Reliable Dynamic Voltage Scaling for Real-Time Systems with Uncertain Execution Time and Resource Constraints

G. AZHAGUNILA
Department of Information Technology
Pondicherry Engineering College
Puducherry-605014, India

J. CHANDRAKALA
Department of Information Technology
Pondicherry Engineering College
Puducherry-605014, India

K. SANDHYA RANI
Department of Information Technology
Pondicherry Engineering College
Puducherry-605014, India

Abstract:

The main aim of this work is to develop a Dynamic Voltage Scaling (DVS) algorithm for real- time system with resource constraints and the system thus developed is fault tolerant as well. The system is assumed to contain independent periodic tasks. Earliest Deadline First scheduling algorithm is considered in this. The algorithm helps in meeting the deadlines of all the tasks and also ensures that the total power consumption is minimized. The other objective is to develop a fault tolerant system. The proposed system is designed to handle hardware faults. Thus the proposed system is energy efficient and reliable.

Keywords: Energy efficient; Dynamic Voltage Scaling; Resource constraints; Fault tolerance.

1. Introduction:

The critical applications that are being done by the computer in real-time environment must produce desired result at the correct time. Now-a-days battery operated devices are extensively used in various fields such as space, military. In such applications, the battery life is limited and continuous charging is not possible. Since energy consumption is need of the hour, energy efficient real- time systems are being developed in the recent years. As such now, the foremost objectives of energy efficient real- time system are

(1) To produce the correct results in time / To meet the deadline of all tasks
(2) To reduce the total energy consumed by the processor

In a hard real- time system the deadlines of the 'n' tasks must be met (i.e.) no task can be left as unexecuted as the output of a hard real system is critical. A task set consisting of 'n' periodic tasks, contains the Worst Case Execution Time (WCET) and a corresponding period (Pi). The period of the task is equal to its deadline. Deadline is the time within which the task must be completed.

A processor following the EDF algorithm always executes the task whose absolute deadline is the earliest. EDF is a dynamic priority-scheduling algorithm; the task priorities are not fixed but change on the closeness of their absolute deadline. EDF is also called the deadline-monotonic scheduling algorithm. EDF is an optimal uniprocessor-scheduling algorithm. That is, if EDF cannot feasibly schedule any task set on a uniprocessor system, than there is no algorithm that can schedule it. If all task are periodic and have relative deadlines equal to their periods than the test of schedulability is as follows:

If the total CPU utilization of the task set is not greater than 1, the task set can be feasibly scheduled on a single processor by the EDF algorithm.

i.e.

$$U > 1$$

Where, U is the utilization factor. If it's greater than 1, the task set cannot be scheduled using EDF algorithm.

In a real- time system, the task set will have the Worst Case Execution Time (WCET) and Deadline (Di) such that WCETi < Di for all i. (Only tasks with WCETi < Di can be scheduled successfully in a real- time system, else the task set is simply discarded). As such, there is always a time gap between the deadline of the task and its actual execution time. In an energy efficient system, we reduce the energy consumption by making processor process at lower frequencies. Dynamic Voltage Scaling is used to minimize the energy consumed. DVS allows a processor to dynamically change its speed and voltage at run time, thus increasing energy efficiency.

Since energy is directly proportional to frequency, when the frequency is decreased, the energy consumed is also reduced. Hence, a task can be executed at lower frequency. At lower frequency, the task takes more time to complete than its actual execution time. We see that the actual execution time added up with extra time is less

than its deadline (i.e.) though the processor works at lower frequencies, the task is completed within its deadline. This is then applied to a task set consisting of 'n' tasks.

In the following Figure 1, the tasks are executed at the maximum frequency of the processor and hence the processor utilizes the 100% power supplied to it.

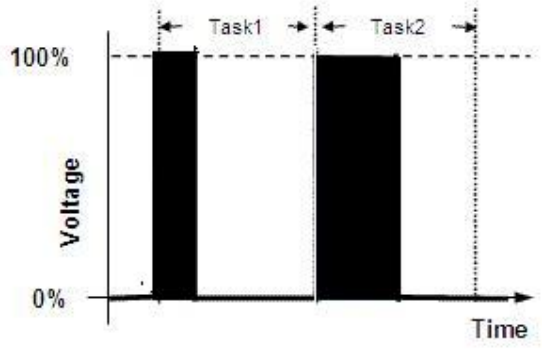

Fig. 1 Execution of tasks at maximum voltage of the processor

The following Figure 2, shows the time and voltage relationship after applying DVS technique. The output of the DVS technique is the task set that is being done at decreased voltage levels with increased execution time.

