Self-Adaptive Power Management of Idle Nodes in Large Scale Systems

YONGPENG LIU[(1)], HONG ZHU[(2)], KAI LU[(1)], AND XIAOPING WANG[(1)]
[(1)] National University of Defense Technology, China
[(2)] Oxford Brookes University, UK

The real workload on a large scale computer system varies from time to time. Often it has many idle nodes during most operation time. These idle nodes consume energy, but do nothing useful. To save the huge amount of energy wasted by such active idle nodes, most modern compute nodes are equipped with multiple level dynamic sleep mechanisms to reduce power consumption. However, awaking sleeping nodes takes time. The deeper a node sleeps, the less energy it consumes, but the longer wakeup latency. How to balance between the systems energy consumption and the response time is a key problem in the power management of large scale systems. This paper proposes a self-adaptive approach to manage the sleep states of idle nodes to achieve low energy consumption and high performance at the same time. The proposed approach has two distinctive features. First, idle nodes are hierarchical organised. In this model, idle nodes are classified into several groups according to their sleep states. Each group contains nodes of same level of sleep depth and forms a reserve pool of a certain readiness level. When a resource is requested, nodes in the pool of highest level of readiness are preferentially allocated. When the nodes in the pool of the highest readiness level are not sufficient, the nodes in the pool(s) of next level(s) of readiness are allocated. After each allocation and reclaim of nodes, the numbers of nodes in each level of pools are adjusted by changing the sleep depth of the nodes up and down. Thus, the reserve pools can be maintained for high performance requirement. When resources are released from applications, they are placed back to reserve pools and put into different levels of sleep states to save energy. Second, the sizes of reservation pools are self-adaptive. Obviously, a key factor that affects the effectiveness of the idle node management is the sizes of the reserve pools. Fixed sizes of reserve pools would not be effective due to the time varying nature of workload on large scale systems. The proposed approach employs a self-adaptive mechanism in which the sizes of reserve pools are dynamically adjusted during the execution of the system according to how well the research pools meet the need of computation resources. Our experiments demonstrated that our approach can significantly improve energy efficiency in large scale systems without significant scarification of performance.

Keywords: Large scale computer systems, Power Management, Dynamic Sleep, Idle node

1. INTRODUCTION

To satisfy the steadily rising demands on computing performance, both the number of compute units in data/compute centers and the system integration density grow rapidly. Consequently, in the past years, the so-called Moore's law of power consumption has been observed; that is,'*the power consumption of computer nodes doubles every 18 months*' [Feng 2003]. Power management has becomes a grave challenge to the development and operation of large scale computing systems.

Large scale high performance computing systems consume a tremendous amount of energy. According to the recent TOP500 list of supercomputers [Meuer et al. 2012], the average power consumption of Top10 systems is 4.65 MW. The peak power consumption of the most power consuming supercomputer, i.e. the K computer, reaches 12.659 MW, which equals the power usage of a middle scale city. In 2006, US servers and data centers consumed around 61 billion kWh at a cost of about 4.5 billion US Dollars. This is about 1.5% of the total US electricity

This research was partly supported the National High Technology Research and Development Program of China (863 Program) under grant No.2012AA01A301,the NSF of China under Grant No.61272141,No.60903059 and No.60903044,and the EU FP7 project MONICA on Mobile Cloud Computing under grant No.PIRSES-GA-2011-295222.
Authors' addresses: Yongpeng Liu, Kai Lu and Xiaoping Wang, School of Computer Science,National University of Defense Technology, Changsha 410073, P.R.China, email: liuyp@nudt.edu.cn. Hong Zhu, Department of Computing and Communication Technologies, Oxford Brookes University, Oxford, OX33 1HX, U.K., email: hzhu@brookes.ac.uk.

consumption or the output of about 15 typical power plants [U.S. Environmental Protection Agency 2007]. In 2007, the electricity consumption of global cloud computing was 623 billion kWh which is larger than the 5th largest electricity demand country in the world, i.e. India [Cook 2012]. Many data center projects have been cancelled or delayed because of an inability to meet such enormous power requirements. High density power consumption causes overheating, which leads to problems of system reliability and availability. Huge construction costs of large scale systems are also incurred in order to accommodate the huge amount of energy demand.

On the other hand, the workload of data centers varies significantly with time where on average the resources in a large scale system typically sit at a low level of utilization. Consequently, a large number of nodes are idle in most time [Krioukov et al. 2011]. Unfortunately, nowadays nodes are not power-proportional. A node in idle state is highly energy inefficient. It can consume the power about 50% of its peak power [Mustafa et al. 2011]. In the result, idle nodes in large scale systems waste a huge amount of energy.

To reduce the power consumption of idle nodes, most compute nodes are now equipped with dynamic sleep mechanisms so that a node can be put into one of multiple sleep states when it is idle and be waken up when it is needed [Liu and Zhu 2010]. Each sleep state consumes less power than idling in the active state. The deeper a node sleeps, the less power it consumes, but the more energy and the more time delay are needed to wake it up. Considering the overhead of state transitions, the deepest sleep state obviously is not always the best choice for idle nodes. In this paper, we propose a self-adaptive solution for the management of sleep states of idle nodes in large scale systems to make an effective tradeoff between energy conservation and system response times.

2. RELATED WORK

Dynamic speed scaling and dynamic resource sleeping are two types of power management mechanisms widely supported in current information industry. Employing the dynamic speed scaling mechanism, the energy consumption of idle nodes can be reduced by scaling down the speed of its hardware equipment. However, even if all the components are scaled down to their lowest speeds, the power consumption of an active idle node is still significantly higher than that in a sleep state [Liu and Zhu 2010]. A huge amount of wasted energy can be still saved.

Dynamic cluster configuration is such a power management technique for large scale systems widely used in practice [Krioukov et al. 2011; Mustafa et al. 2011; Srikantaiah et al. 2008; Chase et al. 2001; Pinheiro et al. 2001]. The basic idea is to put idle nodes into a certain sleep state and waking them up on demand. To balance between energy and performance, most researchers on dynamic configuration of cluster focus on server consolidation, i.e. allocating tasks only on an appropriate active portion of the cluster. The remanding nodes thus become idle and are simply turned off [Horvath and Skadron 2008].

Gandhi et al investigated the importance of employing multiple sleep states for servers in data centers [Gandhi et al. 2011]. However, their solution does not dynamically manage the sleep depths of idle servers. Horvath et al. propose an energy management policy that exploits the multiple sleep states of idle servers [Horvath and Skadron 2008]. They predicate the incoming workload based on history resource utilization change and determine the optimal number of spare servers for each power states accordingly. Extra spare servers are put in the deepest possible sleep states. Obviously, the inaccuracy in workload prediction will have significant impact on either energy consumption or system performance. Our approach differs from their approach in that we dynamically adjust the number of nodes in different sleep depths according to the workload at runtime rather than predications.

[Xue et al. 2007] also manage a pool of active resources whose computing capacity is adjusted dynamically in accordance with the time-varying workload demand. However, in their power management solution spare nodes are simply turned off. They have not taken the advantages of multiple sleep states. Therefore, system performance is slowed down when additional computing

capacity is required, because it will take a long time to wake up the nodes switched off. In this paper, we explore the benefits of multiple sleep states to improve the energy efficiency of large scale systems with minimal sacrifices of system performance.

3. PROPOSED MANAGEMENT MODEL

This section presents the proposed idle node management model ASDMIN, which stands for *Adaptive Sleep Depth Management of Idle Nodes*. We will first review the key features of dynamic sleep mechanisms, then, present the structure and algorithms of the proposed model.

3.1 Basics of Dynamic Sleep Mechanism

The nodes in a cluster environment can be classified into two categories according to whether there is an application running on them, i.e. *busy nodes* and *idle nodes*. If a node has been allocated to any application, the node is busy. Otherwise, it is idle. An idle node, even in active standby state, does not produce any useful computation.

