

# Power-aware Resource Allocation in High-end Systems via Online Simulation<sup>\*</sup>

Barry Lawson  
Department of Math and Computer Science  
University of Richmond  
Richmond, VA 23173, USA  
blawson@richmond.edu

Evgenia Smirni  
Department of Computer Science  
College of William and Mary  
Williamsburg, VA 23187-8795, USA  
esmirni@cs.wm.edu

## Abstract

Traditionally, scheduling in high-end parallel systems focuses on how to minimize the average job waiting time and on how to maximize the overall system utilization. Despite the development of scheduling strategies that aim at maximizing system utilization, parallel supercomputing traces that span long time periods indicate that such systems are mostly underutilized. Much of the time there is simply not enough load to keep the system fully utilized, although time periods do exist where system utilization levels peak at nearly 95%. In this paper, we propose a new family of scheduling policies that aims at minimizing power consumption and cooling costs by selectively choosing to power down (or put in “sleep” mode) parts of the system during periods of low load. Our goal is the development of a scheduling mechanism that adaptively adjusts the number of processors to the offered load while meeting predefined service-level agreements (SLAs). This scheduling mechanism uses online simulation, i.e., lightweight simulation modules that can execute while the system and its scheduler are in operation, and can guide resource provisioning in parallel systems. Detailed experimentation using traces from the Parallel Workloads Archive indicates that the proposed online mechanism is a viable alternative to conserve energy while meeting performance-based SLAs.

## Keywords

Power-aware scheduling, resource allocation, performance evaluation, parallel workload characterization, online simulation

<sup>\*</sup>This work was partially supported by the National Science Foundation under grants ITR-0428330, CCR-0098278, and ACI-0090221.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

ICS '05, June 20-22, Boston, MA, USA.  
Copyright ©2005, ACM 1-59593-167-8/06/2005 ... \$5.00

## 1 Introduction

Scheduling policies in today’s high-end distributed memory systems aim at maximizing the system utilization and usually operate using variations of the basic FCFS discipline. Industrial-strength schedulers that are widely accepted by the supercomputing community, including the Maui scheduler [13], PBS [15], and IBM LoadLeveler [7], operate using variations of FCFS, allow for more than a single waiting queue, and provide a variety of configuration parameters. Together, these allow the system administrator to customize the scheduling policy according to the site’s needs. Backfilling [5, 8, 14] is often found in the core of these schedulers and is used as a more efficient way to execute waiting jobs in the queue.

With backfilling, the scheduler does not always execute the waiting jobs in the queue in simple FCFS order. Depending on the job width (i.e., the requested number of processors by the job), certain jobs are allowed to surpass others in the queue if their execution does not delay certain previously submitted jobs. The goal of backfilling is to decrease system fragmentation and increase system utilization [14, 18] by using otherwise idle processors for immediate execution. Various versions of backfilling have been proposed [9, 14, 16, 20], sharing the goal of maximizing system utilization given a fixed number of processors and an incoming stream of jobs.

Despite all these efforts to maximize system utilization, workload traces from the Parallel Workloads Archive [6] indicate that there are significant time periods during which the workload is insufficient to keep the system well utilized. This observation, when combined with the growing trend in the development of techniques for power management [2], motivates the work presented here. If the system cannot be fully utilized then powering down parts of it would yield significant gains in the costs for electricity consumption and cooling.

Several recent research efforts have focused on resource allocation in hosting centers, with an emphasis on power management. These efforts aim at scheduling various competing applications in order to minimize the center’s operating costs [1, 2, 3, 4, 17]. These works span from the area of capacity planning and QoS provisioning, to different application classes using feedback based techniques that use dynamic load monitoring [17, 19], to techniques for optimizing power con-

sumption in a clustered environment by powering off parts of the system or having it operate using dynamic voltage scaling with a slower speed [3, 17], to techniques for providing negotiated service level agreements [1] (SLAs).

Here, we consider a similar problem but in the context of parallel scheduling in a distributed memory large-scale cluster of multiprocessors. In this context, we integrate the concept of scheduling and power management by proposing a supplement to the well established EASY backfilling policy [14]. We propose a scheduler that continuously monitors load in the system and selectively puts certain nodes in “sleep” mode, i.e., makes them unavailable for execution, after estimating the effect of fewer nodes on the projected job slowdown. Using online simulation, a flexible and inexpensive way to simulate the performance of scheduling policies when applied to jobs that are waiting in the queue, the system adaptively selects the minimum number of processors that are required to meet negotiated service level agreements (SLAs). Detailed simulations using workloads from the Parallel Workloads Archive indicate that indeed this methodology is effective and can result in significant savings in the number of “active” (thus power consuming) processors during the system lifetime without significant compromises on predefined SLAs.

We stress that our contribution is not specific to the EASY scheduling policy; rather, it can be applied to *any* scheduling policy that the system uses. Our general methodology for online simulation can be used with any scheduler, in systems that support multiple levels of SLAs in the form of different classes of service, and even in systems that may suffer from a forced power outage due to overheating that is triggered by partial cooling failures. Although interesting, these topics are outside the scope of this paper and are subjects for future work.

This paper is organized as follows. Section 2 presents analysis of the utilization levels of select traces from the Parallel Workloads Archive, motivating this work. Section 3 provides a description of the online simulation mechanism, the power-aware scheduling policies, and a detailed description of the performance results. Conclusions and future work are summarized in Section 4.

## 2 Motivation

In this section we present the workload characteristics of four workload traces from the Parallel Workloads Archive [6] that motivate this work. More specifically, we use the following workload logs:

- **CTC**: log containing 77 222 jobs executed on a 512-node IBM SP2 at the Cornell Theory Center from June 1996 through May 1997;
- **KTH**: log containing 28 490 jobs executed on a 100-node IBM SP2 at the Swedish Royal Institute of Technology from September 1996 through August 1997;
- **SDSC-SP2**: log containing 59 725 jobs executed on a 128-node IBM SP2 at the San Diego Supercomputer Center from April 1998 through April 2000;

- **Blue Horizon**: log containing 243 314 jobs executed on a 144-node with 8 processors per node IBM SP at the San Diego Supercomputer Center from April 2000 through January 2003.

