Energy Reduction in Weakly Hard Real Time Systems

Yashwant Singh, Mayank Popli, Shiv Shankar Prasad Shukla
Department of CSE & IT,
Jaypee University of Information Technology
Waknaghat, HP, India.
yashu_want@yahoo.com, mayankpopli27@gmail.com, yash22222k1@yahoo.com

***Abstract*—Real Time Systems can be classified into Hard, Soft and Firm Real Time Systems depending on the consequences of a task missing its deadline. A Hard Real Time System produce required results before the specified time bound. In Soft Real Time Systems the few misses of deadline is acceptable if they have no harm. The Generalized case of Real Time Systems is Weakly Hard Real Time System which is motivation of the observation for real time applications when some deadline misses are acceptable as long as they are spaced evenly.**

This paper is an attempt to provide a state-of-the-art review of basic model of Real Time System, clock driven scheduling, priority driven scheduling and proposes an Inverse Rate Monotonic algorithm. The Inverse Rate Monotonic algorithm effectively reduce the energy by 15% along with Dynamic Voltage Scaling strategy which provides significant energy savings while maintain real time deadline guarantee. A tool has been created in C language to observe the scheduling and energy consumption of task by Inverse Rate Monotonic.

Keywords-Hard Real Time Systems, Soft Real Time Systems, Scheduling, Rate Monotonic, Deadline Monotonic, Dynamic Voltage Scaling, Dynamic Power Down, Deeply Red Pattern, Mixed Pattern.

I. INTRODUCTION

A real-time system is expected to respond not just quickly, but also within a predictable period of time. A real time task is one for which quantitative expression of time is needed to describe its behavior. This quantitative expression of time usually appears in the form of a constraint on the time at which the task produces results. The most frequently occurring time constraint is a deadline which is used to express that a task is required to compute its results within tasks deadline [1].

There are various applications of real time systems, which includes Automated Car Assembly Plant, Temperature Control in Chemical Plant, Robot used in Recovery, Cellular Systems, Missile Guidance Systems, Car Braking System and Security Systems. As depicted from the above listed wide range of applications, real time systems can be classified into hard, soft, and firm real time systems depending on the consequences of a task missing its deadline.

A hard real time system is one that is constrained to produce its results within a certain predefined time bounds. The system is considered to have failed whenever any of its hard real time tasks does not produce its required results before the specified time bound. In soft real time systems the few misses of deadline have no harm. It just decreases the overall performance of the system. When a system is considered soft real time system, there is some room for lateness. The generalized case of real time systems is weakly hard real time systems. It is motivation of observation for real time applications, where some deadline misses are acceptable as long as they are spaced evenly. The systems where missing of few deadlines can be tolerated at the cost of compromising quality of service limiting to the acceptable quality measured as (m,k) model [13] known as weakly hard real time systems.

To achieve better quality of service in terms of tolerance and energy consumption while honouring minimum tolerance requirement, we need maximum energy available within strict timing constraint. Reducing the energy consumption increases the computation time which increase the chance of missing the deadline i.e. energy and deadline are counterproductive. There is a need for design of an efficient resource manager that minimizes system energy consumption while giving better tolerance to hard real time system with arbitrary deadline exposed to transient faults. Thus, issues of energy and real time constraints can be integrated into single framework to achieve reduced system energy consumption within deadline by the use of check pointing, tolerance patterns, pre-emption control, speed fine tuning, delay start, speed patterns, criticality and sensitivity.

In this paper we propose a general scheduling algorithm to minimize the energy consumption in weakly hard real time systems, i.e., the (m, k) model, which requires at least m out of k consecutive instances of a task meet their deadlines. Firstly, we proposed a strategy to partition real time jobs into mandatory and optional part in order to meet weakly hard real time constraint. Secondly, we introduce an approach which can effectively reduce the energy by 15% along with Dynamic voltage scaling (DVS) strategy which provides significant energy savings while maintaining real time deadline guarantees. We also ensure time constraints i.e. the completion of an operation after its deadlines is considered useless which may cause a complete failure, so the task has to be completed within a specified time limit otherwise it will cease to failure.