To reduce the energy waste on an idle node, it should be put into a low power consumption sleep state. Currently, a node is usually equipped with mechanisms that support multiple sleep states. For example, in ACPI specification [Hewlett-Packard Corporation et al. 2011], a node can be in one of the power states S0, S1, S2, S3, S4 or S5, where S0 is the active state, S1, ..., S5 denote different levels of sleep state with S5 as the deepest sleep state. The deeper a node sleeps, the less power it consumes, but the more energy and latency is required to wake it up. In each sleep state, the power consumption, wakeup energy and wake up latency are constants, which are denoted by P_i, E_i and D_i, $i = 0, \cdots, M$, respectively, where M is the number of supported sleep states of the node. These parameters satisfy formula (1) below.

$$\forall i, j \cdot (0 \leq i < j \leq M \Rightarrow (P_i > P_j \wedge D_i < D_j \wedge E_i < E_j)) \tag{1}$$

Note that, the power consumption in a sleep state is always lower than the power usage in the active state. Also, additional state transition energy is required to put a node into sleep and wake it up. Thus, to conserve energy by dynamic sleep mechanism, it is necessary to ensure the continuous period of sleep to be long enough so that the energy saved by sleeping is greater than the energy spent on state transition.

Moreover, a node in a sleep state is not available. Before providing any functional service, it must first be waken up. This means that the wake up latency may depress the response speed of the sleep node. The deeper a node sleeps, the longer the latency of wakeup. Consequently, sometimes it is not the best choice to put a node into its deepest sleep state when it becomes idle. An effective power management solution is required to schedule the appropriate sleep or waken up timing of idle nodes to maximize the energy efficiency of the whole system with minimal effect on system performance.

3.2 Structure of ASDMIN model

As shown in Figure 1, in ASDMIN model, the idle nodes are classified into a number of node groups according to their sleep depths. Each group is therefore a reserve pool of nodes of certain readiness. The higher the power consumption is, the higher the readiness level is. The pool of level i, denoted as B_i, is composed of all of the nodes with the same power consumption level P_i. From the cluster-wide point of view, node sleep depth management means to distribute the idle nodes among different groups of corresponding power states.

Assume that the number of nodes in B_i is N_i, and the total number of all idle nodes in the system is N, then we have that:

$$N = \sum_{i=0}^{M} N_i, \quad N_i \geq 0 \tag{2}$$

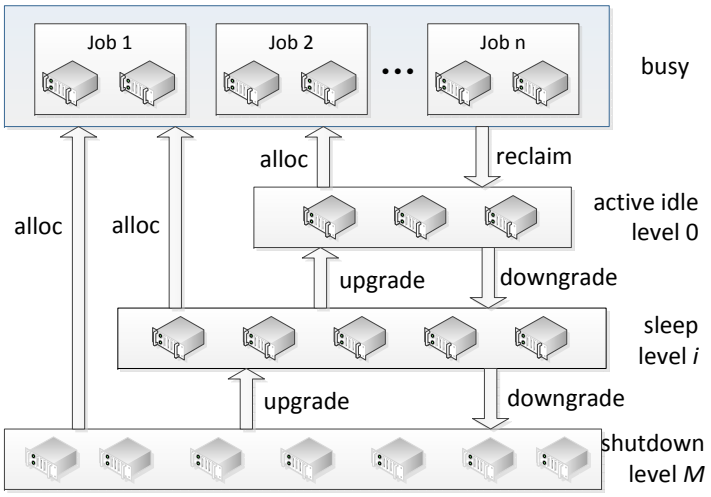

Figure 1. ASDMIN model for adaptive sleep depth management of idle nodes

Thus, the power consumption of all idle nodes, P_{idle}, equals

$$P_{idle} = \sum_{i=0}^{M} P_i \cdot N_i \tag{3}$$

In order to minimize the impact on system performance due to wake-up latency and to maximize the energy saving, ASDMIN's resource allocation policy will preferentially allocate nodes required by an incoming application from the highest reserve pool as many as possible, which has the shortest wakeup latency. When the nodes in the pool of the highest readiness level are not sufficient, the nodes in the pool(s) of next level(s) are allocated.

Let A be the number of nodes required by an incoming application. The highest level of reserve pools that can cover the need of A nodes is denoted by $Level(A)$. Formally, we have that

$$Level(A) = l \Leftrightarrow \sum_{i=0}^{l-1} N_i < A \leq \sum_{i=0}^{l} N_i \tag{4}$$

Assume that allocated nodes are waked up in parallel. Then, the overall wakeup latency for an allocation of A nodes, denoted by $Delay(A)$, is determined by the largest wakeup latency of the allocated nodes. By (1), we have that

$$Delay(A) = Max\{D_0, D_1, \cdots, D_l\} = D_l \tag{5}$$

where $l = Level(A)$.

Constrained by the service level agreement on response time, the level of the reserve pool that covers an allocation should not be lower than the requirement. In other words, enough nodes should be reserved in the pools whose sleep depth are lower than l. Therefore, we set a *reserve capacity threshold*, denoted by R_i, to control the minimum number of nodes in pool B_i. When the number of nodes in B_i is less than R_i, nodes in the lower level pools will be upgraded to fill the reserve capacity. On the other hand, when resources are released by applications, the number of nodes becomes more than the required, hence some idle nodes in a reserve pool will be downgraded to a sleep state as deep as possible.

Generally speaking, the less the reserve capacity thresholds for the highest reserve pools are, the more idle nodes are downgraded, and the less total power consumption by the idle nodes, hence the more energy is saved. However, the wakeup latency is longer and the performance loss caused by dynamic sleep is more significant. Thus, a trade-off between energy consumption and

performance must be made by assigning appropriate values to the reserve capacity threshold R_i. In the optimal scenario, the systems response speed matches closely to the speed when all nodes are put in the active standby state (i.e., in the highest reserve pool), and the energy conservation matches to that when all nodes are in deepest sleep state (i.e., in the lowest reserve pool). Unfortunately, this is hardly achievable with fixed values of R_i because of the unpredictably time varying nature of workload on large scale systems.

Our solution to this problem is to adjust the reserve capability thresholds dynamically and adaptively. The management algorithms are given in the next subsection.

3.3 Power Management Algorithms

The operation of ASDMIN model depends on two sets of algorithms: (a) the downgrading and upgrading algorithms for changing the sleep depths of the idle nodes, and (b) the allocation and reclamation algorithms for deciding which node is to be allocated for an application.

3.3.1 *Downgrading and Upgrading Idle Nodes.* Once an idle node is allocated to an application, the amount of reserve in the corresponding reserve pool is reduced. If the amount of reserve is lower than the required capacity, the pool needs to recruit nodes from lower pools. Consequently, this recruiting of nodes may cause reserve shortage of the lower pools. The progress of recruiting is thus a recursive process propagates from the top level to the lowest level as shown in Algorithm 1 below.

```
Algorithm 1. Upgrade();  (* Upgrade the power states of idle nodes *)
State Variables:
    <B0,B1 ..... BM>: Sets of idle nodes in the reserve pools;
Local Variable:
    k: integer;  (* The level of the target pool to recruit *)
Begin
   for i from 0 to M-1 do {
      k = i +1;
      while (Ni < Ri) {
          if (k > M)  break;
          if (Nk >= (Ri - Ni)) {
             Select (Ri - Ni) nodes from Bk and move them into Bi;
             Ni = Ri;
             Nk = Nk - (Ri - Ni);
           } else {
             Take all nodes in Bk into Bi;
             Ni = Ni + Nk;
             Nk = 0; k = k+1;
   }}}
End
```

When a node is reclaimed due to the finish of a task, it is put into a reserve pool. Consequently, the amount of resource in the pool may exceed the required capacity. The excessive nodes can therefore put into a deeper state of sleep. However, we do not want to put them into sleep immediately because this may result in the frequent changes of the states of nodes, which consumes energy, too. A question is that how long the delay should be so that the stability of the sizes of reserve pools and nodes' sleeping states can be maintained and the balance between performance and energy consumption can be achieved.

