PASS: Power-Aware Scheduling of Mixed Applications with Deadline Constraints on Clusters

Cong Liu, Xiao Qin, and Shuang Li
Department of Computer Science and Software Engineering
Auburn University
Auburn, United States of America

Abstract— **Reducing energy consumption has become a pressing issue in cluster computing systems not only for minimizing electricity cost, but also for improving system reliability. Therefore, it is highly desirable to design energy-efficient scheduling algorithms for applications running on clusters. In this paper, we address the problem of non-preemptively scheduling mixed tasks on power-aware clusters. We developed an algorithm called Power Aware Slack Scheduler (PASS) for tasks with different priorities and deadlines. PASS attempts to minimize energy consumption in addition to maximizing the number of tasks completed before their deadlines. To achieve this goal, high-priority tasks are scheduled first in order to meet their deadlines. Moreover, PASS explores slacks into which low-priority tasks can be inserted so that their deadlines can be guaranteed. The dynamic voltage scaling (DVS) technique is used to reduce energy consumption by exploiting available slacks and adjusting appropriate voltage levels accordingly. Simulation results demonstrate that compared with a well-known energy-efficient algorithm - CC-EDF, PASS saves up to 60 percent of energy dissipation. With respect to the number of high-priority tasks meeting deadlines, PASS outperforms the existing approach by over 10 percent without degrading the overall performance. PASS successfully schedules tasks with hard deadlines in a mix of tasks with soft deadlines. In doing so, PASS embraces a new feature that allows clusters to support a variety of real-time applications, making clusters amenable for commercialization.**

I. INTRODUCTION

Over the past years many high-end computing systems have been deployed at large scales, with deployments incorporating tens of thousands of power-hungry resources in clusters to achieve high performance. This volume of resources raises pressing issues of both reliability [4] and cost [16]. For example, as the number of operating resources increases, a huge amount of energy consumption could lead to high temperatures. According to the Arrenhius' equation [4], computing in high temperatures is more error-prone than in an appropriate environment in that the expected failure rate of an electronic component doubles for every 10^oC of increased temperature. Another consequence brought by large energy consumption is the expensive operational cost. According to Eric Schmidt, CEO of Google, what matters most to Google "is not speed but power—low power, because data centers can consume as much electricity as a city" [16]. Such significant energy consumption results in very high cost. Take the example of a single 200-Watt server, such as the IBM 1U*300. The total energy cost for this single server would be $180/year [2]. Given that a cluster normally consists of hundreds of servers, the power cost will be significantly high.

To address cost and reliability issues, recent research has focused on reducing energy consumption in cluster computing systems. Dynamic voltage scaling (DVS) is an effective technique for reducing energy consumption by adjusting the clock speed and supply voltage dynamically [3]. In CMOS circuits, a dominant component of energy consumption is proportional to fv^2, where f is frequency and v is voltage. Energy is a product of power and running time (measured in terms of number of cycles) that is inversely proportional to frequency. Therefore, energy dissipation per cycle is proportional to v^2 [22]. Given a supply voltage, the maximum frequency at which a CPU can run safely decreases with the decreasing voltage. Thus, one can save processor energy by reducing CPU voltages while running it at a slower speed.

In this paper, we focus on scheduling of various task types with different priorities and deadline constraints while minimizing energy consumption. Being able to handle mixed types of tasks, clusters become commercially viable as a wide range of applications are allowed to execute on the clusters. We consider two types of tasks: hard real-time tasks and soft real-time tasks [13]. Specifically, hard real-time tasks may involve the control of offensive weapons and soft tasks may involve ordering supplies [10]. It is reasonable for a scheduler to prioritize hard real-time tasks while maximizing the total number of tasks having their deadlines met. Our approach is energy efficient because after a schedule is made, the processor voltage is adjusted to the lowest possible level on a task-by-task basis at each scheduling point.

The remainder of this paper is organized as follows. A review of recent related works has been given in Section 2. In Section 3, a system model has been described. Section 4 presents the detailed design of the *PASS* algorithm. Section 5 presents a comprehensive set of simulations that evaluate the performance of *PASS*. Finally, conclusions and suggestions for future work appear in Section 6.

