was observed over a given workload history. The upper limit of the capacity envelope, that is, the maximum amount of capacity deemed necessary given the current workload, is projected by taking the current workload and adding to it an "excess" that is similarly based on the maximum sample-to-sample deviation that was observed over a given workload history.

The uplift is equal to the "uplift gain" times this maximum deviation value, and the excess is equal to the "excess gain" times this maximum deviation value. If the current capacity is between the uplift and the excess, no action is taken. If the current capacity is less than the uplift, one or more servers are scheduled to be powered on; if the current capacity is greater than the excess, one or more servers are scheduled to be powered off. Note that if the workload is roughly constant and the uplift is equal to the excess, the algorithm will alternately power servers on and off at each sample point. The basics of the method are shown in **Figure 2(a)** for a sample window size of six points.

For example [**Figure 2(b)**], if the current capacity is 1100, the current workload is 1000, the history window is 20, the uplift gain is 20%, and the excess gain is 100%, the algorithm runs as follows:

1. Scan the last 20 workload samples and calculate the maximum point-to-point deviation. Suppose that this value is 200.
2. Calculate the projected capacity envelope. The uplift is $1000 + 20\% \times 200 = 1040$, and the excess is $1000 + 100\% \times 200 = 1200$.
3. The capacity is 1100, which is greater than the uplift yet less than the excess. Thus, no action has to be taken, as illustrated in Figure 2(b). If the capacity were less than 1040, one or more servers would have to be powered on to maintain it in the desired capacity range, and if the capacity were greater than 1200, one or more servers would have to be powered off.

The history window size, uplift gain, and excess gain are fundamental to the performance of the algorithm, and must be judiciously chosen. We show the effects of these parameters on a variety of workloads below.

Algorithm based on seasonal characterization

The short-term algorithm does not account for sudden spikes in workload, because these are not presaged by variations in the recent sample window. **Figure 3(a)** illustrates the unfortunate situation in which a short-term algorithm, applied to the numerical example above, fails to predict a sudden spike in workload, thus resulting in unmet demand.

Fortunately, many workload spikes are repetitious [**Figure 3(b)**], based on weekly or daily activity such as nightly backups, weekly logons on Monday, market openings and closings, etc. For epochs that are not daily or weekly, it is a straightforward matter to perform a calculation such as an autocorrelation to determine the workload periodicity, and define the epochs appropriately. For our initial exploratory purposes, we assume (and our data reinforces) that weekly and daily periods predominate.

To accommodate seasonal phenomena, we have developed another algorithm that is based on collecting workload data over a prior epoch in time, characterizing the workload of future epochs based on prior epochs, and setting up a short-term power-on/off schedule based on that characterization. This approach has the benefit that it can speculatively power-on systems before periodic surges in workload. Such a quasi-static epochal capacity schedule can be overridden by exigencies of the moment, by augmenting it with a gain-based policy. For example, if in the next time increment the schedule states that certain capacity is needed, but a gain-based policy as described above indicates that more capacity than that is needed, the capacity indicated by the gain-based policy can be powered on.

Details of hybrid gain-based/seasonal algorithm

The details of an implementation of this algorithm are described below. As before, the algorithm consists of a component to monitor and characterize workload and a component to adjust capacity.

The utilization-monitoring component measures the difference in utilization from one point to the next, with the intent of detecting and recording for future reference a workload spike that may not have been accommodated by the short-term algorithm. It does this by detecting whether the difference in utilization is greater than a pre-programmed amount, and setting flags accordingly for future reference.

For example, if the most recent sample is greater than the previous sample by a given amount (called the *threshold up* parameter), the utilization-monitoring component can set a flag for that particular point in time, indicating that in one epoch *minus one sample interval*, the additional capacity should be added. The amount of capacity scheduled to be added in one epoch minus one sample interval depends on the difference in the most recent and the next most recent samples. If the most recent sample is less than the previous sample by a given amount (called the *threshold down* parameter), the utilization-monitoring component can set a flag for that particular time, indicating that in one epoch minus one sample interval from the current time, capacity should be removed. The utilization-monitoring component performs this characterization for every single sample and stores the results for future reference.

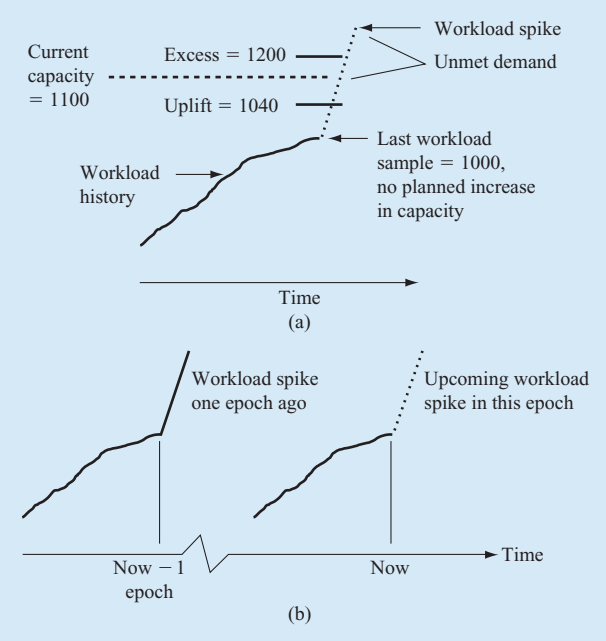

Figure 3

Workload variations: (a) Unpredicted workload spike; (b) epochal spike prediction.

The capacity-adjustment component adjusts capacity for the next sample point on the basis of utilization from prior epochs. At each sample point, it examines the flags for the time point that is one epoch in the past. If the flags indicate that capacity must be added or removed, the capacity-adjustment component does so (**Figure 4**).

A workload usually exhibits multiple concurrent epochs of differing periodicities. For example, a workload may exhibit a daily, weekly, and monthly repetitiveness that can be detected and exploited. Thus, the capacity adjustment component must look one day, one week, and possibly one month into the past to make the capacity-adjustment decision.