Let's first introduce a few more terminologies.

—*Piercing a reserve pool.* We say that a reserve pool is pierced at certain time moment during the operation of the system, if all the nodes in the pool are allocated but the resource is still

insufficient to meet the amount of required nodes.In this case,at least one node in the lower level reserve pool is used.

—*Continuous time period without piercing (CTPOP).* For a given reserve pool, it is the continuous period of time during which no piercing happened.

—*Length of CTPOP.* At a certain time moment in the operation of a system, the length of a CTPOP is the length of time period since the last piercing of the reserve pool.

The length of a CTPOP gives a good indication of how sufficient the resources in a reserve pool is with respect to the runtime characteristics of the application software. If the capability is too small, the pool will be frequently pierced, thus the length of CTPOP is short. Consequently, the performance of the system is slowed down. If the capability is too big, the reserve pool will be rarely pierced, and the length of CTPOP will be long. In this scenario, the power consumption is unnecessary and some of the nodes can be put in a deeper sleep state. Therefore, to balance between energy consumption and performance, the length of CTPOP must be managed at a certain ideal target value T_i for each reserve pool B_i. This target value is called *state continuance threshold*.

—*State continuance threshold (T_i).* For a given reserve pool B_i, it is the value set to judge whether its length of CTPOP is long enough.

By setting the state continuance thresholds for the reserve pools, we can manage the reserve pools as follows.

At a time moment, if the length t_i of the CTPOP of pool B_i is greater than the value T_i, the reserve pool B_i has not being pierced for a period long enough. Thus, its reserve capacity is superfluous than required. The over-reserved nodes in B_i can be downgraded into deeper sleep state to save energy. In such situation, a subset of B_i, notated as DS_i, is selected as the target nodes to be downgraded into the deeper pool B_{i+1}.

In other words, downgrading a node must meet two constraints: (a) the size of the reserve pool is greater than the reserve capacity R_i, and (b) the length of CTPOP is greater than the state continuance threshold T_i. Details of the algorithm are given below.

```
Algorithm 2. Downgrade(); (* Downgrade the power states of idle nodes*)
Input:
    N0, N1 ,..., NM: Integer; (* The current sizes of reserve pools *)
    t0, t1 ,..., tM: Integer; (* The current lengths of CTPOP of reserve pools*)
State variables:
   B0, B1, ..., BM: The sets of idle nodes in reserve pools;
Begin
  for i from 0 to M-1  do {
    if ((ti > Ti) && (Ni > Ri)) {
      select a subset DSi of Bi such that || DSi || = Ni - Ri;
      Bi = Bi - DSi ;
      Bi+1 = Bi+1 + DSi;
      Ni = Ni - || DSi ||;
      Ni+1 = Ni+1 + || DSi ||;
   }}
End
```

3.3.2 *Resouce Allocation and Reclaim.* When a new application or task is initiated and K nodes are required, the recursive resource allocation (RRA) algorithm is invoked; shown in Algorithm 3.

Assume that K is the number of nodes required by an incoming application. If the number of nodes in B_0 (i.e.the top level reserve pool) is less than K (i.e. $N_0 < K$), level 0 piercing occurs. Thus, the length of CTPOP of B_0 is reset to 0. Beside all nodes in B_0 are allocated to the

application, further $(K - N_0)$ nodes are allocated from the next level of pool (B_1). Similarly, if B_1 still cannot satisfy the requirement, level 1 pool piercing occurs, and its length of CTPOP is reset. The allocation progresses recursively until the application gets all its required nodes.

The allocation of nodes results in the decrease of amounts of resources in the corresponding reserve pools. Thus, the remaining number of nodes in the reserve pools may be less than their reserve capacity thresholds. Therefore, the upgrading algorithm described in Algorithm 1 is invoked at the end of each allocation.

```
Algorithm 3. Allocate(); (* Recursive resource allocation algorithm *)
Input:
  K: Integer; (* The number of nodes required *)
Output:
  Ba: the set of nodes allocated to the application;
State Variables:
  B0, B1, ..., BM: the sets of idle nodes in reserve pools;
Local Variables:
    n: Integer; (*the number of allocated nodes *)
    k: Integer; (* the level of target pool *)
Begin
  if (Na > Sum of Ni for i=0 to M ) {
      Report error: "require resource is more than system's capability";
      Return;
  }
  k = 0;  n = 0; Ba = EmptySet;
  while (n < K){
    if (Nk >= (K - n)) {
      Select (K- n) nodes from Bk  and add them to Ba;
      Nk= Nk - (K - n);    n = K;
    } else {
      Take all nodes in Bk into Ba;
      n = n + Nk;   Nk = 0;   tk = 0;
      k = k + 1;
    }};
Upgrade();  (* to make up the loss of resources in *)
                  (* reverse pools due to the allocation  *)
End
```

Note that, the above algorithm leaves the node selection policy issue open. Therefore, it can be combined with other optimization goals. For example, an idle node is allocated first if its temperature is lower than the others.This will help to maintain the system as cool as possible.

When an application or task terminates, all its occupied nodes are freed and become idle. These nodes will be reclaimed and placed in reserve pools. Here, we use a simple and conservative resource reclaim algorithm shown in Algorithm 4. It simply puts all reclaimed nodes into the highest reserve pool B_0. This works well with the downgrading algorithm, which in turn puts the idle node gradually to the lower level reserve pools if they are not required for a period of time.

```
Algorithm 4: Reclaim();  (*Reclaim idles nodes into reserve pools *)
Input:
   Ba: The set of nodes freed by an application;
State Variables:
    B0, B1, ..., BM: the sets of idle nodes in reserve pools;
Begin
```

```
   B0 = B0 U Ba;
   N0 = N0 + || Ba ||;
End
```

Note worthy: other reclaim policies can be easily employed in the model to select the target pools for the newly freed nodes. For example, an aggressive policy may put all reclaimed nodes into the deepest sleep state. It can also be combined with other optimization goals. For example, the idle nodes can be put into different levels of reserve pools according to their temperatures so that the hotter ones are in deeper sleeping states thus they can be cooled down.

3.3.3 *Adjustment of the Reserve Capacity Threshold.* The users' requirements on the balance between performance and energy efficiency has been represented in the state continuance thresholds for the reserve pools and dealt with by the algorithms presented in the previous subsection. This subsection is devoted to a mechanism that deals with the time-varying nature of many applications run on large scale system.

In general, a piercing of level i reserve pool means that the amount of nodes reserved in B_i is not sufficient. Therefore, its reserve capacity threshold R_i should be increased to meet the demand of workload. Here, we propose the following formula (6.a) to guide the adjustment of reserve capacity threshold when handling the piercing of level i reserve pool.

$$R_i = \begin{cases} R_i + \alpha \cdot (C_i - N_i), & C_i > N_i \quad (a) \\ Max\{R_i - \beta \cdot (N_i - C_i), 0\}, & C_i < N_i \quad (b) \end{cases} \tag{6}$$

where C_i is the number of nodes required to allocate from B_i, and α is a *performance weight factor* to reflect the user's preference for system performance.The bigger the α is, the faster the reserve capacity threshold increases. Consequently, the more idle nodes will stay in higher reserve pool, and the more weight is given to system performance.

On the other hand, there may be some residual nodes in a reserve pool after its providing nodes to the application. It means that the reserve capacity of the pool is larger than the requirement. The superfluous nodes in a pool should be put into deeper sleep state to save energy. We thus use Formula (6.b) to decrease the reserve capacity threshold, where β is an *energy weight factor*. The bigger the β is, the faster the reserve capacity threshold decreases, and the energy conservation is more preferable to the performance.