II. RELATED WORK

Recently much emphasis has been given to energy conservation techniques for multiprocessor systems in order to improve both system reliability and to minimize operational cost. In [5], three distributed DVS scheduling strategies proposed for power-aware clusters include: (1) using the

CPUSPEED daemon (2) user-driven scheduling from the command-line, and (3) DVS scheduling using internal API controls. A software framework was also implemented to evaluate various scheduling techniques. Kim *et al.* proposed power-aware scheduling algorithms for bag-of-tasks applications with deadline constraints on DVS-enabled cluster systems [12]. The scheduling algorithm applies DVS to save energy while testing each task's schedulability to meet deadlines. Hotta *et al.* designed a strategy of profile-based power-performance optimization by DVS scheduling in high-performance PC clusters [7]. The algorithm divides a program execution into several regions and selects the best gear for power efficiency by using the execution and power profile. Hsu and Feng proposed a power-aware β-adaptation algorithm that adapts CPU frequencies in a DVS-enabled HPC system [8]. They developed a run-time method to estimate the β which is defined as the intensity level of off-chip accesses.

A handful of DVS-based scheduling algorithms have been developed for real-time systems [14][19][6] [1][11][18]. All these DVS-based algorithms adjust the processor speed by exploiting slack times created by the amount of gap between a task's worst case execution time and the task's actual runtime execution time. In real-time systems, the DVS technique is also used to save energy consumption while meeting task deadlines. Researchers have developed various DVS-based scheduling algorithms to save energy under timing constraints [21][15]. In a single processor system, a scheduling algorithm that exploits DVS and dynamic power management (DPM) has been proposed [9]. A couple of energy-efficient scheduling algorithms that combine feasibility analysis of the rate monotonic algorithm and the DVS technique were described in [20]. The aforementioned algorithms, however, do not consider all of the following criteria: task priorities, deadlines, and energy efficiency. *PASS* is the first of its kind that is able to schedule tasks with various priorities and real-time constraints while minimizing energy consumption in clusters.

III. System Model

A. Modeling Clusters

A cluster computing system contains a number of computing elements (CEs) that provide different computing performance in terms of MIPS (Million Instruction Per Second). There is a main server and a number of supplemental computing servers, which are in charge of collecting information from all CEs within the cluster. The main server plays a role for allocating resources to tasks. If the main server fails, a supplemental computing server will take over.

B. Task Model

We consider two types of tasks: hard real-time and soft real-time tasks. PASS uses such a task taxonomy that takes into account consequence of missing deadlines as well as the importance of task properties. Hard real-time tasks are mission critical; the consequences of missing deadlines of hard real-time tasks are catastrophic, e.g. computing the orbit of a moving satellite to make real-time defending decisions [17]. For soft real-time tasks, failures result in degraded but not destroyed performance. Applications that fall in the category of soft real-time tasks include coarse-grained task-parallel computations arising from parameter sweeps, Monte Carlo simulations, and data parallelism. Such applications generally involve large-scale computation to search, optimize, and statistically characterize products, solutions, and design spaces normally do not have hard real-time deadlines.

C. Power Consumption Model

We adopt a commonly used power consumption model in CMOS circuits [3][12]. The major energy consumption by a task is proportional to V^2 and N_{cycle}, where V is the supply voltage and N_{cycle} is the number of clock cycles of tasks [3]. The relationship between the energy consumption denoted as E and the voltage can be mathematically defined as:

$$E = k \cdot V^2 \cdot N_{cycle} , \qquad (1)$$

where k is a proportional constant. In order to reduce the energy consumption, the DVS technique decreases the supply voltage, thereby leading to slowdowns of execution times.

IV. Scheduling Algorithm

The *PASS* algorithm aims to increase the number of real-time tasks that are completed before their deadlines while reducing energy assumption. Moreover, PASS attempts to aggressively boost energy efficiency without sacrificing the schedulability of real-time clusters.

