Self-Adaptive Management of The Sleep Depths of Idle Nodes in Large Scale Systems to Balance Between Energy Consumption and Response Times

Yongpeng Liu[(1)], Hong Zhu[(2)], Kai Lu[(1)], Xiaoping Wang[(1)]
[(1)] School of Computer Science, National University of Defense Technology, Changsha, P. R. China
[(2)] Department of Computing and Communication Technologies, Oxford Brookes University, Oxford, U.K

Abstract —**Due to the time-varying nature of real workload, a large scale computer system has quite a number of idle nodes in most time of operation. They consume energy, but do nothing useful. To save the huge energy waste caused by such active idle nodes, most modern compute nodes provide multiple level dynamic sleep mechanisms to reduce power consumption. However, awaking sleeping nodes takes time, thus affects the response times and performance of the system. A node is deeper in sleep, it consumes less energy, but has longer wakeup latency. This paper proposes a sleep state management model to balance the system's energy consumption and response times. In this model, idle nodes are classified into different 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. In a resource allocation process, nodes in the pool of highest level of readiness are preferentially provided to the application. 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 at all times. Obviously, a key factor that affects the effectiveness of the idle node management is the sizes of the reserve pools. This paper proposes and investigates a self-adaptive approach to this problem so that the sizes of reserve pools are dynamically adjusted according to the applications. Our experiments demonstrated that, by applying our self-adaptive management, the power consumption of idle nodes can be reduced by 84.12% with the cost of slowdown rate being only 8.85%.**

Keywords – Large scale systems; Power Management; Dynamic Sleep; Idle node; Performance; Response Times

I. 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. 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 [1], the average power consumption of Top10 systems is 4.34 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 U.S. Dollars. This is about 1.5% of the total U.S. electricity consumption or the output of about 15 typical power plants [2]. 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 [3]. 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 the reliability and availability of the system. Huge construction costs of large scale systems are also incurred in order to accommodate the huge amount of energy demand.

On the other hand, workload of data centers varies significantly with time and the average resource utilization of large scale systems typically sit at low levels of utilization. A quite number of nodes are idle in most time. Unfortunately, nowadays nodes are not power-proportional and a node in idle state is highly energy inefficient. The power consumption of an idle node generally reaches about 50% of its peak power [4]. In the result, idle nodes in large scale systems cause huge energy waste.

To reduce the power consumption of a node in its idle state, dynamic sleep mechanism is proposed and multiple sleep states are supported in current common nodes [5]. Each sleep state consumes less power than idling in the active state. The deeper the 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 management solution of multiple sleep state of idle nodes in large scale systems to make an effective tradeoff between energy conservation and system response performance.

II. RELATED WORKS

Dynamic speed scaling and dynamic resource sleep are two power management mechanisms widely supported in current information industry. Even all components were scaled into their lowest speeds, the active power consumption of an idle node is significantly higher than its sleep power [5]. The dynamic cluster configuration, i.e. put idle nodes into sleep states and wake them on demand, is a typical power management technique for large scale systems [4-8]. To balance between energy and performance, most researchers on dynamic configuration of cluster focus on server consolidation via finding an appropriate active portion of the cluster dynamically. The idle remainders are simply turned off [10].

The importance of supporting multiple sleep state for servers in data centers has been investigated by Gandhi *et al.* [9]. However, their approach does not dynamically manage

the sleep depth of idle servers. Horvath *et al.* [10] propose an energy management policy which exploits the multiple sleep states of idle servers. They predicate the incoming workload based on history resource utilization change and select the optimal number of spare servers for each power states in an ad-hoc manner. Extra spare servers are put in the deepest possible sleep states. Different to their heuristic selection of sleep state, we control the state transition of idle nodes in a self-adaptive model where reserve pools of idle nodes with corresponding sleep depths regulate their sizes respectively. Xue *et al.* [11] provide the active resource pool with dynamic computing capacity in accordance with the time-varying workload demand. However, spare nodes in their power management solution are simply turned off. Multiple sleep sates are not considered by them. In this paper, we explore the benefits of these multiple sleep states mechanism to improve the energy efficiency of large scale systems with minimal sacrifices of system performance.

III. SELF-ADAPTIVE MANAGEMENT MODEL

The nodes in a cluster environment can be classified into two categories according to whether any application is 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 generate any useful compute production.