From the traces, for each job we extract the arrival time of the job (i.e., submission time), the job width (i.e., number of processors requested), the estimated duration of the job, the actual duration of the job, and the job completion time.

As a first step, to examine workload variability across the four logs, we concentrate on changes in job arrival intensity and demands. The first column of graphs in Figure 1 illustrates the time evolution of the arrival process in two of the four logs by presenting the total number of arriving jobs per week. We observe significant variability in the job arrival intensity across workloads but also across time within each workload. The arrival rate of jobs in the CTC and KTH workloads changes at most by a factor of three from week to week, while in the SDSC-SP2 and Blue Horizon logs we observe changes of as much as a factor of seven.

The second column of Figure 1 illustrates the time evolution of the service process. The service requirement of each job in a parallel system has a two dimensional property given by the job execution time and the job width. Here, in an effort to collapse both dimensions into a single one, we calculate each job’s *demand*, defined as the product of the service time<sup>1</sup> multiplied by the number of processors. Again, there is a significant variation not only across workloads but also across time, with the SDSC-SP2 and Blue Horizon workloads showing significantly higher demands than CTC and KTH.

Given such workload variations across time, we next examine the system utilization from week to week. The third column of Figure 1 illustrates the system utilization across time and reflects the native scheduling strategies of the four machines.<sup>2</sup> Observe that CTC’s utilization reaches merely a high of 60%, while during most of the time it operates at 50% utilization. KTH is a higher utilized system, as its utilization reaches up to 80% during some weeks. SDSC-SP2 and Blue Horizon are the most utilized systems, with utilizations that reach in some weeks almost their full capacities, but there are several weeks, especially for Blue Horizon, in which the system operates in low utilization.

In summary, analysis such as that depicted in Figure 1 shows significant variability of workload intensities and demands, resulting in systems that are almost never equally utilized across time. In the following section, we elaborate on scheduling policies that aim at power savings by leveraging periods of low system utilization, as driven by the observations above.

---

<sup>1</sup>For simplicity of presentation, instead of presenting demand using the actual execution time of the job in seconds (which results unavoidably in very high numbers), we prefer to scale execution time to the fraction of the day.

<sup>2</sup>The utilization graphs were computed using for each job the actual duration and completion times as determined via the trace.

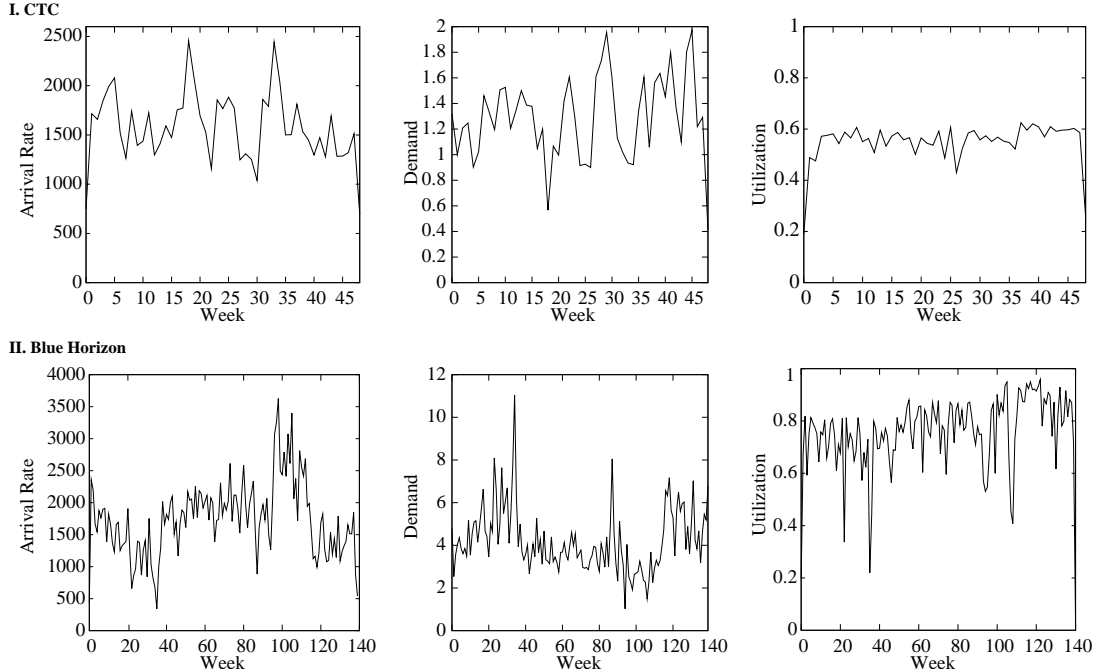


Figure 1. Total number of arriving jobs per week (first column), average demand per week (second column), and system utilization under the EASY scheduling policy (third column) for the CTC and Blue Horizon workloads. All graphs are presented as a function of time (weeks).

### 3 Power-aware Policies and Experimental Results

In this section, we provide a description of the power-aware scheduling policies, including an online simulation mechanism, and also provide detailed results and analysis from our simulation experiments. Our goal is to provide a set of policies that can choose to reduce the number of active processors in the system in order to lower associated costs, but without dramatically impacting job performance on those systems. Furthermore, systems are sure to experience transient periods of alternating high and low utilization. Therefore, the policies must also be able to adapt appropriately to fluctuating workload conditions.