In the first step of PASS, incoming tasks are ranked based upon task priorities and deadlines. After real- time tasks are allocated to computing elements, the processor voltage is adjusted to the lowest possible level on a task-by-task basis. The *PASS* algorithm is outlined in Fig. 1.

```
PASS
// Q is a task queue
Sort Q by decreasing priority, then by increasing
   deadline
for each unscheduled task t∈ Q
  do for each machine m within the cluster
  //visit in random order to balance load
     Visit slacks from latest to earliest
     If t fits within slack while meeting deadline
       Schedule t on m at the latest possible time
       Mark t scheduled
     end if
  until t is scheduled
end for
Each m executes Voltage Assignment
end PASS
```

Fig. 1. The PASS algorithm

A task is composed of execution code, type (i.e. hard or soft), and deadline. Various users may submit a number of tasks with different priorities and deadlines. Our ranking strategy takes priority and deadline into consideration. The scheduler puts all incoming tasks into a task queue. Tasks are sorted by a decreasing order of priority, and then by an increasing order of deadline. The notations used in this section are shown in Table 1.

After the sorting process is done, soft real-time tasks with tight deadlines are given a lower ranking than hard real-time tasks with loose deadlines. This ranking strategy may cause many soft real-time tasks to miss deadlines. If all tasks are scheduled at the earliest point, an obvious disadvantage is that soft real-time tasks with shorter deadlines are unable to be

Table 1. Summary of Notation

Notation	Description
t_i	*ith* task
CE_j	*jth* computing element (CE) in the system
$WCET_{ij}$	worst case execution time of t_i on CE_j
e_{ij}	actual execution time of t_i on CE_j
s_{ij}	start time of tasks t_i on CE_j
d_i	deadline of t_i
pb_{kj}	pre-bound of the k^{th} task slot on CE_j
st_{kj}	start time of the k^{th} task slot on CE_j
ed_{kj}	end time of the k^{th} task slot on CE_j
l_{kj}	length of the k^{th} task slot's slack time on CE_j

completed before their deadlines. This is due to the fact that those hard real-time tasks with longer deadlines have been scheduled much earlier than they should be. To solve this problem, PASS intentionally creates slacks by setting tasks' start times to their latest possible start times. With slacks in place, soft real-time tasks are offered ample opportunities to be scheduled within such slacks. A motivation example is shown in Fig. 2. As shown in Fig. 2a, all tasks are scheduled to the earliest start times. Since hard tasks are scheduled as early as possible, soft real-time task 4 cannot meet its deadline. However, if we schedule hard real-time tasks as late as possible (see Fig. 2b), slacks are created so that soft real-time tasks (e.g. task 4) can be successfully accomplished within such slacks.

Task	Priority	Exec. Time	Deadline
1	Hard	1	3
2	Hard	1.5	7
3	Hard	1	11
4	Soft	0.5	1
5	Soft	1	4.5
6	Soft	1.5	9

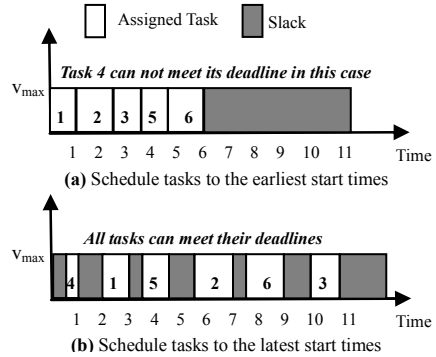

Fig. 2. A motivation example of creating slacks by setting tasks start time to their latest start times and scheduling soft real-time tasks within such slacks.

A. Scheduling Tasks within Slack

The Before illustrating the scheduling algorithm, we define a data structure called task slot. A **task slot** contains two parts: (*i*) slack time and (*ii*) task execution period. The slack time of a task slot is defined as the available time period between this task slot's **pre-bound** and its **start time**. The pre-bound of the task slot equals to its previous task slot's **finish time**, if one existed. For the first task slot, its pre-bound is 0. The start/end time of a task slot is the same start/end time of the corresponding task. Fig. 3 shows the concept of a task slot.