Because of sample time quantization, the monitoring system may estimate the occurrence of a spike inaccurately. Thus, when calculating the flags for a given point in time, it is useful for the algorithm to examine not

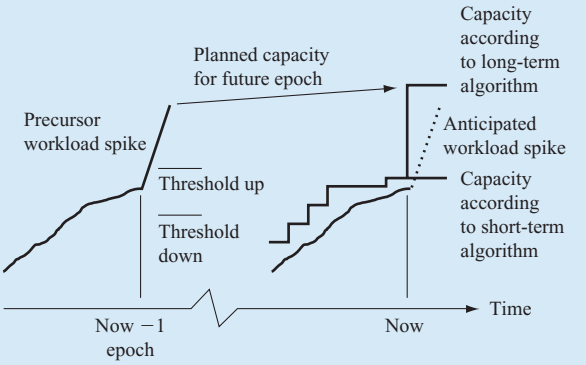

Figure 4

Predicted epochal spike.

only the sample immediately following the point one epoch back in time, but several samples after that point (*lookahead horizon*).

The resulting hybrid algorithm has the capability of reacting to random workloads using the gain-based algorithm, and the capability of accommodating periodic workloads. The algorithm uses the gain-based algorithm as described above to decide whether recent variations in utilization justify adding or removing servers, and uses the epochal algorithm to make the determination on the basis of long-term (i.e., one or more epochs) variations in utilization.

Algorithm pseudocode

The following pseudocode outlines the algorithm:

```
Workload characterization step:
Measure Current Utilization
If current utilization − last utilization
      > Threshold Up
  Record Adjustment Up = current
      utilization − last utilization for
      later use
If last utilization − current utilization
      > Threshold Down
  Record Adjustment Down = current
      utilization − last utilization for
      later use

Capacity adjustment step:
Measure Current Utilization
Short-Term Phase
  Over last N points, calculate DV =
       max(abs(point-to-point variation))
  Calculate Uplift = DV * Uplift Gain
  Calculate Excess = DV * Excess Gain
Long-Term Phase
  For each epoch
    Retrieve Adjustment Up for sample one
         epoch back in time, minus one
         sample interval
    Retrieve Adjustment Down for sample
         one epoch back in time, minus one
         sample interval
  If any epoch has Adjustment Up,
        Adjustment = max(Adjustment Up
        over all epochs)
  Else Adjustment = max(Adjustment Down
        over all epochs)

Compute thresholds step:
  If Adjustment >= 0 (i.e., Adjustment
            Up)
    minCapacity = CurrentUtilization +
         max(Adjustment, Uplift)
    maxCapacity = Current Utilization +
         max(Adjustment, Excess)
  Else (i.e., Adjustment Down)
    minCapacity = Current Utilization +
         max(0, Uplift + Adjustment)
    maxCapacity = Current Utilization +
         max(0, Excess + Adjustment)

Power on/off processors step:
  Determine Capacity = Total Available
         processing power * Desired
         Utilization
  If Capacity < minCapacity
    Power On processors until Capacity >
         minCapacity
  Else if Capacity > maxCapacity
    Power Off processors until Capacity <
         maxCapacity, while ensuring that
         Capacity > minCapacity
         (i.e., after powering off
         processors, Capacity may be <
         maxCapacity, but will definitely
         be > minCapacity)
```

Self-tuning implementation

The algorithm has the following tunable parameters:

- Uplift gain.
- Excess gain.
- Desired utilization.
- Window size.
- Threshold up.
- Threshold down.
- Lookahead horizon (not evaluated in this work).

Uplift gain, excess gain, threshold up, threshold down, desired utilization, and window size comprise a

multidimensional search space which contains an optimum figure of merit that is dependent on the workload characteristics as well as the relative weighting of energy consumption and unmet demand. In general, finding the optimum values of these figures of merit within this search space is tedious and *ad hoc* at best, and certainly not practical or optimal for all workloads and system administration policies encountered in the field. Therefore, we developed a self-tuning gain-based algorithm that calculates energy consumption and unmet demand on the basis of a workload sample for a large set of values of the input parameters of the algorithm. Then, the algorithm searches through this set of input values to find the settings that optimize the figures of merit for the given workload. Any search algorithm can be used; typically, because the state space is small, an exhaustive enumeration could even be used. The self-tuning approach has the advantage that it can dynamically adapt not only to any workload that is encountered in the field, but to changes that occur to the workload over time on any given system.

4. Empirical workload measurements and algorithm performance

Simulation methodology

To evaluate candidate power management algorithms under controlled experimental conditions, we constructed a workload generator and algorithm simulator that can generate a variety of synthetic workloads, ranging from random to periodic, with varying amounts of noise. Various figures of merit such as energy consumption and unmet demand can be calculated and displayed. The algorithm simulator can also read files containing empirical workload trajectories, as discussed below. This simulator has been valuable in rapidly working out algorithm details under closely and easily controlled laboratory conditions.

The simulator makes certain assumptions about the dynamics of the workload. The algorithm works by taking a sample of utilization, looking at recent and epochal workload history, and computing whether one or more servers should be powered on or off. For energy calculation purposes, the simulator assumes that when we command a server to be powered on, it consumes power immediately but delivers no capacity until the next sample. This has the effect of overestimating the unmet demand, since it is likely that demand will be served in a shorter time than the sampling interval. Upon shutdown, the simulator assumes that once a server has been commanded to shut down, it requires an entire sample interval to shut down, during which it consumes power but provides no capacity. This simplification probably overemphasizes the energy consumption, since it is likely that a server can actually shut down in less time than one sample interval.

Startup and shutdown durations are also a function of workload inertia, which influences the time required for the workload from a given server to decay and for full load to be applied to a newly powered-on server. Even if a system can be put into and taken out of standby or hibernation in a few seconds (for servers, this remains to be seen), if the workload takes minutes to decay or ramp up, the effective startup or shutdown time will be constrained by the time constant of this workload. Furthermore, we do not discriminate between the power consumption of a fully utilized and an unutilized server, and the power consumption of the system is calculated as the power consumption of a single whole server times the number of powered-on servers.