After each resource allocation, the reserve capacity threshold adjustment algorithm, shown as Algorithm 5, is called for each reserve pool. The threshold is adjusted according to the difference between the requirement and the original reserve capacity. If a pool is not covered by the allocation, its C_i is zero.

```
Algorithm 5. Adjust(); (*Adjust reserve pool capabilities *)
Input:
    Ni: Integer; (* the number nodes in Bi before a node allocation *)
    Ci:  Integer; (* the number of nodes to be allocated from Bi *)
    Ri: Integer; (* the current reserve capacity threshold of Bi *)
Output:
    R'i: Integer; (* the new reserve capacity threshold of Bi *)
Begin
  R' i= R i;
  if (Ci>Ni)  {R'i = Ri + alpha * (Ci -Ni) } ;
  if (Ci<Ni)  {R'i = Ri + beta * (Ci - Ni)} ;
  if (R'i < 0)  R'i = 0;
  Return R'i;
End
```

4. IMPLEMENTATION AND EVAULATION

We have implemented the above algorithms and conducted two simulation experiments. This section reports the main results of the experiments.

4.1 Design of the Experiments

4.1.1 *The Benchmark.* The times that a node becomes idle or busy and the numbers of nodes that are idle in the system are closely related to the workload trace on the system. Consequently, an evaluation of a power management technique must take into consideration of the workload characters.

Parallel Workload Archive [Feitelson 2011] contains dozens of workload logs on real parallel systems. Each log contains the following information on the jobs:

—*submit time*, the time moment when a job is submitted to the system;

—*wait time*, the time period when the job is waiting;

—*run time*, the time period when the job is executed;

—*number of allocated processors*, the number of processors that are allocated to the job.

From such information and the system scale, one can work out the number of busy nodes in the system at each second.

Table I summarizes the characteristics of the workload logs used in our experiment as the benchmark.

Table I: Characteristic Data of the Workload Logs Used in the Experiments

Name	System Scale	Workload Scale	No. Jobs	Average No. idle nodes	No. Turning points	Average change rate
RICC	1,024	562	1,663	7,847	1,340	2.74
PIK IPLEX	320	259	54	2,545	154	1.62
Sandia Ross	1,524	657	16	1,047	41	0.67
ANL Intrepid	40,960	38,912	631	24,897	1,116	303.47
SDSC DataStar	184	184	108	146	218	1.42
SDSC Blue Horizon	144	143	133	1,070	333	0.77
LANL CM-5	1,024	65	307	992	522	0.41
LANL Nirvana	2,048	2,048	2,158	862	1,333	23.90

In Table I, system scale is the number of nodes in the system, and workload scale is the highest workload in terms of the number of nodes used during the execution. The average number of idle nodes gives the space of energy saving.The change rate CR_t of scale at a time moment t is the number N_t of nodes minus the number N_{t-1} of nodes at the previous time moment $t-1$. It is positive, if the scale increases; and negative if the scale decreases. The average change rate is the calculated from the absolute value of the change rate over the whole period of time. A scale turning point is a time moment when the change rate changes from previously positive to either zero or negative,or from negative to either zero or positive.

4.1.2 *The Power Characteristics of the Nodes.* There is no data about the power characters of idle nodes in the ANL Intrepid log. We measured the power consumption and wakeup time of a real compute node, which contains two 6-core Xeon CPUs and 8 GB DIMMs. The results are shown in Table II. These data are used in the simulations.

The compute nodes have four different idle states S0, S1, S3 and S4. S0 is the active idle state, and S1, S3, S4 are sleep states ranking on sleep depth. The state transition energy consumptions are not considered in the experiments, because they are difficult to measure precisely and small enough to be considered as negligible for the length of continuous sleep states in our experiments,where the time granularity of our simulations is 60 seconds.

Table II: Power Characteristics of Compute Node

State	Power (Watt)	Wakeup latency (Sec.)
Busy	350	
S0	207	0
S1	171	2
S3	32	10
S4	26	190

4.1.3 *The Simulation Scenarios.* Five different scenarios in power management are simulated and compared in the experiments.Four of them using the flat reserve pool structure to simulate the existing power management solutions that do not take advantages of multiple sleep states of compute nodes.The other scenario uses our proposed hierarchical structure of reserve pools.The details of the scenarios are given below.

—*Flat reserve pool structure.* This is the trivial case when there is only one level of reserve pool. The simulation is conducted in four different sub-scenarios, where, in each scenario, the nodes in the reserve pool are at the same sleep state S0, S1, S3 and S4, respectively, whenever it becomes idle. The power states of all idle nodes are same and remain unchanged during their idle period. We will also use S0, S1, S3 and S4 to denote these scenarios, respectively. The wakeup latency is added to the wait time of a job. Hence the wakeup latency is accumulated to latter jobs if the number of idle nodes in system is less than the requirement of the incoming job. Maintaining submit time matching with the Figure 2, the workload running trace on time is influenced by wakeup latency correspondingly.

—*Hierarchical reserve pool structure.* The sleep depths of idle nodes are managed adaptively according to the ASDMIN model, where there are 4 levels of reserve pools. Each pool contains idle nodes of power state S0, S1, S3 and S4, respectively. At the time 0 of each simulation, all nodes are idle and in lowest reserve pool B_3. That is, initially, we have that $N_0 = N_1 = N_2 = 0$, and N_3 is the number of nodes in the system. The initial values of R_0, R_1 and R_2 are configured as 0. Because the B_3 is the lowest pool and the nodes in B_3 cannot be downgraded any further, the value of R_3 always equals to the number of nodes in the system.

Note worthy: first, in scenario S0, all idle nodes are active. Thus, in this scenario, the system has its highest possible response time,but no energy saving.

Second, scenario S4 is when all nodes are put into the deepest sleep state whenever it is idle. Therefore, it is the most energy efficient, but the least responsive in performance.

Finally, in our experiments we have omitted the scenario S2 that idle nodes are put into the S2 sleep state. This is because S2 state is same as S1 state except the CPU and cache context is lost. S1 is the basic state in ACPI. Few commodity CPUs and platforms support S2.

4.1.4 *Measurements and Metrics Used in The Experiments.* There are a number of metrics used to compare the energy consumption efficiency in green computing research [Liu and Zhu 2010]. Here we use the same metrics used by the Green500 list [CompuGreen 2013]. That is, energy efficiency is measured by the $MFLOPS/W$. Formally, let x be the number of instructions executed in a period of time t while consumes w energy. The *energy efficiency* (EE) of the computation is

$$EE = (\frac{x}{t})/w \tag{7}$$

Suppose that in scenario S0, the computation take t_0 time to complete with energy consumption w_0. We have that the energy efficiency EE_0 of the computation in scenario S0 equals

$$EE_0 = (\frac{x}{t_0})/w_0 \tag{8}$$

When a power management solution S_i is applied to the same computation, it takes time t_i to complete the tasks while consumes w_i energy. The energy efficiency EE_i of the power management solution S_i is

$$EE_i = (\frac{x}{t_i})/w_i \tag{9}$$

Since the data about the number x of instructions executed are not available in the workload archive, we can only calculate the energy efficiency improvement for each power management solution with regard to the situation when no power management solution is applied, i.e. the S0 scenario. Here, the *Energy Efficiency Improvement Ratio* ($EEIR$) for solution S_i is defined as follows:

$$EEIR(S_i) = \frac{EE_0}{EE_i} = \frac{((\frac{x}{t_0})/w_0)}{((\frac{x}{t_i})/w_i)} = (\frac{t_i}{t_0})/(\frac{w_i}{w_0}) \tag{10}$$