978-1-4577-0697-4/12/$26.00 ©2012 IEEE

Remainder of this paper is organized as follows: section 2 describes the basic model of a real time system and the various types of real time systems which consist of hard, soft and firm real time systems. Section 3 illustrates various types of scheduling and existing techniques for reducing energy consumption in real time systems as a part of our literature survey. Section 4 provides proposed approach for reduction of energy consumption in real time systems. The result is illustrated in section 5. Section 6 presents the conclusion and the future scope.

II. A BASIC MODEL OF REAL TIME SYSTEM

This figure shows a simple model of a real time systems in terms of its functional blocks. The sensors are interfaced with the input conditioning block,which in turn is connected to the input interface. The output interface, output conditioning and the actuator are interfaced in a complementary manner. A sensor converts some physical characteristic of its enviornment into electrical signals. An actuator is any device that takes its input from the output interface of a computer and converts these electrical signal into some physical actions on its enviornment. A popular actuator is a motor. The computer signals usually need conditioning before they can be used by the actuator. This is called output conditioning.Few important conditions are Voltage Amplification , Voltage level shifting and Frequency range shifting,which is used to reduce the noise components in a signal [7].

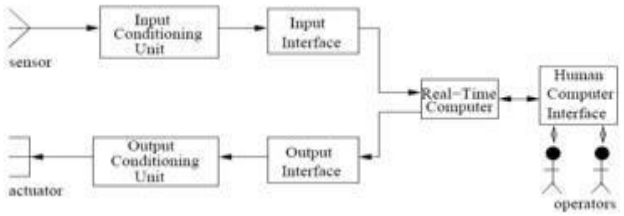

Fig. 1: Model of a real time system

Many types of noise occur in narrow bands, therefore the signal must be shifted from the noise bands so that noise can be filtered out. Interface actuators commands from the CPU are delivered to the actuator through an output interface. An ouput interface converts the stored voltage into analog form and then outputs this to the actuator. Digital computers can not process analog signals. Therefore, analog signals need to be converted into digital. Sampling is done by a capacitor circutory that stores voltage levels. These voltage levels can be discretized after a sampling into a step waveform [2,3].

A. *Hard real time system*

The completion of an operation after its deadlines is considered useless. When a process is considered hard real-time, it must complete its operation by a specific time. If it fails to meet its deadline, its operation is without value and the system for which it is a component could face failure. One example of hard real time system is an anti missile system [7].

B. *Soft real time system*

Soft real-time systems are systems where few misses of deadline does not leads to failure and just degrades the overall performance of the system, for instance web browsing. Fetching of web page may take few seconds or minutes, we still do not consider the system to have failed, but merely express that the performance of the system has degraded [7].

C. *Weakly hard real-time system*

It is the generalized case of real time systems and is measured by a general model i.e. m-k model where it needs to meet at least m deadlines in every consecutive k jobs. Another example of a real-time task is a monitor. A task periodically performs a set of monitoring actions. The sampling period may be carefully computed or decided as a rule of thumb. Again missing an occasional deadline means that the action from monitoring will be delayed by some bounded period of time. Provided that the effect of such a delay can be tolerated and deadlines can be missed [4].

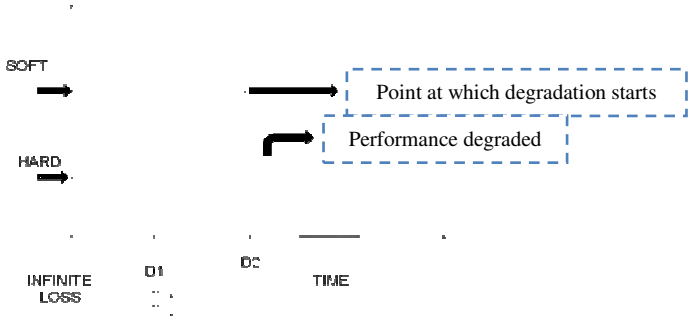

Figure 2: Graph for hard and soft real time system

III. LITERATURE SURVEY

A. *Clock Driven Scheduling*

Clock driven schedulers make their scheduling decisions regarding which task to run next only at the clock interrupt points. Clock driven schedulers are also called off line schedulers because these schedulers fix the schedule before the system starts to run, i.e the scheduler pre- determines which task will run when. Therefore, these schedulers incur very little run time overhead. However, a prominent shortcoming of this class of schedulers is that they cannot satisfactorily handle aperiodic and sporadic tasks since the exact time of occurrences of these tasks cannot be predicted [5,6].