For convenience, we set the power-on/off latency to be equal to the sample period of the raw data, which is 15 minutes for the IBM Domino* server [13], two minutes for the Divahouse server,[3] and five minutes for the IBM server containing data from the 2000 Olympic Games held in Sidney, Australia[4] (these workloads are described below). We have not explored the effect of the frequency of power-on/off opportunities on algorithm performance, except to note that as the algorithm tries to minimize power-on/off cycles, it will not be able to take advantage of frequent opportunities to cycle power, so we would expect the effect of power-on/off latency to be weak.

Further assumptions are made about the persistence of unmet demand. We assume that unmet demand is lost forever, whereas in actual usage it is possible that unmet demand could actually be deferred until the requisite capacity is brought on line. We do not model such demand deferral, which once again probably causes us to overstate unmet demand.

To develop the results presented in this paper, the simulator was set up to automatically evaluate the algorithm using a large combination of input parameters, and compute the energy as a function of unmet demand for all of them. In all subsequent evaluations, no self-tuning of parameters was performed in order to allow us to readily observe the effects of varying each parameter.

Lotus Domino database server workload

We obtained workload data as a function of time for a number of Lotus Domino servers in use at IBM facilities in Poughkeepsie, New York. These servers support business applications of the IBM Corporation such as databases and email. Although the workload is not necessarily fungible, in other respects as an aggregate it seems to

[3] Divahouse is the name of the University of North Carolina (UNC) server.
[4] The site is no longer available.

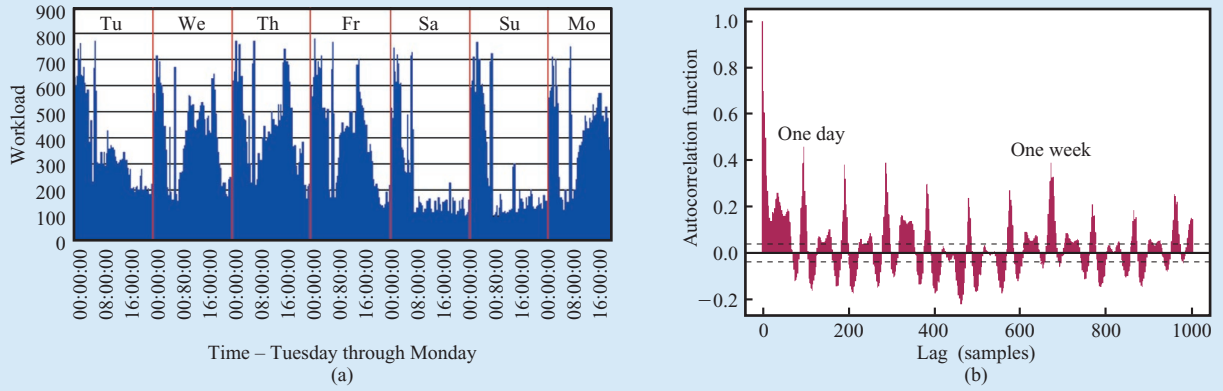

Figure 5

Lotus Domino server workloads: (a) CPU utilization for ten processors; (b) autocorrelation of workload.

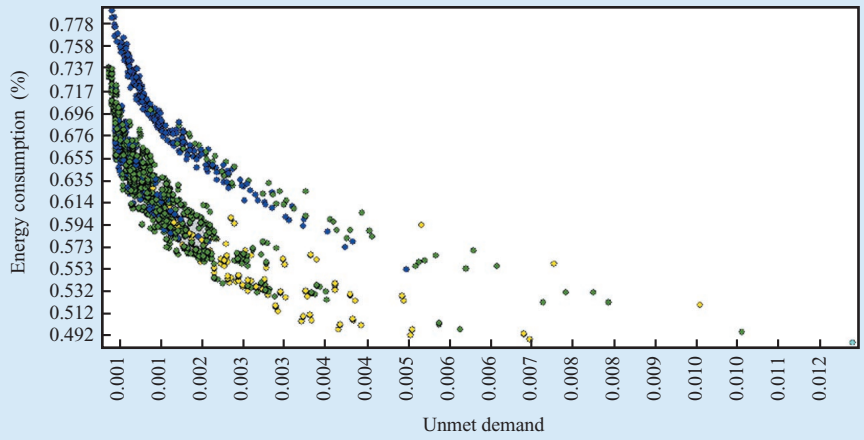

Figure 6

Energy consumption vs. unmet demand for Domino workload.

exhibit realistic workload behavior such as periodicity, high dynamic range, and sudden workload transients.

A typical segment of this data is shown in **Figure 5(a)**. The data workload shows the total CPU utilization of the ten servers, sampled every 15 minutes.[5] The time span is midnight Tuesday to midnight the subsequent Monday during January 2000. The periodic nature, high dynamic range, and sharp transient behavior of the workload are apparent, as is ample opportunity for power management.

The autocorrelation for this workload is shown in **Figure 5(b)**. The "lag" is expressed in samples, so one day corresponds to 95 samples and one week corresponds to 665 samples. A strong daily autocorrelation is easily visible, as is a slightly elevated autocorrelation at one week.

Figure 6 shows the unmet demand versus energy tradeoff for this workload for a large set of input parameters. For Figure 6 [as well as Figures 7, 9(b), and 11(a), shown later], the ordinate shows energy consumption relative to the energy consumption if all servers were powered on all the time (with higher energy consumption at the top of the axis), and the abscissa shows unmet demand relative to the total demand applied to the system over the run duration (with higher unmet demand at the right of the axis). Each circle on the chart shows the energy and unmet demand for a given setting of

[5] The maximum CPU utilization for the ten servers is 1000 "workload units."

the six parameters. The energy savings range from 26% to 52%, while the unmet demand ranges from 0.1% to 1.2%.

The color of the circle indicates the number of power-on cycles. A blue circle indicates that the number of power-on cycles for the entire system for the entire evaluation interval (eight weeks) was less than one per processor per day; a green circle indicates one to two power-on cycles per processor per day; yellow indicates two to three power-on cycles, and red indicates more than three power-on cycles. For reference, a scenario in which each server is power-cycled once per day is not considered to unduly reduce server hardware reliability.