To avoid the energy waste on idle state, an idle node should be put into low-power sleep state. In current information technology, a node may support multiple sleep states. For example, in ACPI specification, the power state of a node can be S0, S1, S2, S3, S4 or S5, where S0 is the active state, S1~S5 denote different levels sleep state with S5 as the deepest sleep state. The deeper a node sleeps, the less power it consumes, however, 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, …, M, respectively, where M the number of supported sleep states of the node. These parameters satisfy the follow formula 1.

$$\forall i,j\cdot(0\le i<j\le M\Rightarrow(P_i>P_j)\wedge(D_i<D_j)\wedge(E_i<E_j)) \quad (1)$$

The power consumption in a sleep state is always lower than the power usage in active standby state. However, additional state transition energy is required to put a node into sleep state or wake it up. Thus, to conserve energy by dynamic sleep mechanism, it is necessary that the continuance time width of sleep state should be long enough. So that the energy saved during sleep is greater than additional energy consumption caused by state transition. On the other hand, a node in sleep state is not available. Before providing any functional service, the node must first be woken up. It means that the wake up latency may depress the response speed of the sleep node. Consequently, it may be not the best choice to put a node into its deepest sleep state whenever it becomes idle. An effective power management solution is required to schedule the appropriate sleep or wake timing of idle nodes to improve the energy efficiency of the whole system. From the cluster-wide viewpoint, node sleep depth management means to distribute the idle nodes among different groups of corresponding power states.

In the paper, we propose a self-adaptive management model for the sleep depth of idle nodes in large scale systems, called ASDMIN. 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.

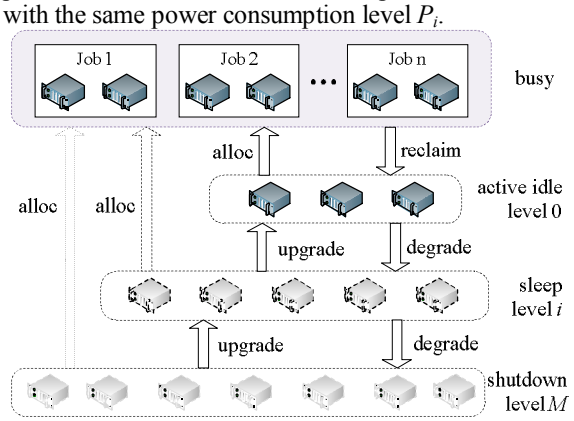

Figure 1. ASDMIN model for adaptive sleep depth management of idle nodes

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 can get that:

$$N=\sum_{i=0}^{M}N_i \qquad (N_i\ge 0) \quad (2)$$

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

$$P_{idle}=\sum_{i=0}^{M}P_iN_i \quad (3)$$

ASDMIN try to reduce the power consumption of idle nodes by exploiting dynamic sleep mechanism. However, in a deeper sleep state means a longer wakeup latency. To provide sufficient nodes to the incoming application as soon as possible, ASDMIN resource allocation policy will preferentially allocate nodes from the highest reserve pool 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 the required total node number of an incoming application is A, the lowest of the levels of reserve pools covered by its allocation is denoted by $Level(A)$. Obviously, we have that

$$Level(A)=l\Leftrightarrow\left(\sum_{i=0}^{l-1}N_i<A\le\sum_{i=0}^{l}N_i\right) \quad (4)$$

Assume that allocated nodes are waked up in parallel. Then, the response time D of an allocation is determined by the largest wakeup latency of the allocated nodes. Let $Delay(A)$ denote the wakeup latency of allocating A nodes. According to (1), we have that

$$Delay(A)=\max\{D_0,D_1,...,D_l\}=D_l \quad (5)$$

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, there should be enough nodes in the pools whose sleep depth are lower than l. Therefore, we set a *reserve capacity threshold*, denoted as R_i, to control the minimum number of nodes in pool B_i. In other words, the number of nodes in B_i should always be no less than R_i. Whenever the node number of B_i is less than R_i, nodes in the lower pools will be upgraded to fill the reserve capacity.

To save energy consumption of idle nodes, ASDMID tries to degrade idle nodes to a sleep state as deep as possible while maintains enough reserves.