In a large scale system, at each time moment, there are many computation tasks executed in parallel. Let P be a given period of time in the execution of a system. Let C_i be the set of tasks executed in the period P of time with power management solution S_i, and for each a in C_i, t_i^a and x_i^a be the time to complete the task and the number of instructions executed for task a,we have that

$$EEIR(S_i) = \frac{\left(\frac{\sum_{a\in C_o} x_0^a}{\sum_{a\in C_0} t_0^a}\right)/w_0^{C_0}}{\left(\frac{\sum_{a\in C_i} x_i^a}{\sum_{a\in C_i} t_i^a}\right)/w_i^{C_i}} = \left(\frac{\sum_{a\in C_i} t_i^a}{\sum_{a\in C_0} t_0^a}\right) \times \left(\frac{w_i^{C_i}}{w_0^{C_0}}\right) \times \left(\frac{\sum_{a\in C_o} x_0^a}{\sum_{a\in C_i} x_i^a}\right) \tag{11}$$

When the period of time is significantly longer than the lengths of tasks, we have that $C_o \approx C_i$. Thus, we have that

$$\frac{\sum_{a\in C_o} x_0^a}{\sum_{a\in C_i} x_i^a} \approx 1 \tag{12}$$

Therefore, we can use the following as a measure of EEIR.

$$EEIR(S_i) \approx \left(\frac{\sum_{a\in C_i} t_i^a}{\sum_{a\in C_0} t_0^a}\right) \times \left(\frac{w_i^{C_i}}{w_0^{C_0}}\right) = SD(S_i) \times ES(S_i) \tag{13}$$

where $SD(S_i) = \frac{\sum_{a\in C_i} t_i^a}{\sum_{a\in C_0} t_0^a}$ is the overall slowdown rate over a given period of time and $ES(S_i) = \frac{w_i^{C_i}}{w_0^{C_0}}$ is the overall energy saving rate over a given period of time.

In a system with power management on idle nodes,the time to complete a job, denoted as execution time, i.e. its end time minuses its submit time, consists of three parts as in formula (14): the *wait time* and the *run time* as logged in the workload archive when no dynamic sleep mechanism is not used, and the *wakeup delay* is the additional delay caused by awakening idle nodes that are allocated to the job.

$$execution\ time = wait\ time + wakeup\ delay + run\ time \tag{14}$$

4.2 Experiment 1: The Impact of Configuration Parameters

The first experiment was conducted to study the influence of configuration parameters on the effectiveness of the proposed approach. Due to the labor-intensive nature of this experiment, only one of the workload logs was used in this experiment.

4.2.1 *The Benchmark.* The ANL Intrepid log in the archive is selected as the workload trace in this experiment. The ANL Intrepid comprises 40,960 quad-core nodes, which is the largest system scale among all logs in the archive [Feitelson 2011]. Our simulations start at the time of 0 of the log. However, to avoid the fulfilling effect of the system starting,the data of the first

month (i.e. 30 days) are neglected, and the workload on the following 48 hours, shown in Figure 2, is investigated as the input of our simulations.

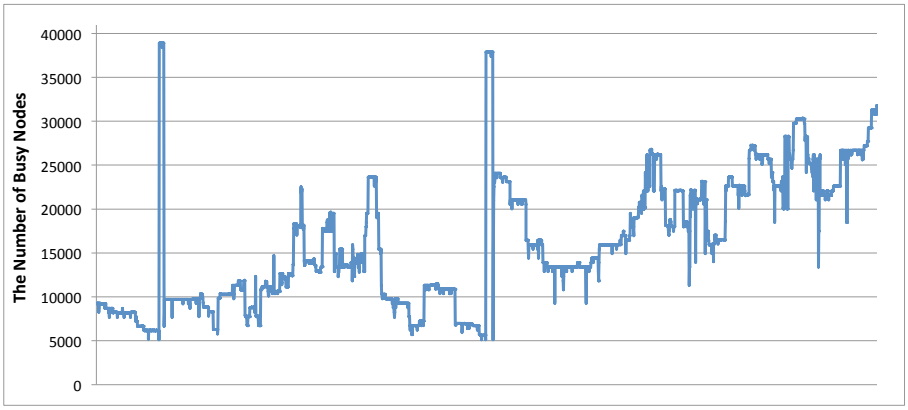

Figure 2. ANL Intrepid Workload Distribution

In Figure 2, the grey area exhibits the number of busy nodes in the ANL Intrepid system along with the time. The top line is the system scale, i.e. 40,960 nodes. The white area is the number of idle nodes at run time. On average, 60.78% of nodes in the system are idle.

4.2.2 *The Impact of the Size of Downgrading Subset.* According to the downgrading algorithm given in the previous section, a subset DS_i of the nodes in the reserve pool B_i is selected as the target nodes to be downgraded to the deeper sleep state. Constrained by the reserve capacity threshold, the maximal size of DS_i that can be downgraded from B_i is $(N_i - R_i)$. We employ $\delta_i \cdot (N_i - R_i)$ as the size of DS_i in the simulation, where δ_i is a fractional constant, i.e. $0 \leq \delta \leq 1$. In each simulation, the δ_i's are invariant. Multiple simulations are executed with different values of δ_i's. The results of the simulations are shown in Figure 3.

Because the execution time in ANL Intrepid workload is much greater than the wakeup delay and the management on the power state of idle node only affects the wakeup delay,the job execution time varies trivially in different scenarios. Generally speaking, the energy consumed by all nodes varies with δ (i.e.the size of the DS), which forms a bathtub curve, while the job execution time only varies slightly. In the result, the energy efficiency also forms a bathtub curve with δ. When δ is 0.4, the energy efficiency is the best. Therefore, in the further experiments we used $0.4 \cdot (N_i - R_i)$ as the optimal configuration of downgrading set size ratio.

4.2.3 *The Impact of Adjustment Speed.* The speed of adjustment of the reserve capacity thresholds can be tuned by setting two coefficients: the performance weight factor α and the energy weight factor β. They are employed to reflect the user's preferences in system performance and energy conservation. In our simulation experiments, we give the equal weight to performance and energy consumption. Thus, we configure the performance weight factor to be equal to the energy weight factor, i.e. $\alpha_i = \beta_i$. Moreover, for the sake of simplicity, all reserve pools hold the same configuration values of these two factors. That is,for all B_i and B_j of reverse pools, we have that $\alpha_i = \alpha_j$ and $\beta_i = \beta_j$. Simulation experiments were carried out to investigate how the value of α and β affect performance and energy consumption. The results are shown in Figure 4. In general, the system energy, job execution time and energy efficiency all varies with the increase of adjustment weight. When α_i and β_i are 0.15, the energy efficiency is the best.

4.2.4 *The Impact of State Continuance Threshold.* Another parameter that affects the effectiveness of power management is the state continuance threshold T_i used in the Algorithm 2. It

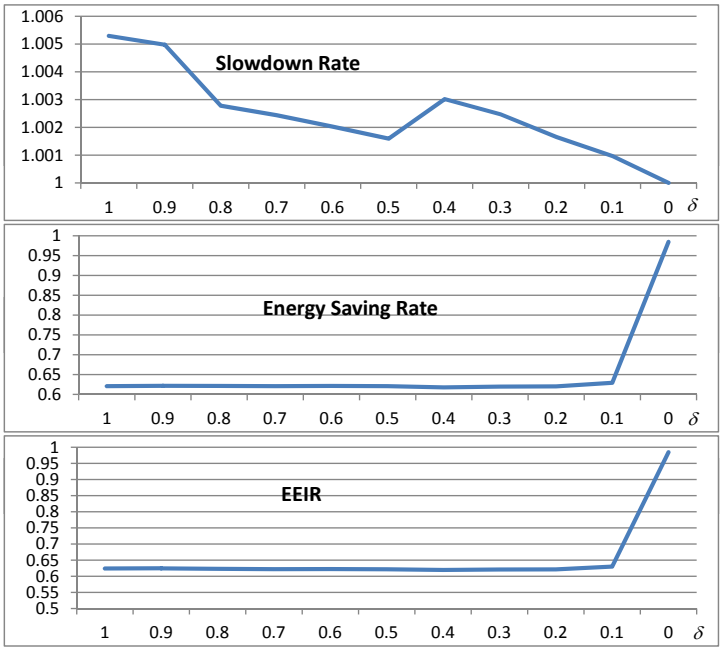

Figure 3. Average management effect with different size of DS.

controls the length of time when idle nodes will stay in a reserve pool before downgrading. Its effect to power management is shown as Figure 5.