#### 3.1 Experimental Methodology

In this work, we consider high-performance batch scheduling systems in which jobs run to completion (i.e., no preemption). Job scheduling is performed using the EASY scheduler [14], one of the most basic backfilling algorithms. Backfilling requires users to provide an estimate of the job execution time [14], which allows certain jobs to surpass others in the queue and begin execution provided that their execution does not delay certain previously submitted jobs. This allows for a tighter packing of the jobs in the system and increases system utilization. The effectiveness of backfilling depends on the accuracy of user estimates, as it is the estimates that allow jobs to surpass other jobs in the queue and create meaningful schedules. If the actual job execution is higher than its estimate, then the job is killed. Prior studies have shown that user

estimates are notoriously inaccurate [10, 12, 14]. Nonetheless, even in the presence of inaccurate estimates, backfilling reduces system fragmentation, increases system utilization, and improves performance. For all experiments that we show in this paper, unless otherwise clearly stated, we use the actual execution times as the (exact) user estimates.

To generate the results presented in subsequent sections, our simulation experiments are driven using the four workload traces from the Parallel Workloads Archive described in Section 2. In our simulations we use the arrival time of the job (i.e., the submission time), the number of processors requested, the estimated execution time of the job, and the actual execution time of the job. We do not use the job completion times from the traces in the remainder of this paper because these numbers are associated with the scheduling policies used on the corresponding systems.

In the results below, we consider *aggregate* performance measures, i.e., average statistics computed for an entire simulation run, giving a single quantifiable measure for the entire workload (e.g., average slowdown for all jobs). As expressed in Section 2, the workload arrival and demand patterns vary significantly across time. Consequently, we also consider *transient* performance measures, i.e., per-week snapshot statistics that are plotted versus experiment time to illustrate how a certain policy reacts to these variabilities in the workload.

Because we are interested in reducing the number of active processors to reduce power and cooling costs but without severely impacting job performance, there are two critical

measures of interest to consider here. From the job perspective, the performance measure of interest is each job’s bounded slowdown [14] defined by

$$s = 1 + \frac{d}{\max\{10, v\}}$$

where  $d$  and  $v$  are respectively the queuing delay time and the actual service time of the job.<sup>3</sup> From the system perspective, utilization provides a measure of the goodness of the power-aware policy, and can be used to understand the balance between job performance and power savings. If the utilization is very high (which can result from lowering the number of active processors too much), job performance will suffer as a consequence; if utilization is very low (which can result from more active processors than needed), job performance will improve, but supporting system costs will rise. For a system with a fluctuating number of active processors, the utilization is computed as the ratio

$$\bar{x} = \frac{\sum_j D_j}{\sum_i (t_{i+1} - t_i) \cdot N_i}$$

where  $D_j$  is the demand (number of processors times execution time) of a completed job  $j$ ,  $N_i$  is the number of active processors during the  $i$ th time interval  $[t_i, t_{i+1}]$ , and changes in the number of active processors occur at distinct points  $t_i$ .

### 3.2 Two-Level Policy

In this section, we describe the first, and simplest, of two general power-aware scheduling policies. In the two-level policy, the system is permitted to fluctuate back and forth between the maximum number of processors  $U$  and a system-specific minimum number of processors  $L$ . Initially, the system starts with  $U$  active processors. The decisions of when to lower and when to raise the number of active processors are defined as follows.

- Assume that the number of active processors is  $U$ . If at any point in time the number of currently executing processors is less than or equal to  $L$ , and if the maximum number of processors needed by any single job currently in the queue is less than or equal to  $L$ , the system reduces the number of active processors to  $L$ .
- Assume that the number of active processors is  $L$ . If a job arrives to the system requesting more than  $L$  processors for execution, the system increases the number of active processors to  $U$ .

With respect to the EASY backfilling scheduler, raising or lowering the number of active processors potentially entails corresponding rescheduling. That is, if a job is scheduled to begin execution but is not currently executing, raising or lower the number of active processors may (but is not guaranteed to) cause the job to be scheduled for execution at a time different than its originally scheduled time. If the number of active processors is being raised from  $L$  to  $U$ , there are no negative consequences from the job perspective—raising the number of

<sup>3</sup>The maximizing function in the denominator reduces the otherwise dramatic impact that very small jobs can have on the slowdown statistic. The 10 second execution time parameter is standard in the literature [14].

active processors can only improve the scheduled time of execution. However, if the number of active processors is being lowered from  $U$  to  $L$ , then scheduled times for execution must be reevaluated (since those times were scheduled with respect to a larger pool of available processors). Though not always, in some cases this causes the scheduled execution time of a job to be delayed. Given our goal to reduce power costs, we believe this delay is appropriate provided that, in general, the delays for jobs are not exorbitant.

For a given system, the choice of the parameter  $U$  is obvious—simply the maximum number of processors in the system. The choice of the parameter  $L$  is less obvious, yet provides the system administrator with some flexibility in tuning system performance. For the results to follow, we selected  $L$  as follows. Let  $\bar{x}$  be the system utilization achieved using all processors for a complete simulation run. Then let  $\hat{x}$  be a “stepped” value greater than  $\bar{x}$  (e.g., if  $\bar{x} = 0.75$ , let  $\hat{x} = 0.80$ ; if  $\bar{x} = 0.82$ , let  $\hat{x} = 0.85$ ). Then define  $L$  according to  $L \simeq \lceil U \cdot \hat{x} \rceil$ . The motivation for defining  $\hat{x}$  as a “stepped” value as described is to provide the system with some flexibility for scheduling in terms of available processors. Our selections of  $\hat{x}$  and  $L$  for each of the four workloads are given in Table 1.