Generally speaking, the less the reserve capacity thresholds for the highest reserve pools are, the more likely a node is degraded, and the less total power consumption of all idle nodes, and then the more energy is saved. However, the wakeup latency is longer and the performance loss caused by dynamic sleep is more significant.

When the nodes allocated to an application are freed, they will be reclaimed and delivered to the appropriate reserve pools.

By dynamically scheduling the sleep depths of idle nodes, ASDMIN can make tradeoff between energy and performance. In the optimal scenario, the system's response speed matches nearly to the speed when all nodes are put in the active standby state (i.e., in the highest reserve pool), and the energy conservation nearly match to that when all nodes are in deepest sleep state (i.e. in the lowest reserve pool). However, in reality, a trade-off between energy consumption and performance must be made. Our approach to this trade-off is to adjust the reserve capability thresholds dynamically and adaptively. The related management algorithms in ASDMIN will be introduced in detail in the next section.

IV. Power Management Algorithms

The operation of ASDMIN model depends on two sets of algorithms: the degradation and upgradation algorithms for changing the sleep depths of the idle nodes, and allocation and reclaim of nodes for deciding which node is to allocated for an application.

A. Degrading and Upgrading Idle nodes

Once an idle node is allocated to an application, the amount of reserve in the 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.

Algorithm 1. Upgrade();
(* Upgrade the power states of idle nodes *)
State Variables:
$<B_0, B_1 \ldots\ldots B_M>$: Sets of idle nodes in the reserve pools;
Local Variable:
k: the level of the target pool to recruit;
Begin
for all pools B_i
{
$k = i + 1$;
while ($N_i < R_i$) {
if ($k > M$)
break;
if ($N_k >= (R_i - N_i)$) {
Select ($R_i - N_i$) nodes from B_k into B_i;
$N_i = R_i$;
$N_k = N_k - (R_i - N_i)$;
} else {
Take all nodes in B_k into B_i;
$N_i = N_i + N_k$;
$N_k = 0$;
$k = k+1$;
}
}
}
End

When a node is reclaimed due to the finish of a task, it should be 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 consumed energy, too. A question is that how long the delay should be so that 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 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 a 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 from the last piercing of the reserve pool.

The length of a CTPOP gives a good indication of how sufficient the capability of the 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 large. In this scenario, the power consumption is unnecessary and some of the nodes can be put in a deeper state of sleeping. 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*, i.e.

- A *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 CTPOP t_i of 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 degraded into deeper sleep state to save energy. In such situation, a subset of B_i, notated as DS_i, is selected as the target to degrade its nodes into the lower pool B_{i+1}.

Therefore, the degradation of a node follows two constraints, i.e. (a) the size of the reserve pool is greater than the reserve capacity (R_i) and (b) the length of CTPOP is greater that state continuance threshold (T_i). Details of the algorithm are given below.

Algorithm 2. Degrade();
(* Degrade the power states of idle nodes*)
Input:
$<N_0, N_1 \ldots\ldots N_M>$: The current sizes of reserve pools;
$<t_0, t_1 \ldots\ldots t_M>$: The current lengths of the CTPOP of reserve pools;
State variables:
$B_0, B_1 \ldots\ldots B_M$: the sets of idle nodes in reserve pools;
Begin
for i from 0 to M–1 do {
if (($t_i > T_i$) && ($N_i > R_i$)) {
select a subset DS_i of B_i such that $||DS_i||=N_i-R_i$;
$B_i = B_i - DS_i$;
$B_{i+1} = B_{i+1} + DS_i$;
$N_i = N_i - ||DS_i||$;
$N_{i+1} = N_{i+1} + ||DS_i||$;
}
}
End

B. Resouce Allocation and Reclaim

When a new application or task is initiated and K nodes are required, a recursive resource allocation (RRA) algorithm is invoked as shown in Algorithm 3.

Assume that the required node number by the incoming application is K. 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 CTPOP of B_0 is reset to 0. Beside all nodes in B_0 are allocated to the application, RRA will further allocate (K–N_0) nodes 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 resent. The allocation progresses recursively until the application gets all its required nodes.