Before scheduling task t_i onto computing element CE_j, the scheduler checks whether the deadline of t_i can be guaranteed:

$$WCET_{ij} <= d_i \,, \qquad (2)$$

where $WCET_{ij}$ is the worst case execution time of t_i on CE_j, and d_i is the deadline of t_i.

t_i will be assigned to CE_j if the above criterion can be satisfied. Instead of setting t_i's start time to the earliest start time, we set its start time, s_{ij}, to the latest possible start time. Thus, we have

$$s_{ij} = d_i - WCET_{ij} \,. \qquad (3)$$

After assigning the first task, a corresponding task slot is created. Let's denote it as the first task slot. The length of this task slot's (the k^{th} task slot where $k=1$) slack time is defined as:

$$l_{kj} = st_{kj} - pb_{kj} \,, \qquad (4)$$

where l_{kj} is the length of the k^{th} task slot's slack time on CE_j, st_{kj} is the start time of the k^{th} task slot on CE_j, and pb_{kj} is the pre-bound of the k^{th} task slot on CE_j. In this case, $k = 1$ and $pb_{kj} = 0$, since it is the first task slot.

For each of the following tasks, t_m, the scheduler checks whether t_m can be scheduled within any slack time while meeting its deadline. The slack time after the last task slot (denoted as the z^{th} task slot) is checked first. The criterion used to verify that t_m on CE_j can be completed before the deadline within this slack is:

$$WCET_{mj} + ed_{zj} <= d_m \,, \qquad (5)$$

where ed_{zj} is the end time of the z^{th} task slot on CE_j.

If t_m can not be scheduled after the last task slot, the scheduler repeatedly searches for remaining slacks starting from the last task slot to the first task slot. The criteria to find a feasible slack time for t_m on CE_j are:

$$WCET_{mj} + pb_{kj} <= min(d_m, st_{kj}) \,. \qquad (6)$$

Eq. (6) defines whether or not t_m can meet its deadline, if scheduled within this slack.

If the above conditions are satisfied, t_m is scheduled within the k^{th} slack (which corresponds to the k^{th} task slot) on CE_j, and its start time is set to:

$$s_{mj} = min(d_m \,, ed_{kj}) - WCET_{mj}. \qquad (7)$$

It is intuitive that scheduling hard real-time tasks as late as possible creates large slack times, enabling other soft real-time tasks to have their deadlines guaranteed within the large slacks.

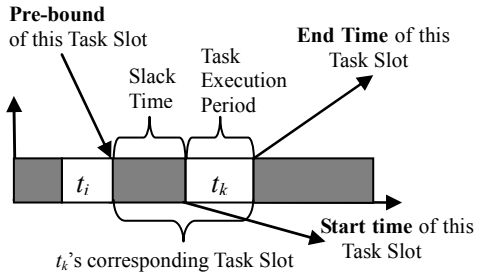

Fig. 3. Illustration of the task slot data structure

B. Voltage Assignment

After generating schedules of all computing elements, we propose a time-sharing scheme to adjust each computing element's voltage to the lowest possible level on a task-by-task basis at each scheduling point. In doing so, our scheme is able

to significantly conserve energy dissipation in computing elements of a cluster. Therefore, we denote the current assigned task set in the ith computing element CE_j as $T_j = \{t_{ij}(e_{ij}, d_{ij}) | i=1, ..., n_j\}$, where n_j is the number of tasks assigned to CE_j, t_{ij} is the ith task on CE_j, and d_{ij} is the deadline of task t_{ij}.

Without loss of generality, the estimated execution times of tasks are assumed to be the Worst Case Execution Times (*WCETs*). Let us denote the maximum supply voltage level of CE_j as v_{max}. After scheduling all the tasks, each task slot on every CE has a corresponding slack, whose length is defined in Eq. (4). For each task slot, *PASS* exploits its corresponding slack and adjusts the voltage accordingly. For CE_j, the corresponding CPU speed for the k^{th} task slot (which corresponds to t_{ij}), v_{kj}, is adjusted to:

$$v_{kj} = v_{\max} \times \frac{WCET_{ij}}{ed_{kj} - pb_{kj}} \times \frac{e_{ij}}{WCET_{ij}} = v_{\max} \times \frac{e_{ij}}{ed_{kj} - pb_{kj}} , \quad (8)$$

where e_{ij} is the actual runtime execution time of t_{ij}. *PASS* adjusts the speed (voltage) within an individual task boundary. In other words, it uses the slack time from the current task for the current task itself. An example of *PASS*'s scheduling and voltage assigning phases is given in Fig. 4.