B. *Priority Driven Scheduling*

It is based on the priority of the jobs. When the higher priority is ready it pre-empts the lower priority which is running. There are two approaches of priority driven scheduling:-

a) STATIC APPROACH :

Priorities are assigned to tasks once for all and every job of a task will have same priority.

I. Rate Monotonic Algorithm (RM)

Rate monotonic algorithm [14] is a dynamic pre-emptive algorithm based on static priorities. The rate monotonic algorithm assigns static priorities based on task periods. Here task period is the time after which the tasks repeat and inverse of period is task arrival rate. For example, a task with a period of 10ms repeats itself after every 10ms. The task with the

978-1-4577-0697-4/12/$26.00 ©2012 IEEE

shortest period gets the highest priority, and the task with the longest period gets the lowest static priority. At run time, the dispatcher selects the task with the highest priority for execution [7,15].

EXAMPLE: In this example we have three tasks T_1, T_2, T_3

T_1 (4, 1), T_2 (5, 2), T_3 (20, 5)

Where 4, 5, 20 are the period and 1, 2, 5 are the execution time of the task. It has a hyper period 18 i.e. The Total period of the task this is calculated by taking LCM of the periods.

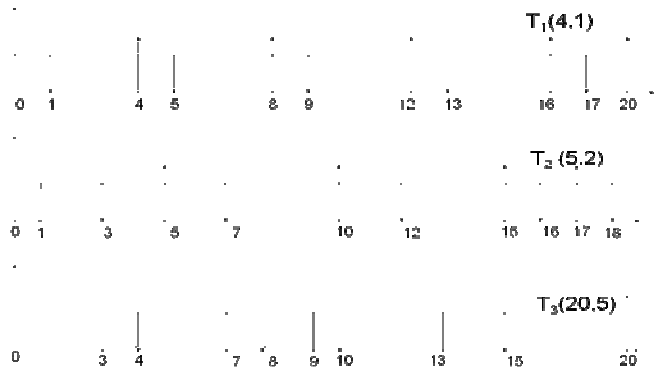

Fig. 3: Example of RM

First we will release T1 because it has shorter period and after execution of first job of T1 we will release T2 who has next shorter period and its jobs execute in the background of T1 so for this reason the execution of the first job in T2 is delayed until the first job of T1 completes and the fourth job in T2 is pre-empted at the time 16 when the fifth job in T1 is released because here the job in T1 has a highest priority than T2 because T1 has shorter period than T2 so here instead of complete the fourth job in T2 we will complete the fifth job in T1 so T1 is pre-empted at time 16. Similarly T3 is executes in the background of T1 and T2. The jobs in T3 execute only when there is no job in the higher priority tasks ready for execution. Since there is always at least one job ready for execution until time 18 the processor never idles.

Preemption: In case of preemption the execution of jobs can often be interleaved. The scheduler may suspend the execution of a less priority jabs and give the processor to more priority job. Later when the more priority job completes the scheduler returns the processor to less priority job so the job can resume execution. The interruption of job execution is called pre-emption.

II. Deadline Monotonic Algorithm

One of the problems with RM is that many systems will need job deadlines shorter than the job's period which violates the assumption mentioned earlier. A solution to this problem arrived in 1982 with the introduction of the Deadline Monotonic (DM) algorithm [14, 15]. With DM, a job's priority is inversely proportional to its relative deadline. That is to say, the shorter the relative deadline higher the priority.

In this example we have three tasks T_1, T_2, T_3

T_1 (2, 0.6), T_2 (2.5, 0.2), T_3 (3, 1.2)

Where 2, 2.5, 3 are the period which is same as the deadline of the task and 0.6, 0.2, 1.2 is the execution time of the task. It has a hyper period 10 i.e. total period of the task which is calculated by taking LCM of the periods.