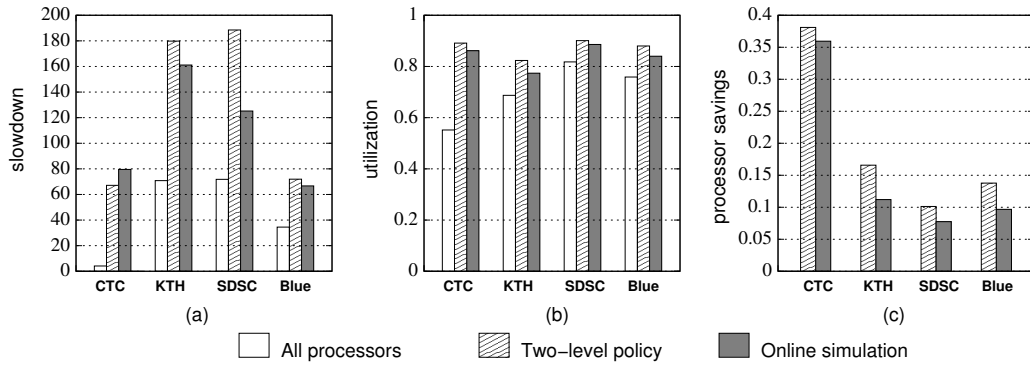
**Table 1. Choices of the two-level policy parameters  $U$  and  $L$  for the experiments to follow.**

Workload	$U$	$\hat{x}$	$L$
CTC	512	0.60	308
KTH	100	0.75	75
SDSC-SP2	128	0.85	110
Blue Horizon	1152	0.80	922

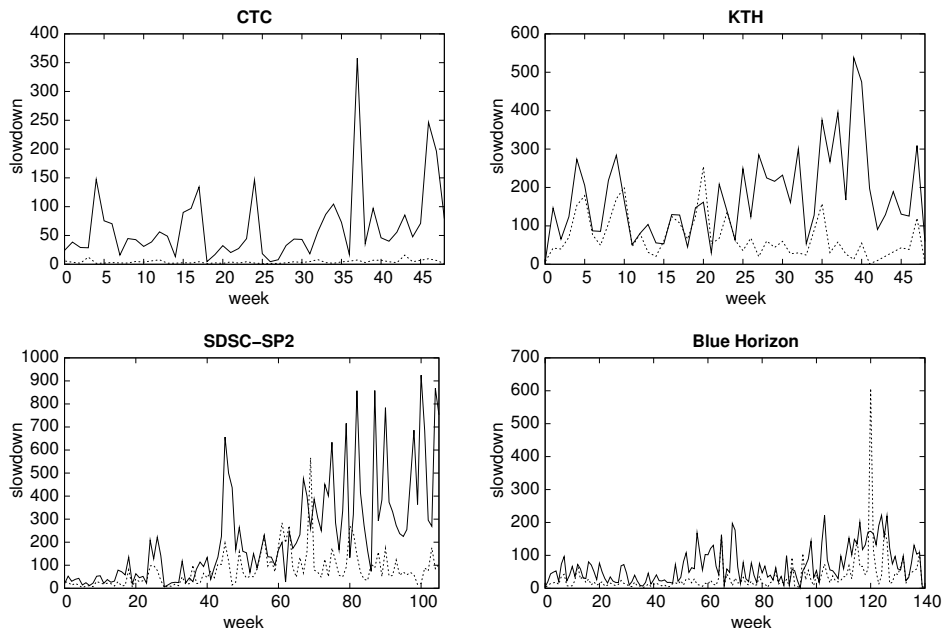
In a practical setting, the system administrator will not have the luxury, as we do, of knowing a priori the system utilization in the presence of all processors. Nonetheless,  $L$  need not be static for perpetuity, but can instead be periodically adjusted up or down by the system administrator to appropriately address perceived system performance. If utilization is not known a priori, the scheduler could alternatively monitor past system utilization, using the past to predict the future (e.g., using a predefined time window or exponentially discounted history).

For each of the four workloads, Figure 2 compares the aggregate bounded slowdown and system utilization for a simulation run with all processors for the duration versus a simulation run using the two-level policy with  $U$  and  $L$  as given in Table 1. Also provided is the overall proportion of processor savings, i.e., the proportion of processors that the system was able to have in the inactive state.

As depicted, using the two-level approach the proportion of processor savings is approximately 10% or more, with corresponding system utilizations of 80% or more for each of the workloads. Notice that, when using the two-level policy, the aggregate slowdown degrades by a factor of approximately two (no more than two and a half) for all but the CTC workload. Therefore, using this simple policy we can expect significant savings in processor power and cooling costs provided that an approximate factor of two-fold increase in slowdown is permissible. We argue that this is a reasonable price to pay in the context of our goals.



**Figure 2. (a) Aggregate slowdown and (b) aggregate utilization for a full-processor simulation run, a two-level policy run, and an online simulation policy run; (c) proportion of processors maintained in an inactive state.**

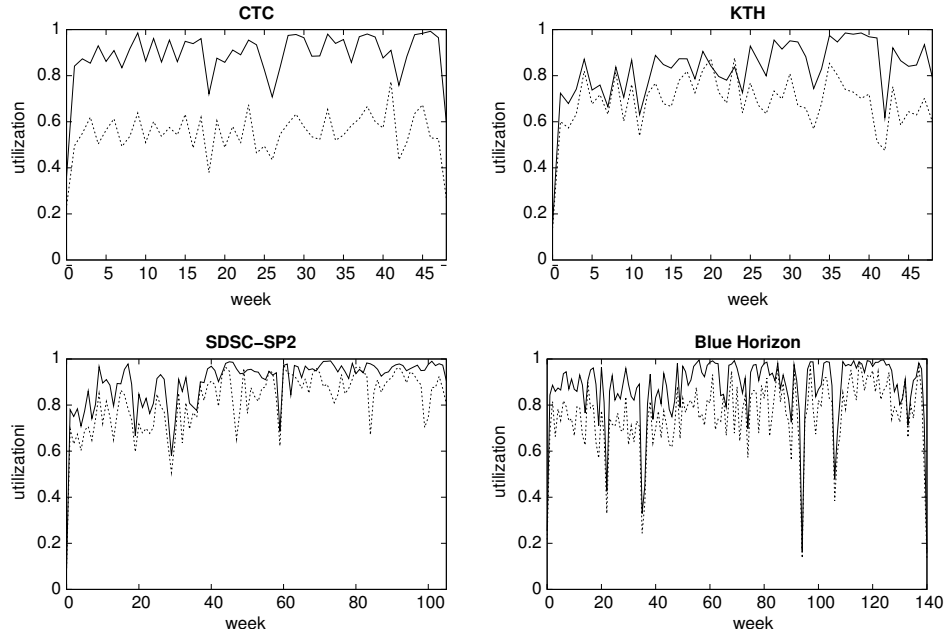


**Figure 3. Slowdown as a function of time for each of the four workloads. For solid lines, the two-level policy is used; for dashed lines, the maximum number of processors is always present.**

Special note should be made of the CTC workload: in its original form, the system is very under-utilized at only 55%. Accordingly, job slowdowns are essentially negligible. Slowdown will clearly increase as utilization increases, so making a factor comparison of slowdown in the CTC case is not meaningful. Notice that using the two-level policy, the processor savings are dramatic and the corresponding utilization is near 90%, yet the aggregate slowdown is comparable to that obtained in the other systems when using *all* processors.