Both the energy consumed by all nodes and the overall execution time varies slightly with the threshold. However, the energy efficiency is the best when T_i is 7.

Based on the results of the above simulation experiments, we configured the ASDMIN in such way that $\delta_i = 0.4$, $\alpha_i = \beta_i = 0.15$, $T_i = 7$. This configuration is used in the further experiments that compare ASDMIN with existing power management solutions.

4.3 Experiment 2: Comparison with Existing Power Management Solutions

The simulations are executed on the benchmark in 5 different scenarios discussed in subsection 4.1.3. Figure 6 shows the power consumptions by the idle nodes in 5 different scenarios in the experiment using the ANL Intrepid workload.It clearly demonstrated that the power consumption in the ASDMIN scenario is almost the same as that of the S4 scenario, which is the scenario in which energy consumption is the lowest.

On the other hand, as shown in Figure 7, where the y-axis is the job execution time, in the ASDMIN scenario, the execution time is almost the same as in scenario S0, where all nodes are kept active idle and the system is in its highest performance.

Figure 6 and 7 shows that, on the ANL Intrepid workload benchmark, the ASDMIN model achieved energy efficiency with a close match to power management solutions represented in scenario S4 and at the same time achieved a performance with a close match to solutions represented in scenario S0. The same phenomena are observed in the simulation experiments with all of the workload logs,as shown in Figure 8 below.

In Figure 8, BUSY is the number of busy nodes in the system, N0,..., N3 are the number of nodes in reserve pool B0,...,B3, respectively. The distributions of idle nodes during the experiment are summarized in Table III.

In Table III, column N_i $(i = 0, 1, ..., 3)$ are the average numbers of idle nodes in reserve pool B_i, and BUSY are the average number of busy nodes. On average, there are 94.98% of idle nodes in the lowest pool (N_3). In other words, most idle nodes are in the deepest sleep state in most of

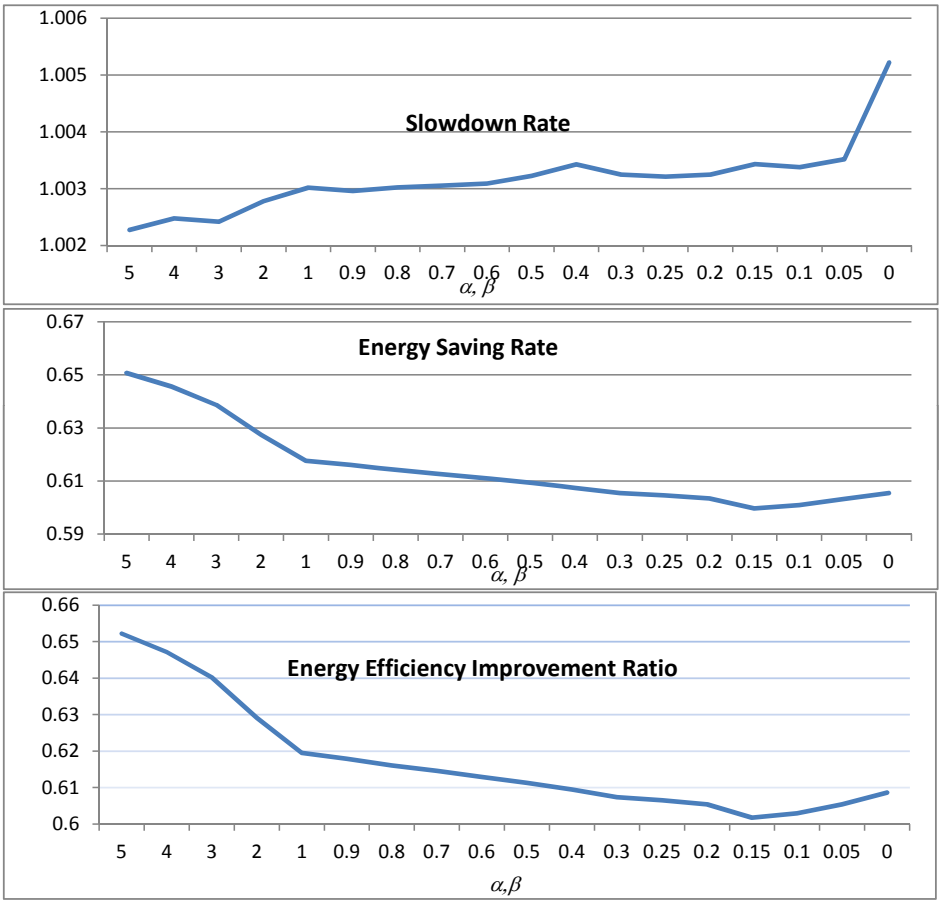

Figure 4. Average management effect with different weight

Table III: Distributions of Idle Nodes over Different Reserve Pools

Workload log	BUSY	N0	N1	N2	N3
RICC	319.92	7.67	4.28	2.09	690.05
PIK IPLEX	16.27	3.11	2.10	0.09	298.43
Sandia Ross	477.01	6.06	2.35	0.01	1038.56
ANL Intrepid	16,383	679.47	463.94	152.01	23,281.58
SDSC DataStar	39.31	2.40	1.69	0.13	140.48
SDSC Blue Horizon	82.66	1.73	1.71	0.02	57.89
LANL CM-5	32.15	0.94	0.98	0.002	989.93
LANL Nirvana	1211.17	65.76	27.82	12.74	730.50
Overall	18,561	767	505	167	27,227

time.

Our proposed model does not only saving energy, but also achieves high performance. The main results of the experiments are summarized in Table IV.

The effects of idle node management in the five different management solutions on the 8 systems are shown in Figure 9.

In comparison with scenario S0, in which all idle nodes are in active state, on average of 8 workload logs, ASDMIN reduces overall system energy consumption by 50.93% at the cost of increasing the average job execution time by 3.49%. In this scenario, thus, by applying Formula