Task	Priority	Exec. Time	Deadline
1	Hard	1	3
2	Hard	1.5	7
3	Hard	1	11
4	Soft	0.5	1
5	Soft	1	4.5
6	Soft	1.5	9

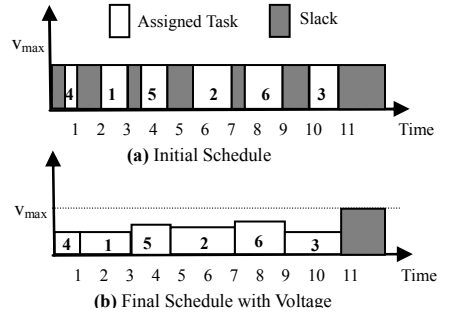

Fig. 4. An example of PASS schedule (a) Initial schedule, (b) Final schedule after adjusting voltage by exploiting slacks on a task-by-task basis.

Lemma. The proposed voltage assignment strategy always adjusts the supply voltage to the lowest possible level while maintaining the schedulability of tasks.

Proof. Suppose that task t_m is to be scheduled within the k^{th} slack on CE_j and its deadline can be met. According to Eq. (6), t_i's WCET is shorter than the length of the k^{th} slack. Thus, it can be written as:

$$WCET_{mj} <= min(d_m, st_{kj}) - pb_{kj} . \quad (9)$$

After scheduling t_m, t_m becomes the k^{th} task slot. The previous k^{th} task slot becomes the $(k+1)^{th}$ one. So that Eq. (9) can be re-written as:

$$WCET_{mj} <= ed_{kj} - pb_{kj} . \quad (10)$$

In accordance to Eq. (8), its supply voltage is adjusted to:

$$v_{kj} = v_{\max} \times \frac{e_{mj}}{ed_{kj} - pb_{kj}} . \quad (11)$$

Thus, the voltage decreasing rate is:

$$\frac{v_{kj}}{v_{\max}} = \frac{e_{mj}}{ed_{kj} - pb_{kj}} \le \frac{WECT_{mj}}{ed_{kj} - pb_{kj}} \le 1 . \quad (12)$$

The final inequality follows from Eq. (10). □

As reported in [1], when the tasks' *WCETs* equal to the Best Case Execution Times (*BCETs*), their algorithms cannot conserve energy. This statement is also true for most of existing DVS-based scheduling algorithms (see, for example, [14][19][6][1][11]), since the existing scheduling schemes adjust the processor speed only by exploiting slack times created by the amount of gap between a task's worst case execution time and its actual runtime execution time. The *PASS* algorithm solves this problem by "naturally" creating slack times, which does not necessarily need the time gap between a task's *WCET* and *BCET*.

V. Experimental Results

We implemented the PASS algorithm tailored for high-performance clusters running both hard and soft real-time tasks. In addition, we conducted extensive experiments to evaluate the performance of PASS. The goal of simulations was to compare the energy consumption and the real-time performance between PASS and a well known energy-efficient DVS scheduling algorithm, named CC-EDF (Cycle conserving EDF) which is proposed by Pillai and Shin [18]. CC-EDF is a cyclic conserving algorithm which exploits Worst Case Execution Time (WCET) scenarios. It re-computes utilization at task level and reduces the operating frequency and voltage when tasks use less than their worst-case time allotment.

A. Parameters

We simulate a cluster system with 32 DVS-enabled processors. Each processor is modeled with Athlon-64, which has an ability to adjust its supply voltage and clock speed. The frequency range of each processor is between 0.8 GHz and 2.0 GHz. In the simulation, we generate 14400 tasks in total. The task execution time follows a uniform distribution. The deadlines and number of tasks were chosen such that the cluster system is *close to its breaking point where tasks start to miss deadlines*. The system parameters are shown in Table 2.