Figure 3 depicts transient slowdown versus wallclock time, providing insight into the ability of the system to respond to workload fluctuations across time. The solid line represents the average job slowdown for all jobs that complete in the corresponding week under the two-level policy; the dashed line represents the same metric using all possible processors.

Generally speaking, the area under the curve corresponding to the two-level policy is greater, resulting in increased job slowdown at the gain of reducing the number of active processors. Notice, however, that there are weeks in which the two-level policy outperforms the maximum-processor policy (e.g., week 120 for Blue Horizon). With the CTC workload aside, the two curves for a given workload are generally similar, with the exception of notable increases in slowdown at the tails of the KTH and SDSC-SP2 workloads. These increases are consistent with the workload analysis provided in Section 2—KTH experiences a significant increase in the arrival rate, while for SDSC-SP2 the demand increases dramatically with time. These figures provide further evidence that, in fact, the variability in workload demands causes striking differences in performance across time.



**Figure 4. Utilization as a function of time for each of the four workloads. For solid lines, the two-level policy is used; for dashed lines, the maximum number of processors is always present.**

It should be noted that the two-level policy in no direct way reacts to any measure of perceived system performance. The number of active processors is increased or decreased based solely on the processor requirements of incoming jobs, and not the system utilization or job slowdown. This observation, along with the transient increase in slowdown evident at the tails for KTH and SDSC-SP2 in Figure 3, provides sufficient motivation to consider additional power-aware policies that are more capable of adapting to transient workload conditions and responding to measures of system performance. Such policies are considered in the next section.

Figure 4 depicts transient utilization versus wallclock time. The solid line represents use of the two-level policy and the dashed line represents the policy when the maximum number of processors is always available. These figures are consistent with the overall increase in system utilization. Furthermore, notice the variability of utilization across time, especially for the Blue Horizon results. Note that a few very low utilization peaks exist even using the two-level policy. Referring to Figure 1, these peaks correspond to times when demand is high but the corresponding arrival rate is low, or to times when demand is low but the corresponding arrival rate is high. Moreover, the variability across time motivates more adaptive power-aware policies, presented in the next section.

### 3.3 Online Simulation Policy

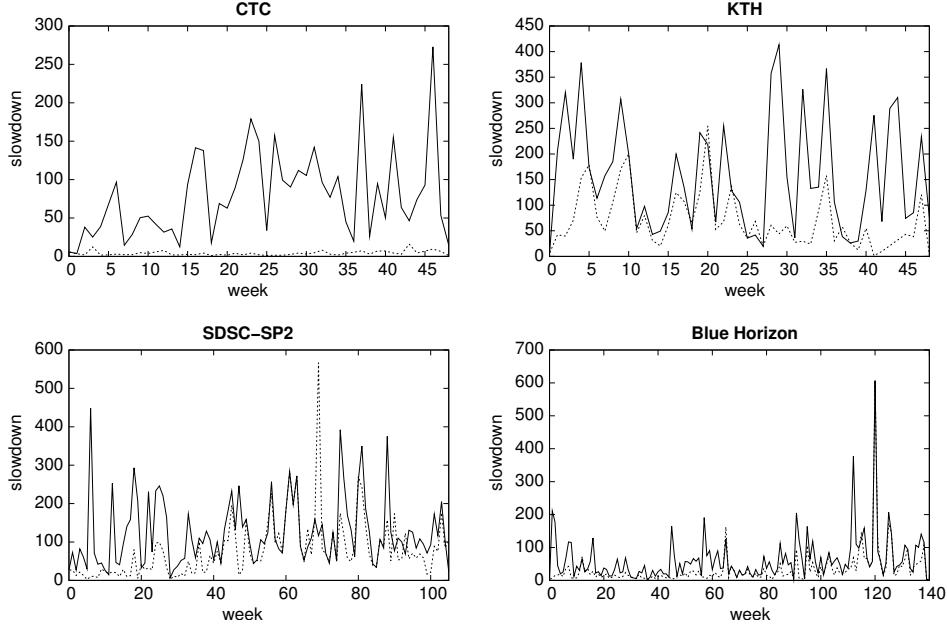
As the results in the previous section show, the two-level policy is suitable for reducing power and cooling costs while keeping the system at a high level of utilization at the cost of a reasonable increase in job slowdown. However, the two-level policy, while relatively simple, does not always adapt well in the presence of fluctuating workload conditions. Furthermore,

the two-level policy reacts based solely on the processor demands of jobs without taking into account any quantified measure of system performance. These issues motivate the development of an additional set of power-aware policies.

Online simulation has been previously shown an effective technique in scheduling for high-end systems [11], although in a context different than the current one. In the previous context, online simulation permits the scheduler to adjust its parameters (specifically, the number of queues) to best match the workload conditions. In the current power-aware context, online simulation can be used instead to predict the number of processors that can effectively be powered down without severely impacting job scheduling performance.