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

Algorithm 3. Allocate()
(* Recursive resource allocation algorithm *)
Input:
K: The number of nodes required by the incoming application;
Output:
B_a: the set of nodes allocated to the application;
State Variables:
$<B_0, B_1 \ldots\ldots B_M>$: the sets of idle nodes in reserve pools;
Local Variables:
n: the number of allocated nodes;
k: target pool;
Begin
if ($N_a > \sum_{i=0}^{M} N_i$) {
report error: "require resource is more than system's capability";
return;
}
$k = 0$;
$n = 0$;
$B_a = \emptyset$;
while ($n < K$){
if ($N_k >= (K - n)$) {
Select (K- n) nodes from B_k and add them to B_a;
$N_k = N_k - (K - n)$;
$n = K$;
} else {
Take all nodes in B_k into B_a;
$n = n + N_k$;
$N_k = 0$;
$t_k = 0$;
$k = k + 1$;
}
};
/* upgrade algorithms is invoked to make up the loss of the allocation */
Upgrade();
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.

At the end of an application or task, all its occupied nodes are freed and become idle. These nodes will be reclaimed and delivered into reserve pools. Here, we use a simple and conservative resource reclaim algorithm, seen as Algorithm 4. It simply put all reclaimed nodes into the highest reserve pool (B_0). This works with the degradation algorithm to put the idle node gradually to the lower level reserve pools is they are not required for a period of time.

***Algorithm* 4**: Reclaim();
(*Reclaim idles nodes into reserve pools *)
Input:
B_a: The set of nodes freed by an application;
State Variables:
$<B_0, B_1 \ldots\ldots B_M>$: the sets of idle nodes in reserve pools;

Begin
$B_0 = B_0 \cup B_a$;
$N_0 = N_0 + \|B_a\|$;
End

Note worthy: other reclaim policies are easy to be 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.

C. Adaptive 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 section. This section 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 dynamically to meet the time-varying of the workload. Here, we propose the following formula (11.a) to guide the adaptive adjustment of reserve capacity threshold when handling the piercing of level *i* reserve pool.

$$R_i = \begin{cases} R_i + (C_i - N_i) & C_i > N_i \quad \text{(a)} \\ \max\{R_i - (N_i - C_i), 0\} & C_i < N_i \quad \text{(b)} \end{cases} \qquad (11)$$

where C_i is the number of nodes required to allocate from B_i.

On the other hand, there may be some residual nodes in a reserve pool after its providing enough 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 (11.b) to decrease the reserve capacity threshold adaptively.

At the end of each resource allocation, the reserve capacity threshold adjustment algorithm, shown as Algorithm 5, is called for each reserve pool. The threshold is adjusted adaptively 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*:**
N_i: the number nodes in B_i before a node allocation;
C_i: the number of nodes to be allocated from B_i;
R_i: the reserve capacity threshold of B_i;
Output:
R'_i: the new reserve capacity threshold of B_i;
Begin
$R'_i = R_i + (C_i - N_i)$;
If $(R'_i < 0)$ $R'_i = 0$;
return R'_i;
End

V. Implemention and Evaluation

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

A. 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 [12] publishes dozens of workload logs on real parallel systems. Each log contains the following information on the jobs: submit time, wait time, run time and number of allocated processors. From the information and the system scale, one can work out the number of nodes in the system at each second.

The ANL Intrepid log is selected as the workload trace in our simulations. The ANL Intrepid comprises 40,960 quad-core nodes, which is the maximal system scale among all published logs in [12]. 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 24 hours are neglected, and the workload on the following 48 hours is investigated as the input of our simulations. There are quite a number of idle nodes in about 94.79% of the simulation time.

B. The Power Characteristics of the Nodes

There is no data about the power characters of idle nodes in ANL Intrepid. We measured the power consumption and wakeup time of a typical compute node, the Tianhe-1A compute node, with two 6-core Xeon CPUs and 8 DIMMs. The results are shown in Table 1. These data are used in the simulations. There are four different idle states supported by the node, S0, S1, S3 and S4. S0 is the active idle state, and S1, S3, S4 are sleep states ranking on sleep depth. The transition overhead between sleep states are not considered in this paper, because transition energy consumptions are difficult to measure precisely. The time granularity of our simulations is 60 seconds.

Table 1. Multiple sleep states of a typical node

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

C. The Simulation Scenarios

Five scenarios are simulated with different power management solution for idle nodes.