The *Hard Task Acceptance Ratio*, *Overall Acceptance Ratio*, *Energy Consumption per Task*, and *Total Energy Consumption* have been used as the main metrics of evaluation. They are defined as:

$$Hard\ Task\ Acceptance\ Ratio = \frac{N_{hard}}{N_{total}} , \quad (13)$$

where N_{hard} is the number of hard tasks meeting deadlines, and N_{total} is the total number of tasks.

$$Overall\ Acceptance\ Ratio = \frac{N_{hard} + N_{soft}}{N_{total}} , \quad (14)$$

where N_{soft} is the number of soft tasks meeting deadlines.

$$Energy\ Consumption\ per\ Task = \frac{E}{N_{hard} + N_{soft}} , \quad (15)$$

where E is the total energy consumption.

Table 2. Characteristics of system parameters

Parameter	Value(fixed)-(varied)
Number of tasks	(1600)-(1600,3200,4800, 6400,8000,9600,11200,12800,14400)
CPU	Athlon-64
Number of CPUs	(32)-
Frequency	(0.8 GHz-2.0 GHz)
Voltage	(0.9 V-1.5V)
WCET/BCET ratio	(2)-(1,2,3,4)
Task execution time range (Uniform distribution)	(1,500) second

B. Energy Consumption

The first experiment set was to compare the energy consumption of *PASS* with *CC-EDF*. Moreover, by varying the $\frac{WCET}{BCET}$ ratio, we intend to study the relationship between energy consumption and variability of the actual workload.

Fig. 5 shows the total energy consumed to execute 1600 tasks. We observe that *PASS* saves up to 60 percent of the energy over *CC-EDF*. The reason that *PASS* can achieve such significant energy savings is because *PASS* creates large integrated slacks by scheduling tasks to the latest possible start time. *CC-EDF* schedules tasks according to the rule of Minimum Completion Time (*MCT*) which always schedules a task to its earliest possible start time. In this way, *EDF* hardly leaves any slack time that may be used by the DVS technique.

Furthermore, the simulation results indicate that the energy consumption would be highly dependent on the variability of the actual workload. When $\frac{WCET}{BCET}=1$, there is no CPU time for *CC-EDF* to reclaim, while PASS is able to save energy by exploiting the slack times among tasks. Once we increase the $\frac{WCET}{BCET}$ ratio, energy savings of both algorithms continue to increase.

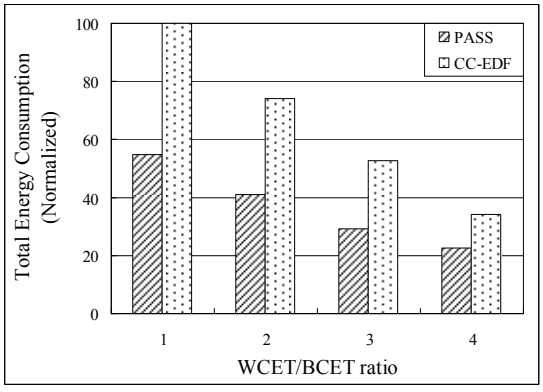

Fig. 5. Total energy consumption (Execute 1600 tasks).

C. Performance

The The second experiment set was to compare the performance of *PASS* against *CC-EDF*. We performed simulations for different task loads in order to examine the performance consistency.

With respect to *Hard Task Acceptance Ratio*, from Fig. 6, we observe that *PASS* yields 10% better performance on average than *CC-EDF*. When the number of tasks is below 6400, *PASS* guarantees that all hard tasks can meet their deadlines. As the number of tasks further increases, *PASS* is still able to schedule most of the hard tasks while *EDF* is no longer able to reach similar performance level. The reason is because *PASS* always schedules hard tasks first, which helps meet hard tasks' deadlines. *CC-EDF* does not consider task priority, which results in a number of un-schedulable hard tasks.