In the online simulation policy (OLS), the system executes several lightweight simulation modules online in an attempt to predict the best parameters for the future. More specifically, each  $k$  units of time, the system executes multiple online simulations. The initial state of each online simulation is the current state of the actual system, except that each simulation assumes a system with a different number  $l$  of active processors, where  $l$  is between  $L$  and  $U$  ( $L$  is subject to the same minimum-processor requirements as in the two-level policy). Each online simulation then simulates the scheduling and execution of those jobs (both executing and queued) currently in the actual system but in the presence of only  $l$  processors, and computes the average job slowdown experienced by those jobs. These online simulations are sufficiently lightweight for practical purposes—they execute very quickly and give meaningful feedback nearly instantaneously.

To choose the winner from among the online simulations, the system requires an a priori service level agreement (SLA). In



**Figure 5. Slowdown as a function of time for each of the four workloads. For solid lines, the OLS policy is used; for dashed lines, the maximum number of processors is always present.**

this context, the SLA is a predefined threshold  $S$  in terms of acceptable slowdown. The system chooses the online simulation with the lowest number of active processors  $l$  whose computed average slowdown for the existing jobs in the system is less than or equal to  $S$ . If such a simulation is found, the system proceeds to reduce the number of active processors to  $l$ , similar to that described for the two-level policy. If the system is unable to find any online simulation with  $l < U$  processors such that the SLA is met, the system increases (if necessary) the number of active processors to  $U$ . The corresponding rescheduling that applies to the two-level policy also applies here (see Section 3.2).

In summary, this policy aims at reducing the amount of power and cooling needed by the system by reducing the number of active processors. The policy reacts to transient workload conditions by evaluating different scheduling parameters on-the-fly, in an attempt to best address the transient nature of the workload present at that instant in time. Below, we expound on specific implementations of this general policy, providing detailed performance results.

### 3.3.1 OLS with Power-Aware Adjustment

For the OLS policy with power-aware adjustment, we permit the number of active processors to increase or decrease in any nonnegative increment (subject to scheduling constraints such as the maximum number of processors needed by any job in the system). More specifically, given SLA  $S$ , the results to follow were generated using the following implementation.

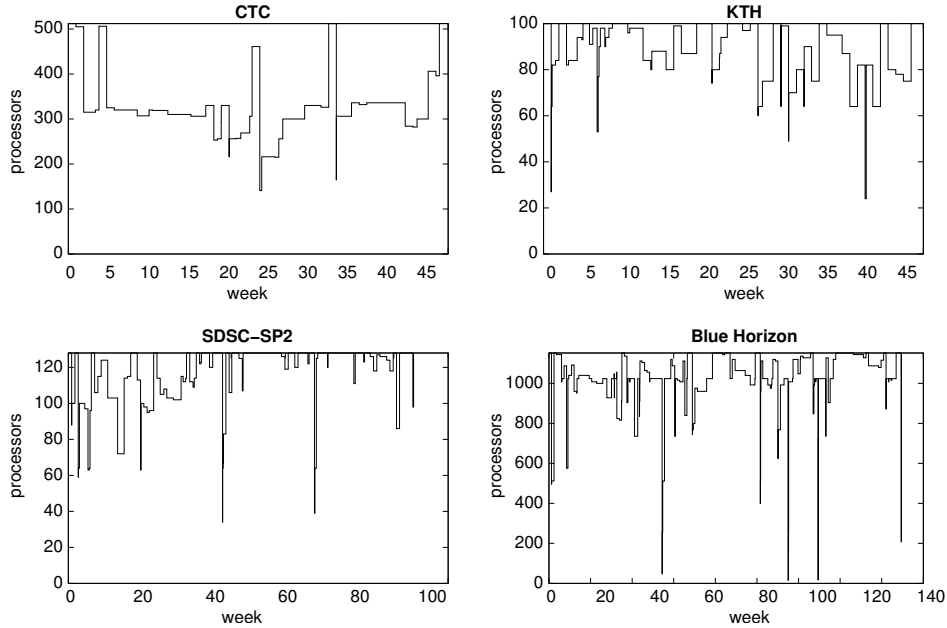
- Define the lower bound  $L$  for the number of processors to simulate to be the maximum of {the maximum number of processors needed by any single job in the queue, the

number of currently executing processors}.

- Perform an online simulation with  $l = L$  processors.
- If the computed average slowdown for jobs in the queue is less than or equal to  $S$ , then  $l$  becomes the number of active processors in the system; otherwise, define  $l = \lceil (l + U)/2 \rceil$ , and repeat the online simulation process.
- No further simulations are performed whenever  $l = U$ .
- Furthermore, if any job arrives requesting more than the currently available processors (but no more than  $U$ ), then the number of processors is increased to meet the processor requirement of that job.

Again, the potential rescheduling for queued jobs necessary for the two-level policy applies. Also note that because these online simulations execute very quickly, the corresponding overhead is negligible.

Figure 2 also depicts the aggregate slowdown and utilization for OLS with  $S = 200$  (the OLS slowdown threshold) versus the maximum-processor and two-level policies, and depicts the proportion of processor savings achieved using OLS. When compared with the aggregate performance of the two-level policy, the OLS policy generally decreases the overall slowdown at the expense of a slight decrease in utilization. There is a slight increase in the slowdown for CTC, but note the significant decrease in slowdown for SDSC-SP2. The message here is clear: by attempting to address job performance on the fly, the system is not as able to blindly focus on increasing system utilization as in the two-level approach. Nevertheless, by permitting the system to adapt the number of processors in a variable manner, the payoff is still commendable with processor savings (not including CTC's dramatic savings) around 10% while maintaining reasonable increases in slowdown that



**Figure 6.** Number of active processors as a function of time for each of the four workloads using the OLS policy with power-aware adjustment.

are clearly within the target SLA<sup>4</sup>.

More interestingly, in Figure 5 note how in the transient context the OLS reacts according to the varying workload. Compared with the corresponding two-level policy figures in Figure 3, the OLS policy adjusts for the heavy tail phenomena in KTH and SDSC-SP2 transient slowdown by reducing the magnitude of the tail, spreading the slowdown more evenly across the entire length of the simulation. (Note that the vertical scales for KTH and SDSC-SP2 in Figure 5 are smaller than in Figure 3.) Because the online simulations occur only at the beginning of each week, the number of opportunities to reduce active processors is fewer than with the two-level policy, yet the OLS policy performs quite well. More frequent online simulations are likely to result in even higher utilization, at the expense of the overhead of more frequently decreasing and increasing the number of active processors.