- *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 use S0, S1, S3 and S4 to denote these scenarios, too. 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 original workload, the 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=40,960. 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 are not able to degrade further, R_3 always equals 40,960 during whole lifecycle. The state continuance thresholds, T_i, of all pools are set equally as 10.

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, 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, but commodity CPUs and platforms seldom support S2.

D. The measurement and Metrics Used in The Experiments

We consider the response time of the system as the most important factor of system performance. To understand how system's performance is affected by the idle node management, we use *slowdown rate* as a metric, which is defined by formula (12) below, where the *wait time* of a job is the difference between the time of a user submitting the job and the time of the job starting on the system.

$$slowdown\ rate = \frac{wait\ time\ with\ dyanamic\ sleep}{wait\ time\ without\ dynamic\ sleep} \quad (12)$$

The *system slowdown rate* is the average of the relative slowdown rates of all jobs. The bigger is the slowdown rate, the more decrement of system performance caused by the management of idle nodes.

We adopted the widely used metric $E*D^n$ [5] of the *power efficiency* to measure the effectiveness of an idle node management technique. As shown in formula (13), it considers both the power consumed by idle node and the corresponding slowdown rate, where n is a weight factor to reflect the user's preference for energy conservation or system performance. In our simulations, n is configured as 1.

$$power\ efficiency = wasted\ power \times (slowdown\ rate)^n \quad (13)$$

E. Configuration Parameters

According to the degradation 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 degraded to the deeper sleep state. Constrained by the reserve capacity threshold, the maximal size of DS_i that can be degraded from B_i is (N_i-R_i). We employ δ_i*(N_i-R_i) as the size of DS_i in the simulation, where δ_i is a fractional constant, i.e. $0 \le \delta \le 1$. 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 2, where simulation results are normalized according to the results of δ=1.

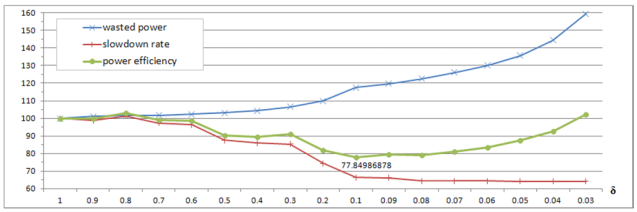

Figure 2. Average management effect with different size of *DS*

Generally speaking, the system saved power increases with the decrease of the size of the *DS*, while the slowdown rate decreases with it. The power efficiency varies with the size of *DS,* which forms a bathtub curve. When δ is 0.1, the power efficiency is the best. Therefore, in the further experiments we used 0.1*(N_i-R_i) as the optimization configuration of degradation set size.

F. The Main Results

The simulations are executed on the benchmark in 5 different scenarios discussed in subsection C.

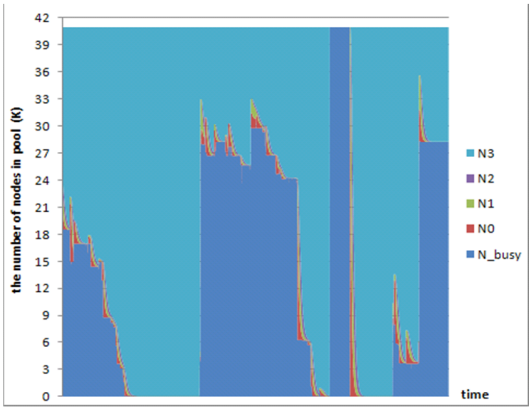

Figure 3. The variation of the numbers of nodes in the ASDMIN scenario

The achievement of close match to S4 on energy efficiency and at the same time to S0 on performance is due to the adaptive mechanism in the ASDMIN model. This is clearly demonstrated in Figure 3, which shows the variation of the number of idle nodes in each reserve pools during the execution of the system on the benchmark.

In Figure 3, the *N_busy* trace is the number of busy nodes in the system. On average, there are 94.28% of idle nodes in

the lowest pool (N_3). In other words, most idle nodes are put into lowest sleep state in most of time.

Normalized by the data obtained in the scenario S0 (i.e. without dynamic sleep), the effects of idle node management in the five different management solutions are shown in Figure 4. In particular, the power consumption by idle nodes in the ASDMIN scenario is decreased by 84.12% in comparison with scenario S0 while the cost of slowdown rate is only 8.85% in comparison with scenario S4.