The twin-hyperbola curve has a "wishbone" shape, with energy vs. unmet demand exhibiting a hyperbolic tradeoff along two distinct arcs. To understand which parameters cause the figure of merit to traverse a particular arc of the curve, which ones cause it to cross from one arc to the other, and, ultimately, what constitute relatively good settings, a number of vector field plots were produced that show the effect of changing a single parameter at a time by the same increment.

The first plot [**Figure 7(a)**] shows that the effect of increasing uplift gain is to reduce unmet demand and increase energy within a hyperbolic arc, but without crossing to the other arc. An increase in uplift gain causes the algorithm to be more likely to turn on processors on the basis of a given degree of variation in the short-term window. Thus, energy consumption is higher, but the likelihood of unmet demand is lower.

The next chart [**Figure 7(b)**] shows the effect of increasing excess gain. As excess gain increases, the system is less likely to power-off processors on the basis of a short-term projected capacity surfeit, so once again energy consumption increases while the likelihood of unmet demand decreases, and a change in excess gain does not cause the point to change arcs.

The next chart [**Figure 7(c)**] shows the effect of threshold up on energy consumption and unmet demand. Threshold up determines how sensitive the algorithm is to sudden increases in demand that occurred one epoch ago (for this workload, either one day or one week ago). If a workload increment in the previous epoch is greater than the threshold up, the algorithm will immediately schedule a commensurate increase in capacity at the current time. Thus, a low value of threshold up causes the algorithm to closely track the epochal variation, and a high value of threshold up causes the algorithm to ignore any epochal variation.

This explains the characteristic "wishbone" shape of the energy–unmet demand curve. The left arc of the curve corresponds to low values of threshold up: Because the algorithm is sensitive to the existing epochal variations in the workload, and proactively schedules capacity increases based on those variations, the unmet demand is lower

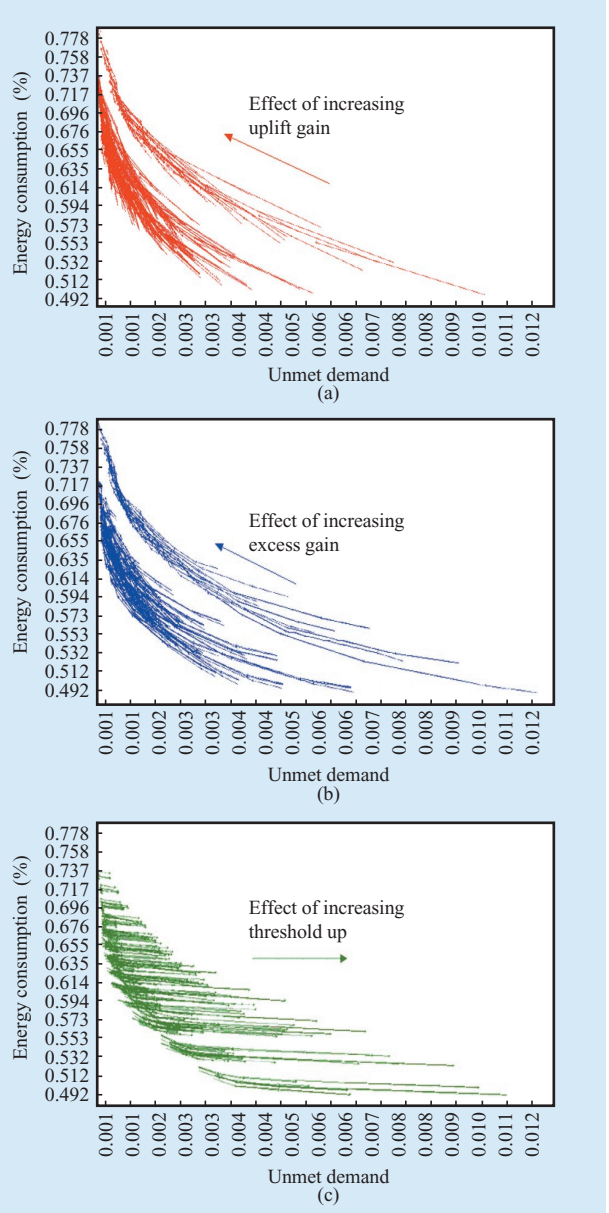

Figure 7

Energy consumption vs. unmet demand for (a) uplift gain vector field; (b) excess gain vector field; and (c) threshold up vector field. Vectors are in the direction of the increasing parameter.

than in the right fork in the curve. In the right arc of the curve, where threshold up is higher, the algorithm does not take epochal variations into account, is surprised by the unpredicted workload transients, and incurs larger amounts of unmet demand.

Thus, the threshold up vector field chart shows that, for workloads that exhibit periodic workload increments, the epochal algorithm can be extremely effective in reducing

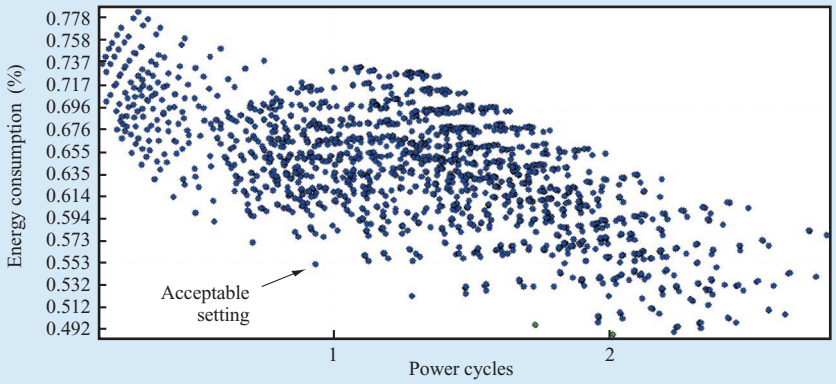

Figure 8

Energy consumption vs. power-on cycles per server day for Domino workload.

unmet demand. We also note that if the workload had no periodicity, the wishbone curve would collapse into a single hyperbola, and threshold up would have little if any effect. We actually see this for other, less periodic workloads.