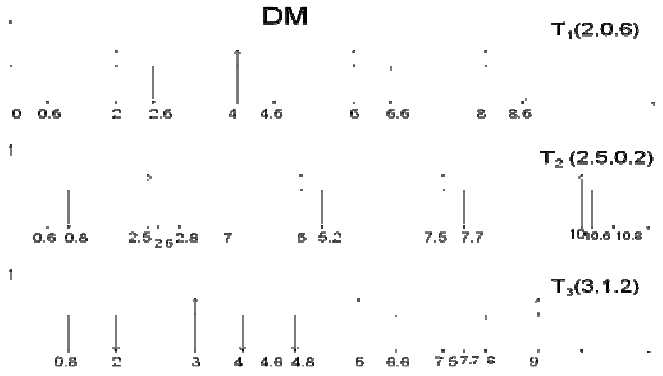

Fig. 4: Example of DM

In this T1 has the higher priority because it has a shorter deadline, so first we release T1, when its execution time completes then we release T2 who has next higher priority. It job executes at the background of T1.So for this reason the execution of the first job in T_2 is delayed until the first of T_1 completes. The 2nd job in T_3 is pre-empted at time 4, when the 3rd job in T_1 is released because it has higher priority than T_3. Similarly the 3rd job in T_3 is pre-empted at time 7.5.Since there is always at least one job ready for execution until hyper period, the processor never get idle until that time.

b) DYNAMIC APPROACH

The priorities of tasks may change from request to request; different jobs will have different priorities. One of the dynamic approaches is:

1) Earliest Deadline First (EDF)

The algorithm is optimally used to schedule jobs on a processor as long as preemption is allowed and jobs do not have the resources [10]. It is based on absolute deadline where earlier the deadline, highest the priority. In EDF scheduling, at each of the point where scheduling is done the task having the shortest deadline is taken up for scheduling.

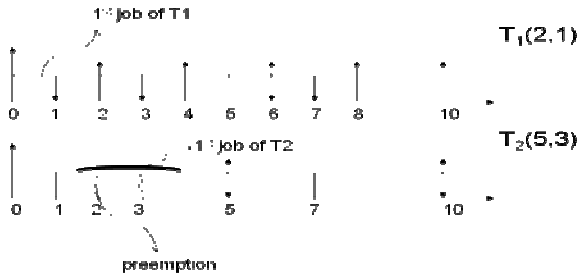

Fig. 5: Example of EDF

In this we have two tasks

T_1 (2, 1), T_2 (5, 3)

T1 has a higher priority because its deadline is earlier than T_2 .First we release T_1 when its execution will complete , then we release T_2.At the time 0,first job J(1,1) and J(2,1) of both tasks are ready .But the absolute deadline of J(1,1,) is 2 while the absolute deadline of J(2,1) is 5 .So consequently J(1,1) has a higher priority and executes .When J(1,1) completes J(2,1) begins to execute. At time 2 the 2nd job T_1 is released and its deadline is 4, earlier than the deadline of job 1 of T_2. Here 2nd job of T_1 pre-empts the first job of T_2 and execute. When job 2nd of T_1 completes , the processor ten executes Job 1 of T_2 .At time 4 job 3 of T_1 is released its deadline is 6 which is later than the deadline of job 1 of T_2 , hence processor continue to execute the first job of T_2. At time 5 T_2 has completed its first job now the deadline of T_2 is 10 and deadline of T_1 is 6. Here T_1 has higher priority because it has deadline earlier than T_2 so this way jobs continue to execute in tasks T_1 and T_2.