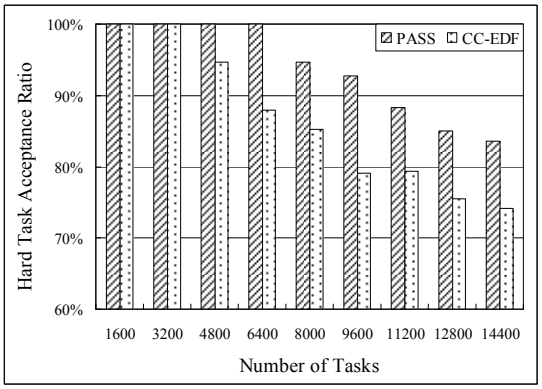

Fig. 6. Hard task acceptance ratio.

For *Overall Acceptance Ratio*, from Fig. 7, it is observed that *CC-EDF* yields about 2% better performance on average than *PASS*. In order to create more slacks, *PASS* sacrifices a few soft tasks. However, the fact that only 2% *Overall Acceptance Ratio* performance degradation is tolerable since *PASS* achieves significant energy savings and better *Hard Task Acceptance Ratio* performance.

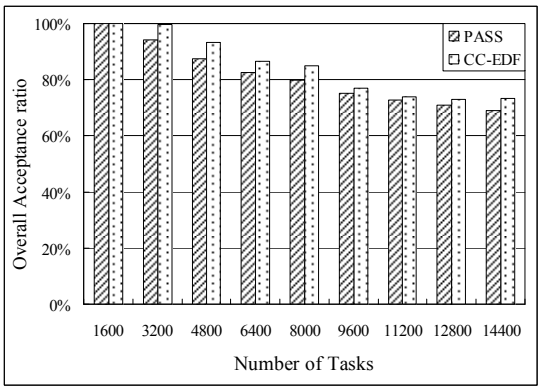

Fig. 7. Overall acceptance ratio.

D. Energy Consumption per Task

As Fig. 7 shows, both algorithms cannot schedule all tasks when the number of tasks exceeds 3200. It becomes unfair if we compare two algorithms using the *Total Energy Consumption* metric, since the number of accepted tasks by using *PASS* is different from the number by using *CC-EDF*. Instead, we use the *Energy Consumption per Task* as the metric. In this way, we are able to study the effect brought by the number of tasks on energy consumption.

As shown in Fig. 8, *PASS* consistently performs better than *CC-EDF* with respect to *Energy Consumption per Task*. It is again because *PASS* decreases the processor speed for each

task by utilizing the corresponding slack time. An interesting observation is that as the number of tasks increases, the *Energy Consumption per Task* achieved by *PASS* also increases. Once there are more incoming tasks, less slack times will be available since more tasks need to be scheduled within those slack times.

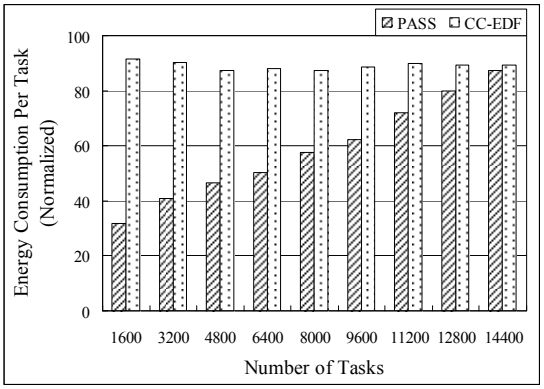

Fig. 8. Energy consumption per task ($\frac{WCET}{BCET} = 2$)

VI. CONLUSTION

In this paper, we address DVS scheduling of mixed tasks with deadline constraints on power-aware clusters. We developed an algorithm called Power Aware Slack Scheduler (*PASS*) for tasks with different priorities and deadlines. *PASS* schedules high-priority tasks first in order to meet their deadlines. Moreover, *PASS* explores slacks into which low-priority tasks can be inserted so that their deadlines can be guaranteed. *PASS* reduces energy consumption by exploiting available slacks and adjusting appropriate voltage levels accordingly. Detailed simulations demonstrate that *PASS* effectively reduces energy consumption and increases the *Hard Task Acceptance Ratio,* without degrading the *Overall Acceptance Ratio* much. In the future, we will investigate the schedulability analysis of *PASS* in order to provide deadline guarantees as well as address reliability issues.