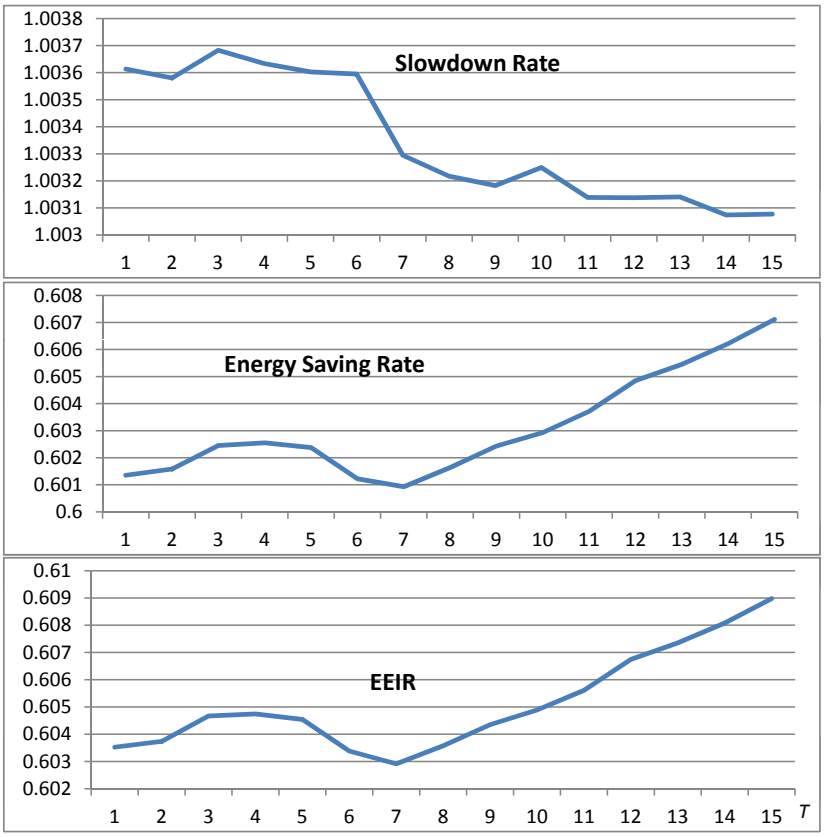

Figure 5.Average management effect with different state continuance threshold

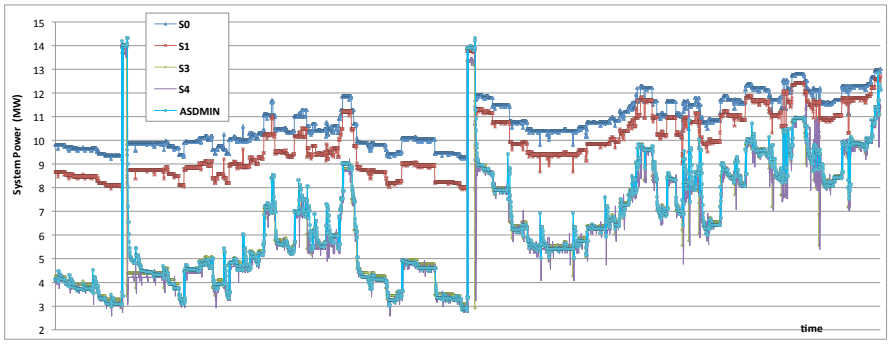

Figure 6. The variation of power consumption by all nodes with the time of executions on the benchmark

(13), we have that the energy efficiency is improved by 49.32%.

In scenario S4 all idle nodes are always put into the deepest sleep state. Thus is consumes the least energy, i.e. 47.62% of S0. However, its job execution time is much higher, 111.25% of that in S0 scenario.In comparison with the S4 scenario, ASDMIN improves the energy efficiency by 4.21%.

5. CONCLUSION

To save the huge amount of energy wasted by active idle nodes in large scale systems, this paper proposes a self-adaptive solution to manage the sleep depths of idle nodes to balance between

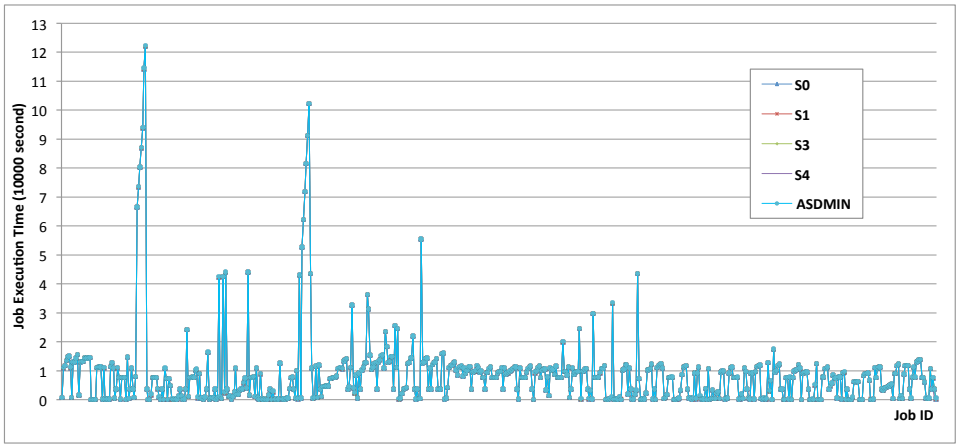

Figure 7. The variation of job execution time of different jobs on the benchmark

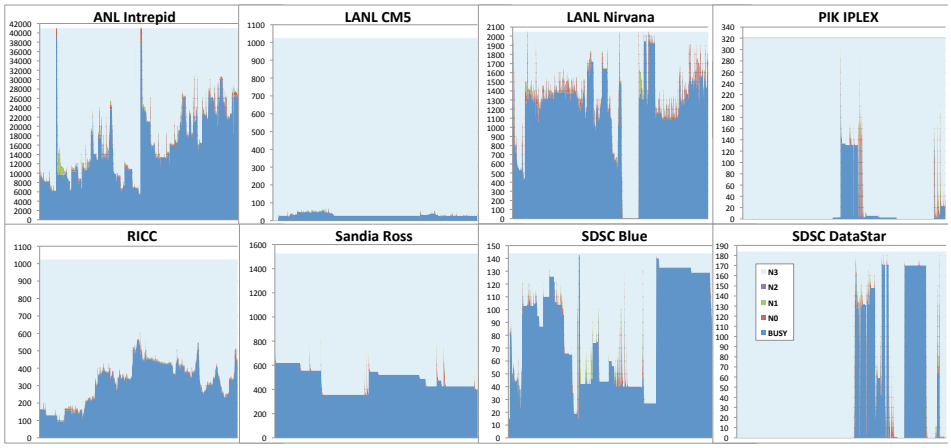

Figure 8. The variation of the numbers of nodes in the reserve pools in the ASDMIN scenario

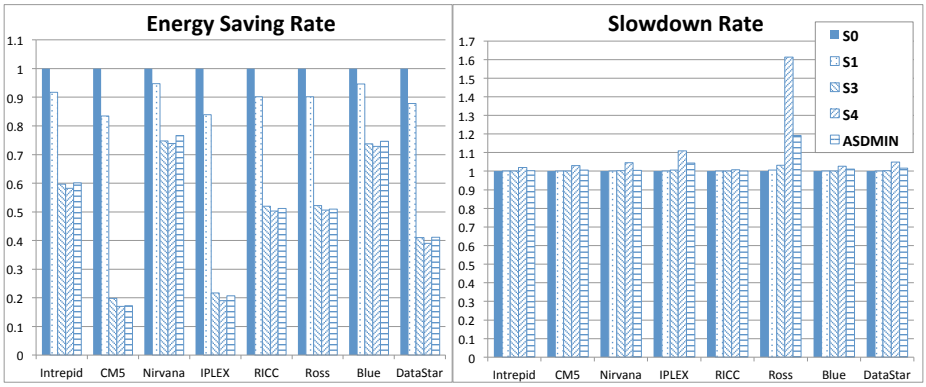

Figure 9. The results of different management scenarios.

energy consumption and system response speed. Idle nodes in the same sleep depth form a reserve pool and the system consists of a hierarchy of reserve pools with different sleep depths. The nodes in a pool of lower sleep depth are allocated with higher priority because they have shorter wakeup

Table IV: Main Results of the Comparison Experiment

(a) System Energy Consumption

Workload Log	S0	S1	S3	S4	ASDMIN
RICC	741,220,891	667,997,705	385,406,394	372,934,620	379,286,204
PIK IPLEX	196,723,003	165,050,561	42,784,236	37,483,092	40,752,095
Sandia Ross	1,104,659,470	996,021,910	576,701,412	558,803,232	563,224,823
ANL Intrepid	31,041,595,008	28,464,589,696	18,528,119,808	18,077,460,480	18,653,846,315
SDSC DataStar	125,108,554	109,912,162	51,335,784	48,804,912	51,409,516
SDSC Blue Horizon	119,470,344	113,008,612	88,147,638	87,085,692	89,116,071
LANL CM-5	623,507,152	520,625,394	123,415,416	106,243,416	107,497,740
LANL Nirvana	1,709,196,453	1,620,065,725	1,277,229,696	1,262,147,364	1,310,535,398