C. Power Consumption Reduction Techniques

a) Dynamic Voltage Scaling(DVS)

It is based on adjusting the processor voltage and frequency on the fly. Power requirement depends on operating frequency as well as voltage i.e. the dynamic processor power is strictly increasing convex function of the processor speed. The DVS reduce the processor speed to the extent it is possible to obtain higher energy saving. The speed of a frequency dependent component is said to be reduced if it is either operating at lower voltage or frequency. The task response time increases with the reduced processor speed leading of the following consequences: A release may miss its deadline when it is feasible at higher speed. The longer execution time will be able to decrease the dynamic energy consumption of the processor. Frequency is dependent component remains active for longer time and increase energy consumption. Longer execution time implies more losses in energy due to leakage current [8].

b) Dynamic Power Down

The dynamic power down technique suggest to switching to sleep state (low energy consuming state) of the idle components to reduce the energy consumption. For switching from active state to sleep state and back from sleep state to active state will incur an overhead called the DPD overhead. Thus, switching to sleep state too often may be counterproductive and estimated a threshold value for shutting down the components by comparing the energy consumption required in idle state with energy consumption on power down state and wakeup. If the energy consumption for switching to sleep state is less than or equal to the energy consumption in idle time then switching to sleep state is preferred over leaving the component idle [8,9].

D. Partitioning Techniques

The partitioning techniques are used in case of selection of m jobs from a window of k consecutive jobs for execution.

a) Deeply Red Pattern(Red_ Pattern):

This pattern was proposed by Koren & Shasha [12]where first releases out of consecutive releases of task are mandatory. Mathematically, this can be described as

$$\pi_i^j = \begin{cases} 1, & 0 \le j \bmod k_i < m_i \\ 0, & otherwise \end{cases} \quad j = 0, 1, \ldots, k_i$$

When is 1, release is mandatory while it is optional in case 0 is assigned to. We refer this pattern as Red_Pattern. Advantage of applying this pattern to a task set for energy minimization is that it aligns the optional jobs together so that a component has a better opportunity to switch into sleep state to save energy. For a task whose critical speed is higher than or equal to the highest possible speed the operating speed should never be scaled down. Assigning Red_Pattern to such a task helps to extend the idle interval for switching to sleep state. However, for a task whose critical speed is lower than

Red_Pattern overloads the system leading to large size busy intervals and need more energy to be feasible.

b) Evenly Distributed Pattern(Even_Pattern)

In this evenly distributed pattern in which the first release is always mandatory and the distribution of mandatory and optional is even i.e., alternating [11]. Mathematically, this can be described as

$$\pi_i^j = \begin{cases} 1, & if\ j = \left\lfloor \left\lceil \frac{j \cdot m_i}{k_i} \right\rceil * \frac{k_i}{m_i} \right\rfloor \\ 0, & otherwise \end{cases} \quad for\ j = 0, 1, \ldots k_i -$$

We refer it to as Even_Pattern.

c) Reverse Evenly Distributed Pattern(Rev_Pattern)

This pattern is a reverse of the Even_Pattern, hence the first release is always optional and the distribution of mandatory and optional is alternating [12]. Mathematically:

$$\pi_i^j = \begin{cases} 0, & if\ j = \left\lfloor \frac{j \cdot (k_i - m_i)}{k_i} \right\rfloor * \frac{k_i}{(k_i - m_i)} \\ 1, & otherwise \end{cases} \quad j = 0, 1, \ldots k_i -$$

We refer it as Rev_Pattern.

d) Hybrid Pattern(Hyd_Pattern)

In this instead of assigning same pattern to all the tasks in the task set, we assigned different type of patterns (Red_Pattern or Even_Pattern) to each task. For example, task is partitioned into mandatory and optional according to Red_Pattern while and could be assigned Red_Pattern or Even_Pattern. Thus, yielding possible combination of pattern assignment where is the number of the tasks in the task set [11].

e) Mixed Pattern(Mix_Pattern)

The hybrid pattern allows a task in the task set to be scheduled by Red_Pattern or Even_Pattern. In both cases at least the first release of each task is mandatory (if not more e.g., (m, k) first two releases of both the task are mandatory with the Hyd_Pattern) and are in phase hence, will overload the system, forcing it to be feasible with high energy requirement. Therefore, to improve the performance of Hyd_Pattern suggested a mixed pattern (Mix_Pattern) which combines the Hyd_Pattern with the Rev_Pattern yielding 3^n [12].