Figure 6 shows the change in the number of active processors across time using the OLS policy with power-aware adjustment. Although there are a few periods of frequent activity, generally the relative frequency of changes in the number of active processors is reasonable. Furthermore, the CTC figure provides meaningful insight here. The CTC workload with the maximum-processor policy yields only about 55% utilization. The CTC figure in Figure 6 tends to track just above 55% of the maximum number of processors, providing sufficient room for reasonable scheduling while maintaining a much higher utilization. A similar analogy can be made for the three other more highly utilized systems, though not as visually obvious. To address the periods of frequent activity, one potential mea-

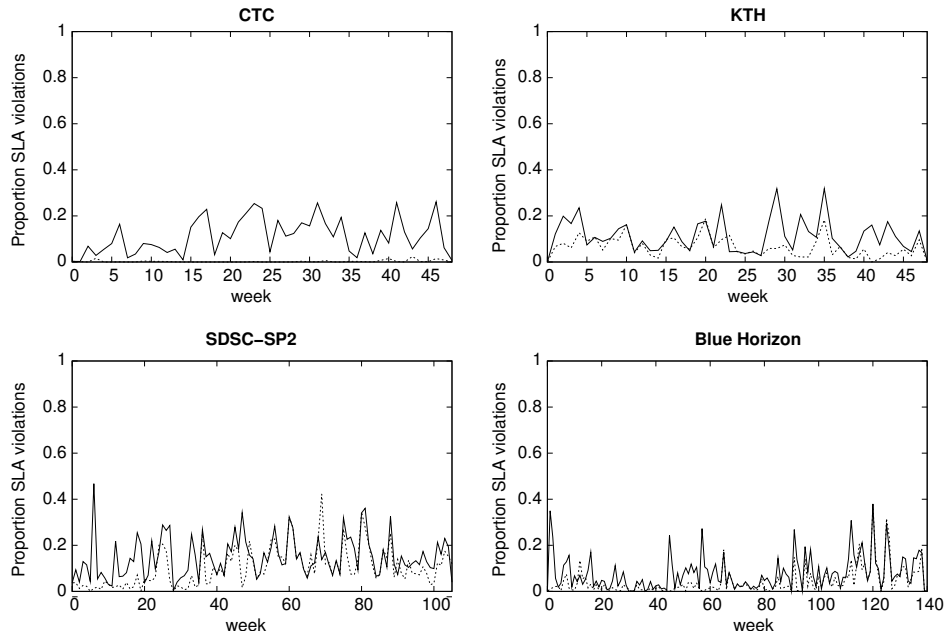
<sup>4</sup>We experimented with different slowdown thresholds and results are qualitatively the same as with those presented here.

sure for a more practical implementation would be to avoid dramatic drops in the number of active processors (e.g., near weeks 40 and 70 for SDSC-SP2 and near weeks 40, 90, and 100 for Blue Horizon). As shown, these dramatic drops tend to persist for a very short time, so the system would maintain a similar overall utilization, but with much less processor up/down activity, if these drops were avoided.

Figure 7 depicts the proportion of jobs per week that violate the SLA, i.e., that have a slowdown greater than  $S$ . The solid lines represent results using the OLS policy while the dashed lines represent results obtained using the maximum-processor policy. This figure provides strong evidence that the OLS policy is performing as intended. Note that, except for the otherwise highly under-utilized CTC workload, the solid lines track the dashed lines rather well. The performance loss in slowdown that results from the OLS policy, in comparison with the maximum-processor policy, is acceptable, especially given the processor savings that result.

### 3.3.2 Sensitivity Analysis

Figure 8 provides insight into the sensitivity of the OLS policy with respect to the frequency of online simulations, and correspondingly the number of adjustments in number of active processors. This is motivated by an effort to minimize the number of wake/sleep cycles in order to minimize hardware failures due to thermal stresses and/or electrical surges. Consistent with intuition, there is generally a slight decrease in the slowdown, utilization, and processor savings as online simulations become less frequent (i.e., fewer opportunities to reduce the number of active processors). Nonetheless, the results for all three frequencies are comparable, suggesting that the policy can perform well even when the number of active



**Figure 7. Proportion of jobs violating the SLA per week as a function of time. For solid lines, the OLS policy is used; for dashed lines, the maximum number of processors is always present.**

processor reductions should be restrained.

Finally, to examine the policy sensitivity to inexact user estimates, we investigated the performance of the OLS policy when only inexact user runtime estimates are used for scheduling. Although omitted here for brevity, the results obtained using inexact runtime estimates are qualitatively similar to the results based on actual runtimes presented earlier in this paper, demonstrating that the policy is robust even in the presence of inexact estimates.