(b) Overall Job Execution Time

Workload Log	S0	S1	S3	S4	ASDMIN
RICC	42,496,355	42,499,681	42,512,985	42,812,325	42,539,093
PIK IPLEX	93,990	94,098	94,530	104,250	98,176
Sandia Ross	4,956	4,988	5,116	7,996	5,908
ANL Intrepid	5,535,305	5,536,567	5,541,615	5,644,314	5,553,541
SDSC DataStar	423,556	423,772	424,636	444,076	430,782
SDSC Blue Horizon	951,832	952,098	953,162	977,102	962,500
LANL CM-5	1,925,360	1,925,974	1,928,430	1,983,690	1,937,316
LANL Nirvana	8,831,008	8,835,324	8,852,588	9,232,118	8,865,616

(c) Energy Efficiency Improvement Ratio

Workload Log	S0	S1	S3	S4	ASDMIN
RICC	1	0.901,283	0.520,165	0.506,877	0.512,219
PIK IPLEX	1	0.839,964	0.218,734	0.211,337	0.216,381
Sandia Ross	1	0.907,477	0.538,917	0.816,154	0.607,803
ANL Intrepid	1	0.917,191	0.597,561	0.593,831	0.602,910
SDSC DataStar	1	0.878,982	0.411,376	0.409,000	0.417,930
SDSC Blue Horizon	1	0.946,178	0.738,851	0.748,284	0.754,287
LANL CM-5	1	0.835,261	0.198,253	0.175,559	0.173,479
LANL Nirvana	1	0.948,316	0.749,095	0.771,986	0.769,760

latency. The state of an idle node is dynamically upgraded to a lower sleep depth pool to be ready for wakeup with a shorter latency, or downgraded to a deeper sleep depth in order to save energy. Corresponding resource allocation and reclaim algorithms, dynamic state transition algorithms are designed to maintain the proper distribution of idle nodes among different pools. It is recognized that for such a power management solution to be efficient in energy consumption and having little impact on system performance, the sizes of the reserve pools must match well with the characteristics of systems workload, which is highly time varying and hard to predict accurately. To address this problem, a self-adaptive mechanism is employed to adjust reserve pool capabilities based on the notion of lengths of continuous time period without piercing (CTPOP), where a piercing of a reserve pool occurs when it is insufficient to provide required amount of resources on demand. When the length of CTPOP is small than a set threshold (which is called the state continuance thresholds), adjustment of the reserve pool capacity happens. Because piercing means longer wakeup latency, state continuance threshold determines the degree that the system will be tolerant of performance slowdown. This parameter can be set by the user. Thus, it represents the users preferences between performance and energy efficiency. It determines how fast the downgrading of sleep states of the idle nodes should be made and how often the adjustment of the capabilities of reserve pools should be performed.

In comparing with existing solutions of power management for large scale systems, which do not utilize the multiple sleep state mechanisms, our simulation experiments demonstrated that our solution can upgrade the energy efficiency by 49.32% on average. The proposed self-adaptive sleep depth management for idle nodes is an effective approach to optimize the system energy efficiency.

For future work, we are exploring the combination of various policies in the selection of idle node for downgrading and upgrading their sleeping states to achieve other system management goal, such as to select nodes according to their temperature in order to save cooling power consumption.

REFERENCES

Chase, J., Aderson, D., Thakar, P., and et al. 2001. Managing energy and server resources in hosting centers. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*. Banff, Canada, 103–116.

CompuGreen. 2013. Green 500. URL: http://www.green500.org/.

Cook, G. 2012. How clean is your cloud? Greenpeace International.

Feitelson, D. 2011. Parallel workloads archive. http://www.cs.huji.ac.il/labs/ parallel/workload/l-anl-int/ANL-Intrepid-2009-1.swf.gz.

Feng, W. 2003. Making a case for efficient supercomputing. *ACM Queue 1,* 7, 54–64.

Gandhi, A., Harchol-Balter, M., and Kozuch, M. A. 2011. The case for sleep states in servers. In *Proc. of the HotPower '11*. Cascais, Portugal.

Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, and et. al. 2011. Advanced configuration and power interface specification revision 5.0. URL: http://www.acpi.info/spec50.htm.

Horvath, T. and Skadron, K. 2008. Multi-mode energy management for multi-tier server clusters. In *Proc. of the 17th International Conference on Parallel Architecture and Compilation Techniques,*. Toronto, Canada, 270–279.

Krioukov, A., Mohan, P., Alspaugh, S., and et al. 2011. Napsac: Design and implementation of a power-proportional web cluster. *ACM SIGCOMM Computer Communication Review 41,* 1, 102–108.

Liu, Y. and Zhu, H. 2010. A survey of the research on power management techniques for high performance systems. *Software Practice and Experience 40,* 1, 943–964.

Meuer, H., Strohmaier, E., Dongarra, J., and Simon, H. 2012. Top 500. http://www.top500.org.

Mustafa, R. M., Nishkam, R., Srihari, C., and et al. 2011. Power management for heterogeneous clusters: an experimental study. In *Proc. of the 2nd International Green Computing Conference (IGCC'11)*. Orlando, USA.

Pinheiro, E., Bianchini, R., Carrera, E., and et al. 2001. Load balancing and unbalancing for power and performance in cluster-based systems. Tech. Rep. Technical Report DCS–TR–440,, Department of Computer Science, Rutgers University. May.

Srikantaiah, S., Kansal, A., and Zhao, F. 2008. Energy aware consolidation for cloud computing. In *Proc. of the 2008 Workshop on Power Aware Computing and Systems (HotPower'08)*. San Diego, USA.

U.S. Environmental Protection Agency. 2007. Report to congress on server and data center energy efficiency. http://www.energystar.gov/ia/partners/prod-development/downloads/EPA-Datacenter-Report-Congress-Final1.pdf.

Xue, Z., Dong, X., Ma, S., and et al. 2007. An energy-efficient management mechanism for large-scale server clusters. In *Proc. of the 2007 IEEE Asia-Pacific Services Computing Conference.* 509–516.

Yongpeng Liu received his BS, MS, and Ph.D degree in the School of Computer Science from National University of Defense Technology, China, in 2000, 2002, and 2013, respectively. He is now a assistant professor in the School of Computer Science at National University of Defense Technology. His research interests include power management, fault tolerance and high performance computing.

Hong Zhu obtained his BSc, MSc, PhD degrees in Computer Science from Nanjing University, China, in 1982, 1984 and 1987, respectively. He was with Nanjing University from August 1987 to November 1998 as a lecturer, associate professor and then full professor and supervisor of PhD students. From October 1990 to December 1994, while on leave from Nanjing University, he was a research fellow with Brunel University and the Open University, UK. He joined Department of Computing of Oxford Brookes University in November 1998 as senior lecturer in computing and became a professor of computer science in October 2004, where he chairs the Applied Formal Methods Research Group. He is a senior member of IEEE Computer Society, a member of British Computer Society, ACM, China Computer Federation, and China Artificial Intelligence Association. His research interests are in the area of cloud computing and software engineering including software testing, software development methodology, agent technology, automated software development tools, etc.

Kai Lu received his BS and Ph.D degree in the School of Computer Science at National University of Defense Technology, China, in 1995 and 1999, respectively. He is now a professor in the School of Computer Science at National University of Defense Technology. His research interests include computer architecture, operating system, parallel and distributed computing.

Xiaoping Wang received his BS, MS, and Ph.D degree in the School of Computer Science from National University of Defense Technology, China, in 2003, 2005, and 2010, respectively. He is now an assistant professor in the School of Computer Science at National University of Defense Technology. His research interests include parallel and distributed computing.