IV. PROPOSED MODIFICATION: INVERSE RM

Our main focus is to reduce the energy consumption of a real time system and to enhance the performance of the

978-1-4577-0697-4/12/$26.00 ©2012 IEEE

system. Reduction in energy consumption increases the computation time which therefore increases the chance of missing the deadline i.e. energy and deadline are counterproductive. So, we have introduced an approach which can effectively reduce the energy by 15% along with DVS strategy which provide significant energy savings while maintaining real time deadline guarantees. It is inverse of rate monotonic scheduling where shorter the period higher is the priority but in case of inverse RM higher is the period, higher is the priority of the task set. It includes peripheral devices and energy is calculated using the existing techniques such as DVS and DPD. Let's observe the scheduling and energy consumption of task set T using Inverse RM:

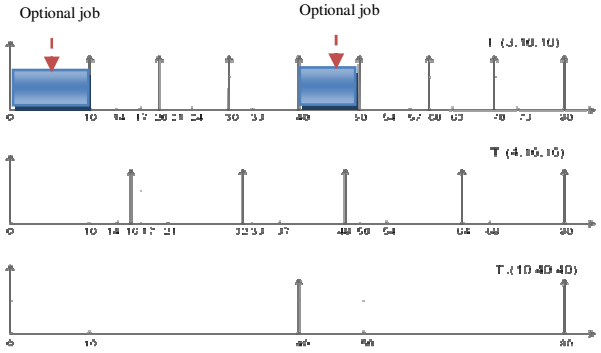

Fig. 6: Example of IRM

In this example we have three task set T1 (3, 10, 10 , 1,4) , T2 (4, 16, 16, 1, 1) , T3(10 , 40, 40 , 1 ,1) which includes execution time e , period p, deadline d , m jobs which can miss deadline , k consecutive jobs. Here 3, 4, 10 are the execution time of the task T1 ,T2, T3 and 10, 16, 40 are the period and deadline of the tasks T1, T2, T3. It has the hyper period 80 i.e. total period of the task which is calculated by taking the L.C.M. of the periods. As release is available at time 0, three of the tasks are available but as T3 has the higher period so it will have the highest priority whereas T2 will have the 2nd priority and T1 had the 3rd priority. Now T3 will take 10 units of time, the first job of the task T3 is executed then T2 will start executing, it will execute till the next release of job. After the execution of task T2, T1 will execute its 2nd job as 1st job is set to be optional by using m-k model where m jobs can miss their deadlines accordingly. Since, 1st and the 4th release is made optional of the task T1, will execute 3 units of every job. It will reduce the energy consumption of the system.

A. *Calculation of Energy by Inverse RM*

A = { T1, T2, T3 } where T_i = { (e_i , p_i , d_i , m_i , k_i) } : { (3, 10, 10, 1, 4) , (4, 16, 16, 1, 1) , (10, 40, 40, 1, 1)}

Utilization of the Tasks:

$U = e_i / p_i$

T1 = 3/10 = .3; T2 = 4/16 = .25;

T3 = 10/40=.25

Total Utilization:

.3+.25+.25 = .80 <1

Energy $= e_i * [(\gamma_p(S_i) + \gamma_d]*n$ .....[8]

Where e_i = execution time of the task

S_i = operating speed

γ_p = speed of the processor

γ_d = speed of the device

n = no. of jobs

Power = Energy/ unit time: $e\gamma$;

Frequency scalable (Processor): $\gamma_p \alpha S^3$

Where s : operating speed

Dynamic voltage scaling DVS: $\gamma_p(Si) \alpha Si^3$

Energy $= e_i * [(\gamma_p(S_i) + \gamma_d]*n$

T1 = 3*[(10)^3+50]*6 = 18900

T2 = 4*[(10)^3+100]*5 = 22000

T3 = 10*[(10)^3+150]*2 = 23000

Total Energy = T1+T2+T3 = 63900

B. *Comparison between Inverse RM and RM*

Calculation of Energy by RM:

A = { T1, T2, T3 } where T_i = { (e_i , p_i , d_i , m_i , k_i) } : { (3, 10, 10, 1, 4) , (4, 16, 16, 1, 1) , (10, 40, 40, 1, 1) }

Utilization of the Tasks:

$U = e_i / p_i$

T1 = 3/10 = .3; T2 = 4/16 = .25;

T3 = 10/40=.25

Total Utilization:

.3+.25+.25 = .80 <1

Energy $= e_i * [(\gamma_p(S_i) + \gamma_d]*n$

Where e_i = execution time of the task

S_i = operating speed

γ_p = speed of the processor

γ_d = speed of the device

n = no. of jobs

Power = Energy/ unit time: $e\gamma$;

Frequency scalable (Processor): $\gamma_p \alpha S^3$ where s : operating speed

Dynamic voltage scaling DVS: $\gamma_p(Si) \alpha Si^3$

Energy $= e_i * [(\gamma_p(S_i) + \gamma_d]*n$

T1 = 18900

T2 = 39358

T3 = 41147

Total Energy = T1+T2+T3 = 99405

By implementing inverse RM, total energy is 63900 therefore energy consumption is reduced by 35.7%. Hence proved algorithm is feasible.

V. RESULTS AND DISCUSSION

Our experimental results demonstrate that our approach can greatly reduce the number of idle intervals and thus the power consumption, while still providing (*m*, *k*)-firm guarantee. We also propose a novel pre-emption control scheme, which can be well incorporated into our dynamic scheduling algorithm. Extensive experiments have been performed and demonstrate the effectiveness of our approach.

Our results indicate that the energy-consumption of the real time systems along with the greedy algorithms when the system is significantly energy-constrained is reduced by 15% and enhance the performance.

Energy Consumption in case of RM and IRM in case of T1 and T2 fixed and varying the time period of T3.

978-1-4577-0697-4/12/$26.00 ©2012 IEEE

Example I

Task 1: (3,10,10,1,4); Task 2: (4,16,16,1,1); Task 3: (10,40,40,1,1)

Energy consumed by task 1 by rm: 18900
Energy consumed by task 2 by rm: 39358
Energy consumed by task 3 by rm: 41147
Total energy consumption by rm is: 99405
Energy consumed by task 1 by irm: 18900
Energy consumed by task 2 by irm: 22000
Energy consumed by task 3 by irm: 23000
Total energy consumption by irm is: 66200

A linear increase as period of task 3 is changed, while the periods of task1 and task 2 remain fixed, as shown in the graph.

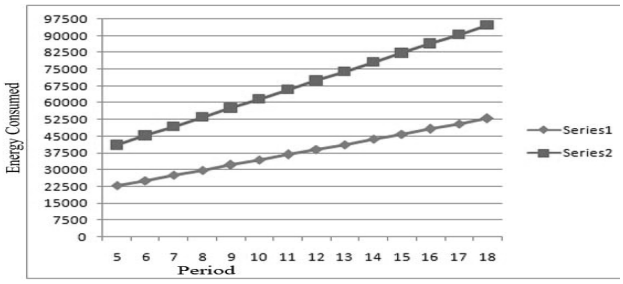

Fig. 7: Graph for RM and IRM for Task 3.
Series 1: IRM and Series 2: RM

Example II

Task 1: (5,15,15,1,1); Task 2: (7,20,20,1,1); Task 3: (8,60,60,1,1)

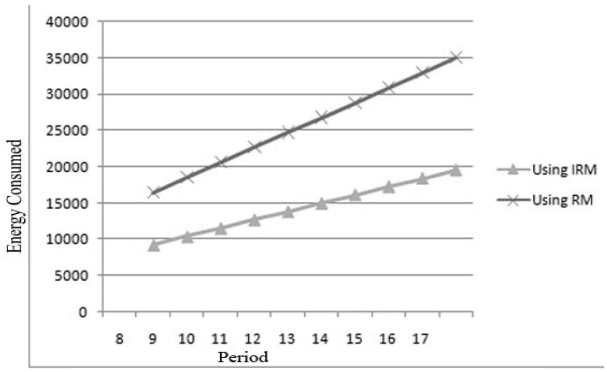

Fig. 8: Graph for RM and IRM for Task 3

Energy consumed by task 1 by rm: 21000
Energy consumed by task 2 by rm: 41325
Energy consumed by task 3 by rm: 16458
Total energy consumption by rm is:78783
Energy consumed by task 1 by irm: 21000
Energy consumed by task 2 by irm: 23100
Energy consumed by task 3 by irm: 9200
Total energy consumption by irm is: 53300

A linear increase as period of task 3 is changed, while the periods of task1 and task 2 remain fixed, as shown in the graph.