Thus, applying Formula (13), we have that the power efficiency is optimized by 82.71%. If all idle nodes are always put into the deepest sleep state, shown as S4 in the figures, the power consumption by idle nodes will be reduced by 87.44% in comparison with scenario S0. However, its slowdown rate will be increased significantly by 177.27%. Even compared with the S4 scenario (the deepest sleep scenario), ASDMIN also optimizes the power efficiency by 50.36%.

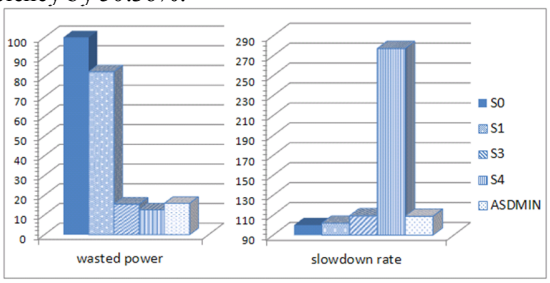

Figure 4. The results of different management scenarios.

VI. Conclusion

To conquer the huge energy waste caused by active idle nodes, the paper proposes a self-adaptive mechanism to manage the sleep depths of idle nodes in large scale systems to balance between energy consumption and system response speed. Idle nodes are classified into different groups according to their sleep states. The nodes in the lower sleep depth are allocated in preference. The state of an idle node is dynamically upgraded or degraded to improve the system power efficiency. Corresponding resource allocation and reclaim algorithms, dynamic state transition algorithms are designed to maintain the proper distribution of idle node among different groups. The self-adaptive mechanism employs two sets of control parameters: (a) reserve pool capabilities; (b) state continuance thresholds. The former decides how large a reserve pool should be. It is adjusted dynamically according to the workload of the system at runtime. The latter represents the users' requirements on performance. It determines how fast the degradation of sleep states of the idle nodes should be made and how often the adjustment of the capabilities of reserve pools should be performed.

The simulation experiments demonstrated that our solution can reduce the power consumption of idle nodes by 84.12% with the cost of slowdown rate being only 8.85%. 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 conducting more experiment with the system in order to gain a full understanding of the relationships between various parameters. We are also exploring the combination of various policies in the selection of idle node for degradation and upgradation of their sleeping states.

Acknowledgment

This work is partly supported by the National High Technology Research and Development Program of China (863 Program) under grant No. 2012AA01A301, the NSF of China under Grant No. 60903044, No. 61003075 and No. 61103193, and the EU FP7 project MONICA on Mobile Cloud Computing under grant No. PIRSES-GA-2011-295222.

References

[1] Top 500, http://www.top500.org, Jun. 2012.

[2] U.S. Environmental Protection Agency. 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. Aug. 2, 2007.

[3] Gary Cook. How Clean is Your Cloud?. Greenpeace International. Apr. 2012.

[4] Rafique M. Mustafa, Ravi Nishkam, Cadambi Srihari, et al. Power management for heterogeneous clusters: an experimental study. Proc. of the 2nd International Green Computing Conference (IGCC'11). Orlando, USA. July 2011.

[5] Yongpeng Liu, Hong Zhu. A survey of the research on power management techniques for high performance systems. Software Practice and Experience, 2010, 40(1): 943-964.

[6] Shekhar Srikantaiah, Aman Kansal and Feng Zhao. Energy Aware Consolidation for Cloud Computing. Proc. of the 2008 Workshop on Power Aware Computing and Systems (HotPower'08), San Diego. Dec. 7, 2008.

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

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

[9] Anshul Gandhi, Mor Harchol-Balter, Michael A. Kozuch. The case for sleep states in servers. Proc. of the HotPower '11, Cascais, Portugal, 2011.

[10] Tibor Horvath, Kevin Skadron. Multi-mode Energy Management for Multi-tier Server Clusters. Proc. of the 17th International Conference on Parallel Architecture and Compilation Techniques, Toronto, Canada, 2008:270-279.

[11] Zhenghua Xue, Xiaoshe Dong, Siyuan Ma, et al. An energy-efficient management mechanism for large-scale server clusters. Proc. of the 2007 IEEE Asia-Pacific Services Computing Conference. 2007:509-516.

[12] Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/l_anl_int/ANL-Intrepid-2009-1.swf.gz. Apr. 2011