The effects of changing the other parameters of the algorithm are similar to the effect of changing uplift gain and excess gain, and are not reproduced here. To summarize,

- Increases in threshold down cause energy to rise and unmet demand to fall along a given arc of the wishbone.
- Increases in desired utilization cause energy to fall and unmet demand to rise along a given arc of the wishbone.
- Increases in window size cause energy to rise and unmet demand to fall along a given arc of the wishbone.

Figure 8 shows energy consumption as a function of the number of power-on cycles per day per server for the Domino workload. The first observation is that for this workload, a large number of parameter settings result in significant energy savings, with well under one power-on cycle per server day, and quite reasonable unmet demand. For example, the point marked by the arrow shows one such setting, with a 45% energy savings, a 0.5% unmet demand, and 0.935 power-on cycles per server day.

UNC ibiblio.org Web server workload

The second workload that we used to evaluate the algorithm was obtained from the main *ibiblio.org* file transfer protocol and Web server called "Divahouse." This server, located at the University of North Carolina at Chapel Hill (UNC), receives several million hits per day. The workload, which consists of samples taken every two minutes, spans 27 days in January 2002.

The autocorrelation is shown in **Figure 9(a)**. Because the samples are at two-minute intervals, a workload with a strong daily periodicity would have a peak at a lag of 720 samples, and a one-week periodicity would have a peak at a lag of 5040 samples. These periodicities are evident from the autocorrelation, but they are not as pronounced as in the Domino workload; therefore, we would not expect the long-term epochal prediction technique to be as effective as in that case.

Figure 9(b) shows the energy versus unmet demand tradeoff chart for Divahouse. A 20% energy savings is accompanied by 0.5% unmet demand, which on the surface appears to be very reasonable algorithmic performance. Note the absence of the wishbone shape, implying that the algorithm is not effectively exploiting what little epochal behavior is exhibited by the workload.

However, for this workload, it proved more difficult to obtain large energy savings with a small number of power-on cycles. As the indicated point in **Figure 10** exemplifies, to achieve energy savings of the order of 20% requires 3.29 power-on cycles per server day, although unmet demand remains at a comfortable 0.5%. We feel that this is partly because of the high-frequency components of the workload, and partly because the algorithm has (and takes) an opportunity to turn a server off or on every two minutes (compared to every fifteen minutes for the Domino workload), resulting in a higher frequency of power cycling. (The different colors for the points in this energy vs. power-on cycles chart reflect differing values of normalized unmet demand. A blue point signifies <1% unmet demand, green is <2%, yellow <3%, and red >3% unmet demand.)

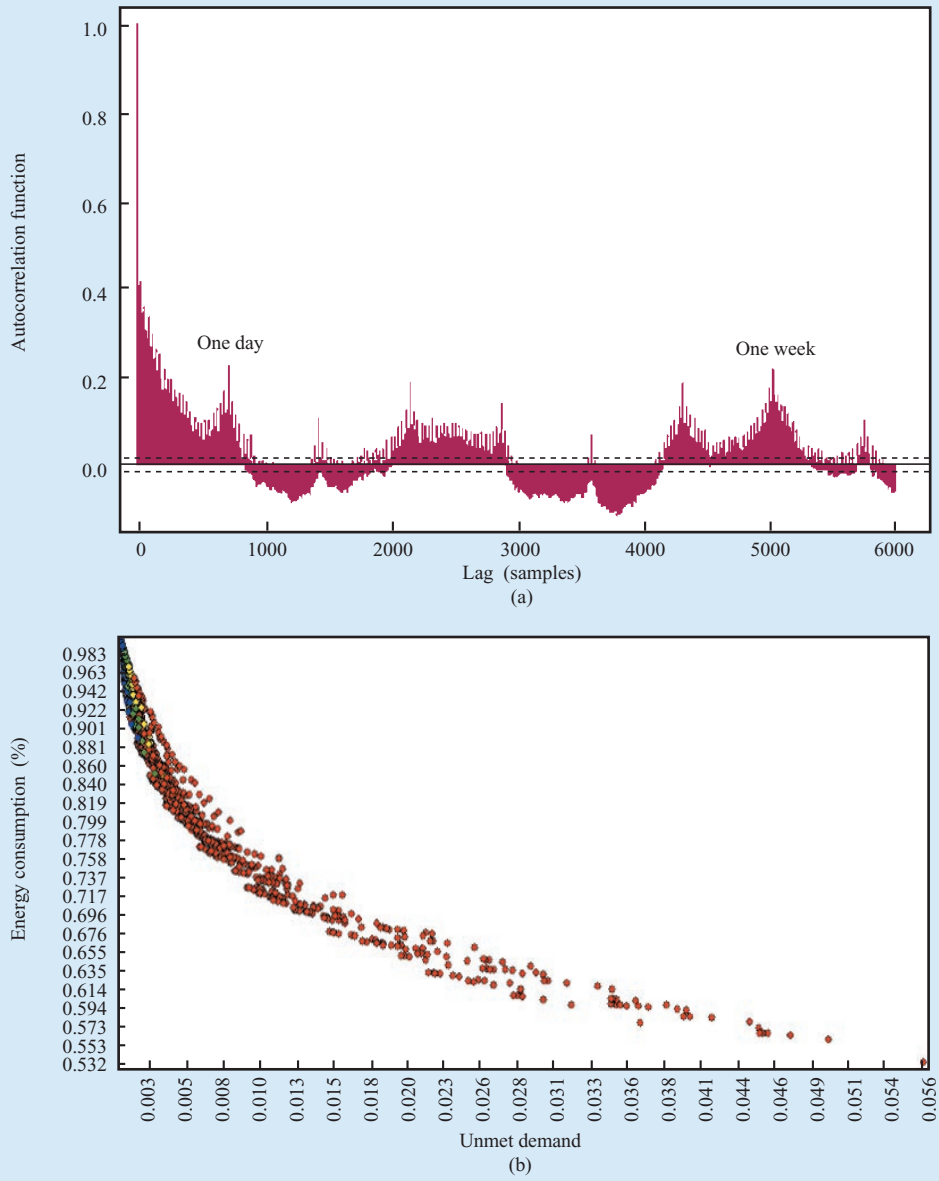

Figure 9

(a) Autocorrelation for Divahouse workload; (b) energy consumption vs. unmet demand for *ibiblio.org* Divahouse Web server.