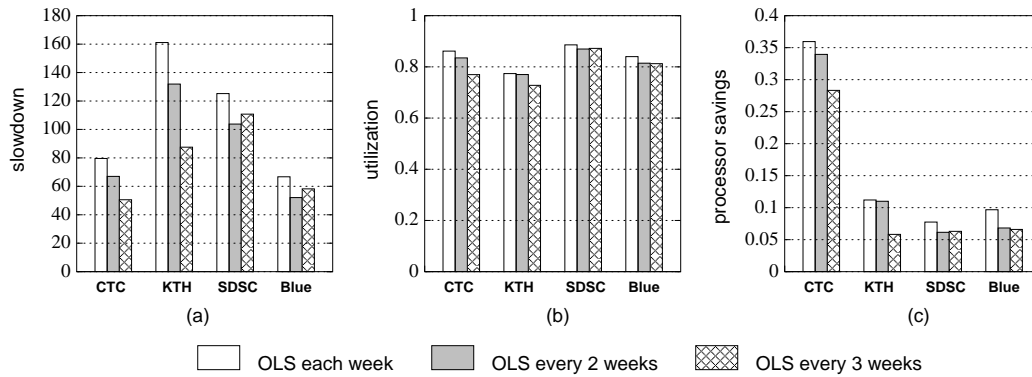
## 4 Conclusions and Future Work

We presented the effectiveness of a family of scheduling policies that aims at minimizing the power consumption of high-end distributed memory systems while maintaining certain user-negotiated SLAs. Using traces from the Parallel Workloads Archive and detailed simulations of the EASY scheduling policy, we showed that it is not necessary to operate the system at full capacity at all times, reducing operating costs while still meeting SLAs. We demonstrated that it is possible to adjust the number of processors in the system that operate in “sleep” mode in an adaptive way in order to meet severe fluctuations in the workload arrival patterns and service demands. One of the presented policies is based on online execution of lightweight simulation modules, which prove effective in modeling scheduling micro-scenarios that use different numbers of processors. This online simulation approach emphasizes speed (allowing for low-overhead, on-the-fly decisions of the number of active processors while the actual system is still in operation) in addition to accuracy (allowing the system to adapt appropriately in response to the transient nature of the workload).

We stress that our contribution is not specific to the EASY scheduling policy but instead can be applied to *any* scheduling policy that the system uses. In the future, we intend to examine variations of the proposed OLS policy that consider how to better deal with inexact user estimates, reduce the average job response time and improve system performance and individual job slowdown by using a multiple-queue backfilling strategy [10], support multiple levels of SLAs in the form of different classes of service, and even consider the policy in systems that may suffer from a forced power outage due to overheating that is triggered by partial cooling failures. For the case of partial cooling failure, a more graceful approach would be to monitor the temperature and as it approaches a critical level begin selectively deactivating nodes that are idle. We are currently working with the system administrator of our local parallel cluster to quantify the savings in energy consumption that can occur when we power down or put to sleep parts of the system. This will allow us to enhance our online simulation modules with accurate estimates of the power consumption itself.

## 5 Acknowledgments

We thank Dimitrios Nikolopoulos for useful conversations that helped shape this topic. We also thank Tom Crockett for his feedback in earlier versions of this work. We thank Dror Feitelson for the availability of workload traces via the Parallel Workloads Archive. We also thank Dan Dwyer and Steve Hotovy for providing the CTC workload; Lars Malinowsky for the KTH workload; Victor Hazlewood for the SDSC-SP2 workload; and Travis Earheart and Nancy Wilkins-Diehr for the SDSC Blue Horizon workload.



**Figure 8. (a) Aggregate slowdown, (b) aggregate utilization, and (c) proportion of processor savings (relative to the maximum-processor policy) for OLS using power-aware adjustment with decreasing frequency of online simulation.**

## 6 References

- [1] J. Chase, D. Anderson, P. Thankar, and A. Vahdat. Managing energy and server resources in hosting centers. In *Proceedings of the 18th Symposium on Operating Systems Principles SOSP'01*. ACM, 2001.
- [2] J. Chase and R. P. Doyle. Energy management for server clusters. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*. ACM, 2001.
- [3] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proceedings of the Second Workshop on Power Aware Computing Systems*, pages 179–196, 2002.
- [4] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. In *Proceedings of USENIX Symposium on Internet Technologies and Systems, USITS'03*, 2003.
- [5] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling — a status report. In *Job Scheduling Strategies for Parallel Processing (JSSPP 2004)*, pages 1–16. Springer-Verlag, 2004. LNCS vol. 3277.
- [6] Parallel Workload Archive. <http://www.cs.huji.ac.il/labs/parallel/workload/>.
- [7] IBM LoadLeveler. <http://publib.boulder.ibm.com/clresctr/windows/public/llbooks.html>.
- [8] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the Maui scheduler. In *Job Scheduling Strategies for Parallel Processing (JSSPP 2001)*, pages 87–102. Springer-Verlag, 2001. LNCS vol. 2221.
- [9] P. Keleher, D. Zotkin, and D. Perkovic. Attacking the bottlenecks in backfilling schedulers. *Cluster Computing: The Journal of Networks, Software Tools and Applications*, 3(4):245–254, 2000.
- [10] B. Lawson and E. Smirni. Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. In *Job Scheduling Strategies for Parallel Processing: (JSSPP 2002)*, pages 72–87. Springer Verlag, 2002. LNCS vol. 2537.
- [11] B. Lawson and E. Smirni. Self-adaptive scheduler parameterization via online simulation. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, Denver, CO, April 2005.
- [12] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In *Job Scheduling Strategies for Parallel Processing (JSSPP 2004)*, pages 253–263. Springer-Verlag, 2004. LNCS vol. 3277.
- [13] Maui Scheduler Open Cluster Software. <http://mauischeduler.sourceforge.net/>.
- [14] A. Mualem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, June 2001.
- [15] Portable Batch System. <http://www.openpbs.org/>.
- [16] D. Perkovic and P. Keleher. Randomization, speculation, and adaptation in batch schedulers. In *Proceedings of Supercomputing 2000 (SC2000)*, November 2000.
- [17] V. Sharma, A. Thomas, T. F. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS management in web servers. In *Proceedings of the Real-Time Systems Symposium, RTTS'03*, pages 63–73, 2003.
- [18] D. Talby and D. Feitelson. Supporting priorities and improving utilization of the IBM SP2 scheduler using slack-based backfilling. In *Proceedings of the 13th International Parallel Processing Symposium*, pages 513–517, April 1999.
- [19] B. Urgaonkar and P. Shenoy. Brief announcement: Catclysm: handling extreme overloads in internet services. In *Proceedings of the 23rd ACM symposium on Principles of Distributed Computing (PODC '04)*, pages 390–390. ACM Press, 2004.
- [20] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramanian. An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration. *IEEE Trans. Parallel Distrib. Syst.*, 14(3):236–247, 2003.