VI. CONCLUSION AND FUTURE SCOPE

The system consists of a core processor a number of peripheral devices, which have different power characteristics. Energy consumption is critical in the design of pervasive real-time computing platforms. The power consumption for peripheral devices, as a significant part of the overall power consumption, must be taken into consideration to reduce the system wide power consumption. Along with the adopted single known mandatory/optional partitioning strategy, we propose to incorporate different partitioning strategies based on the power characteristics of the devices as well as the application specifications. We introduce a feasibility condition, and based on which, we propose an algorithm to performance the mandatory/optional job partitions. In this , we presented a dynamic DPD and DVS approach(calculation) to reduce the system wide energy consumption while guaranteeing the QoS requirement, which are modelled as the (m,k)-constraints. The energy saving performance of our approach comes from the facts that we dynamically change the mandatory/optional job settings, and merge the idle intervals effectively by delaying the execution for mandatory jobs.

In the future, we would like to expand this work beyond the deterministic/absolute real-time paradigm presented here. In particular, we will investigate other scheduling technique with probabilistic or statistical deadline guarantees. We will also explore integration with other energy conserving mechanisms.

REFERENCES

[1] B.Dasarathy, "Timing constraints of real time systems: Constructs for expressing them, methods for validating them," *IEEE Transactions on Software Engineering, Vol. 11(NO.1), pp 80-86,* January 1985.

[2] Steve Health, "Embedded Systems Design," Elsevier, 2003.

[3] Brian Santo, "Embedded Battle Royale," *IEEE Spectrum, pp 36-42,* December 2001

[4] G. Bernat, A. Burns, and A. Llamosi, "Weakly Hard Real-Time Systems," *IEEE Transactions on Computers, vol. 50, no. 4, pp. 308* –321, Apr. 2001.

[5] C Liu. and J.W. Layland, "Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment," *Journal of ACM, Vol. 20(1), pp 46-61,* 1973.

[6] C. M. Krishna and K.G. Shin, "Real-Time Systems," *Tata McGraw-Hill,* 1997.

[7] Jane W. S. Liu, "Real-Time Systems," University of Ilinois at Urbana-Champaign, ISBN-10: 0130996513 Prentice Hall, 2000.

[8] Smriti Agrawal, Rama Shankar Yadav, Ranvijay, "A Preemption Control Technique for System Energy Minimization of Weakly Hard Real-Time Systems", *in the International Conference on Software , Artificial Intelligence, Networking and Parallel/Distributed* (SNPD-2008),

[9] Smriti Agrawal, Rama Shankar Yadav, Ranvijay, "A Speed Fine Tuning Techniques for System Energy Minimization of Weakly Hard Real-Time Systems", *accepted for publication in International Journal of Computer and Application, ACTA Press Publication*, Canada, 2009.

[10]Smriti Agrawal, Rama Shankar Yadav, Ranvijay, " a preemption control approach for energy aware fault tolerant real-time systems" *International journal of recent trend in engineering, Vol. 1, No. 1, PP. 381-386*, ISBN: 1797-9617

[11] M.Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm deadlines," *IEEE Transaction on Computers, vol.44, no.4, pp. 1443-1451,* 1999.

[12] G.Koren and D. Shasha, "D-over: An optimal on-line scheduling algorithm for overloaded real-time systems," *in proc. 13th IEEE Real Time Systems Symposium, Phoenix, Arizona, USA, pp.290-300*, December 1992,

[13] Moncef Hamdaoui and Parameswaran Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm deadlines," *IEEE*

Transactions on Computers, vol. 44, no. 12, pp. 1443 1451, December 1995.

[14] Mark H. Klein, John P. Lehoczky, and Ragunathan Rajkumar, "Rate-monotonic analysis for real-time industrial computing," *Computer, pp.25-32*, 1994.

[15] J.P. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behaviour," *In Proceedings of 10th IEEE Real-Time Systems Symposium, pp. 166-171*, 1989.

978-1-4577-0697-4/12/$26.00 ©2012 IEEE