Sydney 2000 Summer Olympic Games Web site workload

The final workload we evaluated was obtained from 12 Web servers located in Sydney, Australia (**Figure 11**). This data spans 27 calendar days representing activity before and during the 2000 Summer Olympic Games, with five minutes between samples. The energy savings in this highly overprovisioned application range from 54% to 80%. Unmet demand [Figure 11(a)] was typically <1%, and many settings achieved <1 power-on cycle per server day [Figure 11(b)].

Summary of empirical results

Table 1 summarizes the performance of the algorithm for the workloads we considered. We also compare the energy consumption obtained via our algorithm to the energy consumption for a "perfect" algorithm, in which a) only sufficient servers are powered on to service the workload

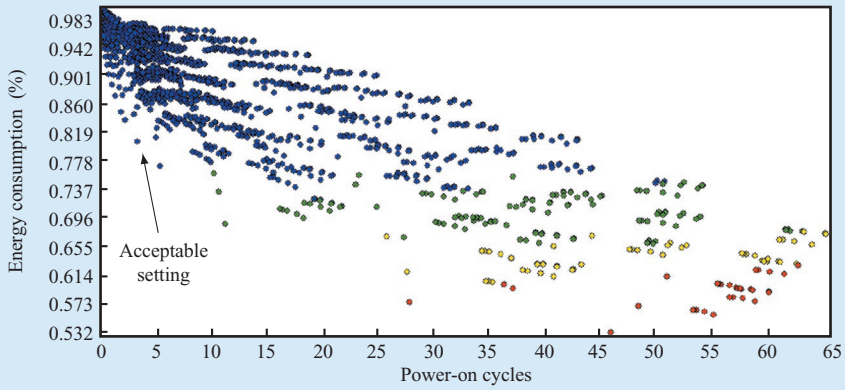

Figure 10

Energy consumption vs. power-on cycles per server day for Divahouse Web-serving workload.

Table 1 Algorithm performance summary.

Workload	*Energy consumption, compared with all servers powered on, "perfect" algorithm (%)*	*Energy consumption, compared with all servers powered on, CoolRunnings algorithm (%)*	*Unmet demand, compared with total demand (%)*	*Power-on cycles per server day*	*Comments*
Lotus Domino database	37	55	0.5	<1	Strongly seasonal enterprise database. High dynamic range.
"Divahouse" Web server	32	80	0.5	3.29	Not overprovisioned. Weak seasonality. Low dynamic range.
Sydney 2000 Summer Olympics Web server	14	20	<1	<1	Highly overprovisioned. Low seasonality. High dynamic range.

at any given time, b) servers power up or down instantaneously and can accept workload immediately upon power-up, and c) the number of power-on/off cycles is unconstrained. We think that the poor energy performance on the "Divahouse" workload, compared with the perfect algorithm, is largely due to its high-frequency components, which are difficult to accommodate with a power-on/off cycle minimization constraint.

5. Practical considerations

In some architectures, power-based workload management can be accomplished by invoking existing workload-balancing functionality and informing it that a given server has to have its workload removed or that a new server is available for use. This is likely to be the case in the IBM Director/Network Dispatcher scenario outlined below. Power control can be accomplished by communicating the power control demands to a service processor resident in the server whose power is to be controlled. This is usually accomplished over either an in-band or out-of-band management network, using established application programmable interfaces (APIs). Depending on its functional capabilities, a server can be either powered off or put into a suspended/hibernation state. Note that for RAS purposes, hibernation is equivalent to a power-off cycle and thus should not be invoked with total abandon, but it could result in much shorter power-off/on cycles.

Standalone implementation

In one implementation, all of the functions of workload-aware power management could reside in a general-purpose distributed system environment having the workload execution, workload measurement, workload balancing, power management algorithm, and power control means.