ACKNOWLEDGMENT

The work reported in this paper was supported by the US National Science Foundation under Grants No. CCF-0742187 and No. CNS-0713895, Auburn University under a startup grant, and the Intel Corporation under Grant No. 2005-04-070.

REFERENCES

[1] H. Aydin, R. Melhem, D. Mosse and P.M. Alvarez, "Power-aware Scheduling for Periodic Real-time Tasks," *IEEE Transactions on Computers,* 2004, vol. 53, no. 5, pp. 584 - 600.

[2] R. Bianchini and R. Rajamony, "Power and Energy Management for Server Systems," *Computer*, 2004, vol. 37, no. 11, pp. 68-74.

[3] T. D. Burd and R. W. Brodersen, "Energy Efficient CMOS Microprocessor Design," *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*, 1995, pp. 288-297.

[4] W. Feng, "Making a Case for Efficient Supercomputing," *ACM Queue*, 2003, vol. 1, no. 7, pp. 54-64.

[5] R. Ge, X. Feng, and K. W. Cameron, "Performance constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters," *Proceedings of the ACM/IEEE SC*, 2005, pp. 34-44.

[6] F. Gruian, "Hard Real-Time Scheduling Using Stochastic Data and DVS Processors," *Proceedings of the International Symposium on Low Power Electronics and Design*, 2001, pp. 46–51.

[7] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi, "Profile-based Optimization of Power Performance by using Dynamic Voltage Scaling on a PC cluster," *Proceedings of 20th IEEE International Parallel and Distributed Processing Symposium*, 2006, pp. 340-347.

[8] C. Hsu and W. Feng, "A Power-Aware Run Time System for High Performance Computing," *Proceedings of the ACM/IEEE SC*, 2005, pp. 1-9.

[9] N. K. Jha, "Low-Power System Scheduling, Synthesis and Displays," *IEEE Proceedings on Computers and Digital Techniques*, 2005, vol. 152, no. 3, pp. 344-352.

[10] J.K. Kim, et al., "Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment", *Journal of Parallel and Distributed Computing*, 2007, vol. 67, no. 2, pp. 154-169.

[11] W. Kim, J. Kim, and S. L. Min, "A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis," *Proceedings of Design, Automation and Test in Europe*, 2002, pp. 788–794.

[12] K.H. Kim, R. Buyya, and J. Kim, "Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters," *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid*, 2007, pp. 541-548.

[13] P.A. Laplante, Real-Time Systems Design and Analysis. *Wiley-IEEE Press*, 2004.

[14] S. Lee and T. Sakurai, "Run-time Voltage Hopping for Low power Real-Time Systems," In *Proceedings of the 37th Design Automation Conference*, 2000, pp. 806–809.

[15] J.R. Lorch and A.J. Smith, "PACE: A new approach to dynamic voltage scaling," *IEEE Transactions on Computers*, 2004, vol. 53, pp. 856-869.

[16] J. Markoff and S. Lohr, "Intel's huge bet turns iffy," *New York Times Technology Section*, 2002, Section 3, Page 1, Coumn 2.

[17] National Aeronautics and Space Administration [Online]. Available: http://liftoff.msfc.nasa.gov/academy/rocket_sci/satellites. [Accessed Jan. 8, 2008].

[18] P. Pillai and K.Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *Proceedings of 18th ACM Symposium on Operating Systems Principles*, 2001, pp. 89–102.

[19] D. Shin, J. Kim, and S. Lee, "Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications," *IEEE Design and Test of Computers*, 2001, vol. 18, no. 2, pp.20–30.

[20] T. Wei, P. Mishra, K. Wu, and H. Liang, "Online Task-Scheduling for Fault-Tolerant Low-Energy Real-Time Systems," *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 2006, pp. 522-527.

[21] A. Weissel and F. Bellosa, "Process Cruise Control- Event-Driven Clock Scaling for Dynamic Power Management," *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2002, pp. 238-246.

[22] N. H. E. Weste and K. Eshraghian, Principles of CMOS VLSI Design. *Addison-Wesley*, 1993.