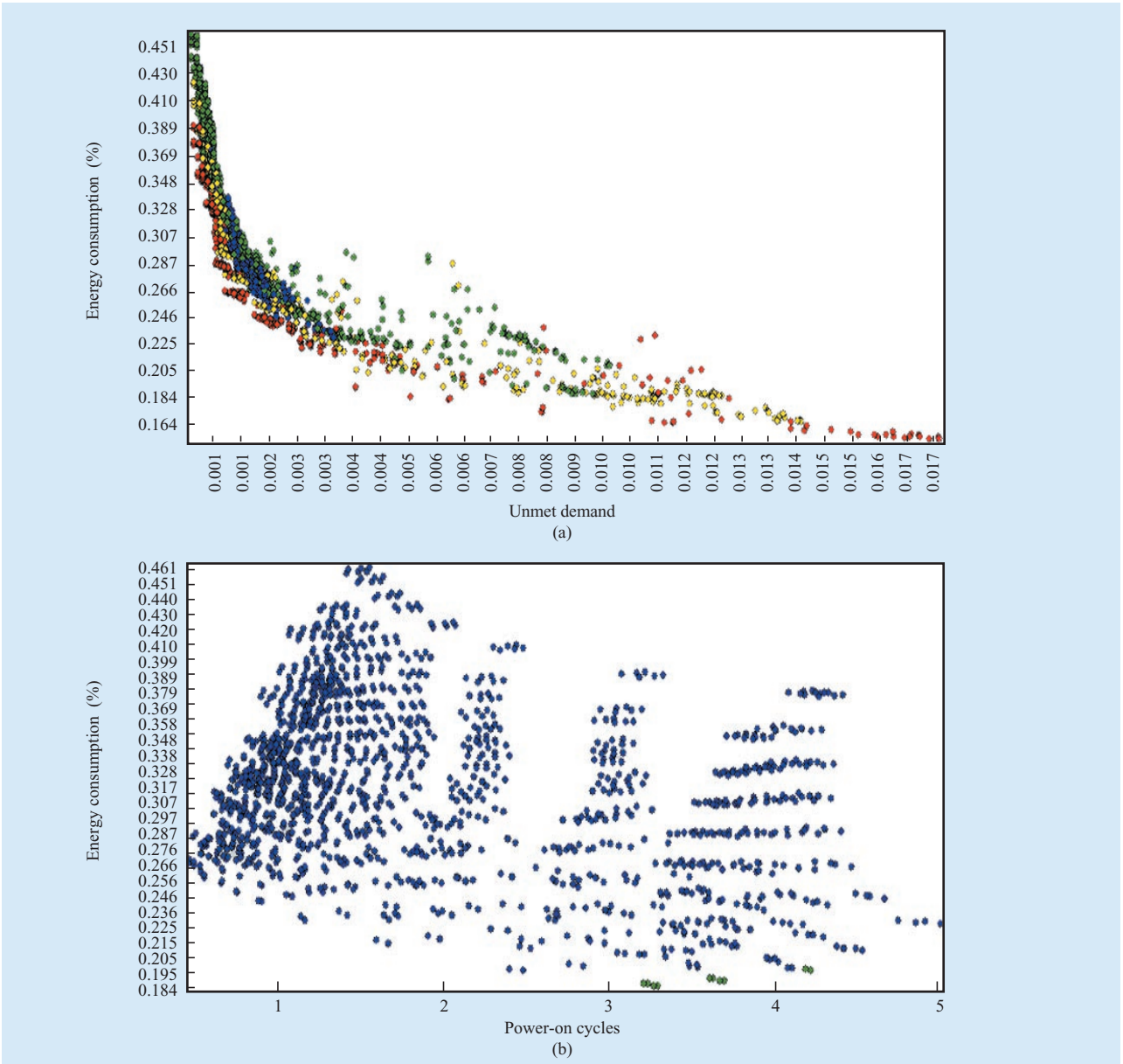

Figure 11

Olympic Games (Sydney, Australia, 2000): (a) Energy vs. unmet demand; (b) energy vs. power-on cycles.

When no distinguished server is designated as being responsible for the collection of utilization data and the execution of the power management algorithm, a leader election algorithm can be run to determine which server of the currently extant group should run this algorithm and make the power control decisions. Subsequently, the leader can also invoke the workload management and power control functions to implement the decision.

Implementation integrated into IBM Director and Network Dispatcher

Figure 12 illustrates how power management functionality could operate in an environment containing IBM Director [1] and Network Dispatcher, a component of IBM Websphere Edge Server [14].

There are several possible sources of utilization information for the algorithm. The figure shows it being provided by a utilization calculation thread that is shown

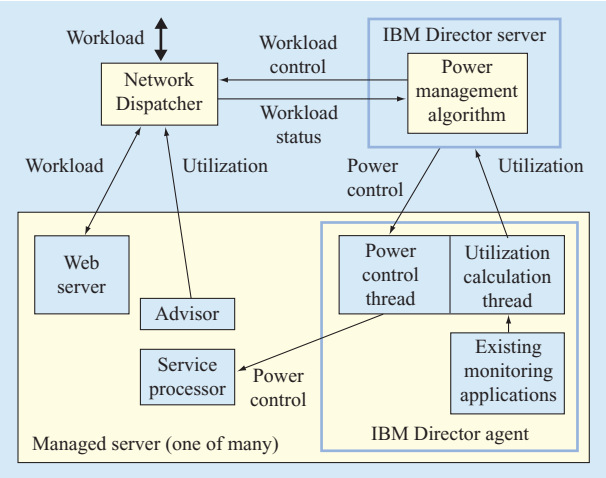

Figure 12

Integration of intelligent power management into IBM Director.

as part of IBM Director, perhaps using the Capacity Manager database, but it could alternatively be obtained from Network Dispatcher as part of workload status, in which case actual response time measurements would be available; or, rather than having the utilization calculation thread obtain the data from the existing Director monitoring applications, it could obtain utilization data from the Network Dispatcher advisor.

6. Conclusions and recommendations for future work

This paper describes a real-time power management algorithm that is applicable to a parallel complex of servers having a parallelizable, migratable workload. The algorithm is designed to minimize power utilization, unmet demand, and server power cycles. It accomplishes this by using short-term and long-term workload history and a simple workload prediction method to anticipate short-term fluctuations in workload. The algorithm then issues power cycling commands as appropriate to reduce power while meeting this predicted workload, with some upside reserve. Our experiments with this algorithm on email/database and Web-serving workloads show that energy savings of 20% or more can readily be achieved, with typically less than 1% of unmet or deferred demand and an average of one power cycle per server day. If the workload has a high dynamic range, or if the system is substantially overprovisioned, much higher energy savings are obtained.

The work we have described has led to several ideas for future work items. We have assumed daily and weekly epochs, which is a limiting assumption. It should be straightforward to improve the epochal algorithm to detect epochs automatically, perhaps using frequency analysis or autocorrelation of the workload. Another opportunity for improving algorithm behavior might be to look at the autocorrelation of upsets or spikes of missed demand, perhaps as a function of the lookahead window, and dynamically adjust parameters accordingly. It should also be possible to reduce the impact of sampling granularity on missed workload transients by enhancing the epochal algorithm to detect smeared or jittery epochal behavior, perhaps by further developing the lookahead concept discussed in the paper.

The dynamical model of the current algorithm assumes that it takes one sampling interval to power up and apply workload, or power down a server. In reality, this interval can be either significantly greater or significantly less than one sample interval. For example, a server may be able to be brought into and out of standby very quickly (instant on/off), or it may take a long time to adjust the workload for a given application (high workload inertia). The performance of the algorithm can be improved by incorporating these different dynamical constraints on a scenario-by-scenario basis.

We currently use easily measured indicators of server workload (CPU utilization, etc.) that are independent of any application. This of course has the advantage of generality, but perhaps better algorithm performance could be obtained for a particular application if application-specific indicators of utilization were used, such as transaction response time. However, because high-level indicators of performance may not offer diagnostic insight into the root causes of poor performance, it seems reasonable to formulate diagnostic correlations between basic utilization parameters that are easily measured on the server and the application-level figures of merit.

There are several factors to consider when deciding which machines in a complex to power on or off. Ultimately, a decision criterion must be formulated on the basis of a combination of factors such as trying to equalize power-on cycles across all machines, preferentially powering off machines in hot spots, preferentially powering off machines that are unhealthy or require rejuvenation, machine characteristics, and other aspects we have not yet considered. For example, if different machines have different power dissipations or different capacities, we need to adapt the algorithm to power on the machines that maximize capacity subject to minimizing power; this appears to be a linear programming problem. Our current position is that we should keep the more powerful machines powered on all the time, and preferentially cycle the lower-throughput machines for vernier capacity control, but analysis remains to be performed.

In reality, server group membership can change over time for a variety of reasons, such as failures, recoveries, new servers being added to a complex, etc. The algorithm currently assumes a static server group membership, but can easily be extended to accommodate dynamic groups.

Acknowledgments

We thank Paul Jones and John Reuning of the University of North Carolina at Chapel Hill for the Divahouse workload data, and Herb Lee of IBM Yorktown Research for the Domino workload data. Luis Lastras, Jim Challenger, and Paul Dantzig of IBM Yorktown Research provided access to the Sydney Olympics data, for which we are grateful. The comments of one of the reviewers were especially helpful in improving this paper.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Microsoft Corporation or Linus Torvalds.

References

1. See *http://www-1.ibm.com/servers/eserver/xseries/systems_management/xseries_sm.html*.
2. T. Mudge, "Power: A First Class Design Constraint for Future Architectures," presented at the High Performance Computer Conference, Bangalore, India, December 2000; *IEEE Computer* **34,** No. 4, 52–58 (April 2001).
3. A. Vahdat, A. Lebeck, and C. Ellis, "Every Joule Is Precious: The Case for Revisiting Operating System Design for Energy Efficiency," *Proceedings of the 9th ACM SIGOPS European Workshop*, September 2000, pp. 31–36.
4. C. S. Ellis, "The Case for Higher Level Power Management," *Proceedings of the 7th IEEE Workshop on Hot Topics in Operating Systems* (*HotOS-VII*), 1999, pp. 162–167.
5. P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The Case for Power Management in Web Servers," *Power-Aware Computing*, Robert Graybill and Rami Melhem, Eds., Kluwer/Plenum Series in Computer Science, January 2002, Ch. 1, pp. 1–31.
6. J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing Energy and Server Resources in Hosting Centers," *Proceedings of the 18th Symposium on Operating Systems Principles* (*SOSP'01*), October 2001, pp. 103–116.
7. J. Chase and R. Doyle, "Balance of Power: Energy Management for Server Clusters," *Proceedings of the 8th Workshop on Hot Topics in Operating Systems* (*HotOS-VIII*), May 2001, pp. 163–165.
8. E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems," *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, September 2001; *Technical Report DCS-TR-440*, Department of Computer Science, Rutgers University, New Brunswick, NJ, May 2001.
9. International Business Machines Corporation, Websphere Software Platform; see *http://www.ibm.com/*.
10. A. Iyengar, J. Challenger, D. Dias, and P. Dantzig, "High-Performance Web Site Design Techniques," *IEEE Internet Computing* **4,** No. 2, 17–26 (March 2000).
11. J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger, "Adaptive Algorithms for Managing a Distributed Data Processing Workload," *IBM Syst. J.* **36,** No. 2, 242–283 (1997).
12. International Business Machines Corporation, BladeCenter; see *http://www.ibm.com/*.
13. See *http://www.ibm.com/products/*; *http://www.lotus.com/products/*.
14. See *http://www-3.ibm.com/software/webservers/edgeserver/index.html*.

Received November 18, 2002; accepted for publication July 15, 2003

David J. Bradley *IBM Systems Group, Research Triangle Park, P.O. Box 12195, Durham, North Carolina 27709 (drdave@us.ibm.com).* Dr. Bradley is a Senior Technical Staff Member in the xSeries Architecture and Technology Department. He received a B.E.E. degree in 1971 from the University of Dayton, and an M.S. degree and a Ph.D. degree, both in electrical engineering, from Purdue University in 1972 and 1975, respectively. After joining IBM in 1975, he was one of the "original 12" engineers who developed the IBM Personal Computer in Boca Raton, Florida, where he invented "Ctl-Alt-Del." Since then he has developed a variety of Personal Computer systems and founded and managed the PC Architecture Department. Dr. Bradley has most recently appeared as the answer on Final Jeopardy during College Week.

Richard E. Harper *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (reharper@us.ibm.com).* Dr. Harper is a Research Staff Member in the Server Technologies Department at the Thomas J. Watson Research Center. He received a B.S. degree in physics in 1976 and an M.S. degree in aeronautical engineering in 1977 from Mississippi State University, and a Ph.D. degree in computer systems architecture from the Massachusetts Institute of Technology in 1987. Between 1987 and 1995 he developed parallel flight and mission-critical computing system architectures at the Charles Stark Draper Laboratory in Cambridge, Massachusetts. Between 1995 and the IBM Thomas J. Watson Research Center in 1998, he worked on system analysis and architecture at Stratus Computer in Marlboro, Massachusetts, a manufacturer of commercial fault-tolerant computers. At IBM, he has worked on performance analysis and architecture of high-end NUMA/SMP architectures, and proactive problem prediction and avoidance technologies. Dr. Harper is an author or coauthor of numerous patents and technical papers.

Steven W. Hunter *IBM Systems Group, Research Triangle Park, P.O. Box 12195, Durham, North Carolina 27709 (hunters@us.ibm.com).* Dr. Hunter is a Senior Technical Staff Member in the xSeries Architecture and Technology Department. He received a B.S. degree in electrical engineering from Auburn University in 1984, an M.S. degree in electrical engineering from North Carolina State University in 1988, and a Ph.D. degree in electrical engineering from Duke University in 1997. Since 1997 Dr. Hunter has been part of the xSeries organization, where he has most recently been a lead architect and designer for the BladeCenter product. He is licensed as a Professional Engineer, and is a Senior Member of the IEEE and an Adjunct Professor at North Carolina State University. He holds more than a dozen patents spanning hardware and software. Dr. Hunter has published numerous papers and has presented at a variety of conferences and symposiums.