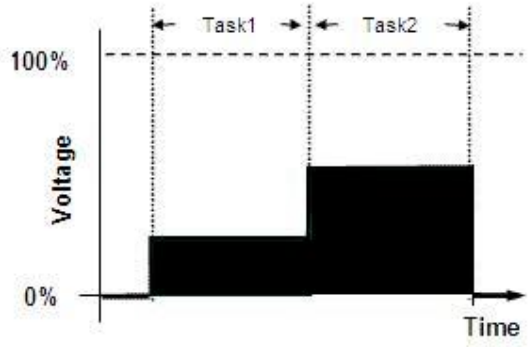

Fig. 2 Execution of tasks at decreased voltage levels using DVS technique

The output/result of a real-time system is vital in all applications. Timing Constraints is always there in a real-time system. However, the system faults may also add up to the complexity. Hence there is a need to develop a fault tolerant system. In a fault tolerant system, the system faults will not lead to the system failure (i.e.) Fault-tolerance is the ability of a system to maintain its functionality, even in the presence of faults. There are two types of faults: 1) Hardware faults and 2) Software faults. In Hardware faults, faults occur in any of the hardware components of the system like the processor, the sensors etc leading to system failure. In these places, system failure can be avoided by implementing hardware redundancy. In Software faults, faults occur in the task that is being executed. It may be due to erroneous inputs, run time errors in the program etc.

Software fault tolerance is the ability for software to detect and recover from a fault that is happening or has already happened in either the software or hardware in the system in which the software is running in order to provide service in accordance with the specification. Software fault tolerance is a necessary component in order to construct the next generation of highly available and reliable computing systems from embedded systems to data warehouse systems. Software fault tolerance is not a solution unto itself however, and it is important to realize that software fault tolerance is just one piece necessary to create the next generation of systems.

2. Literature review:

In Dynamic voltage scaling for multitasking real-time systems [1], the system mainly focuses on three modules. They are: 1) EDF scheduling algorithm 2) Variable continuous frequency assignment 3) Handling bounded frequencies.

In EDF scheduling algorithm, the tasks are scheduled in the order of increasing deadlines. The constraint (utilization factor > 1) is checked since the tasks are periodic with Ti = Di for task i. This scheduled task set is given as input to the next module.

The second module considers the scheduled task set as input. Each task is divided into number of bins, where in each bin consists of equal number of execution cycles. For each bin of a task the optimal frequency and voltage values are formed. This is found for all tasks in the given task set. The frequency values obtained are continuous. The bins of the task and their associated frequencies are given as input to the next module.

In the third module the frequencies of the bins of all tasks are scanned and restricted to the discrete frequencies. The time overhead and frequency overhead involved in switching from one frequency to another are also considered. This is done without compromising the deadlines of the task. These are called the intra task and inter task scheduling.

The algorithm proposed in this paper has a time complexity of O (m^2) where m is the sum of number of tasks and total number of bins.

Software fault tolerance is the ability of the system to withstand faults that have been caused due to errors in the software. Such faults are unpredictable in nature and the source of the errors may not be found easily in this case. Software fault tolerance can be achieved by implementing software redundancy. Software redundancy can be achieved by implementing the following four methods.

1) Triple Modular Redundancy (TMR) model
2) Primary Backup (PB) model
3) Imprecise Computational (IC) model
4) (*m*, *k*)-firm deadline model

In the TMR approach, three versions of a task are executed concurrently and the results of these versions are compared for choosing the most accurate result from the three results. In PB approach, two copies (Primary and Backup) of a task are run simultaneously on two different processors. In the IC model, a task is divided into mandatory and optional parts. The mandatory part must be completed before the task's deadline for acceptable quality of result. The optional part refines the result. The characteristics of some real-time tasks can be better characterized by (*m*, *k*)-firm deadlines in which *m* out of any *k* consecutive tasks must meet their deadlines.

In Fault-Tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems [13] describes a fault tolerant scheduling algorithm in a multiprocessor environment. The method implemented in this paper for providing fault tolerance is Primary Backup model (PB model). In the PB approach, two versions are executed serially on two different processors and an *acceptance test* is used to check the result. The backup version is executed (after undoing the effects of primary version) only if the output of the primary version fails the acceptance test, either due to processor failure or due to software failure. However developing reliable uniprocessor system is the need of the hour as a single processor system is also prone to errors. And hence, we focus on developing a reliable single processor system which can handle software faults.

3. Proposed system:

The proposed system has three modules

1. EDF scheduling with resource constraints
2. Dynamic Voltage Scaling
3. Fault tolerance

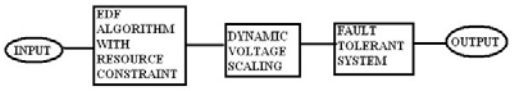

Fig. 3 Modules of the Proposed System

3.1 *EDF scheduling with resource constraints:*

Earliest Deadline First scheduling algorithm is an effective algorithm known. The tasks are scheduled according to the increasing order of their deadlines (i.e.) the task with least deadline is executed first. In our proposed system, the tasks are assumed to be independent and periodic. As such, P_i, D_i and $WCET_i$ denote the time period, deadline and worst case execution time of a task i. Here deadline is equal to the period of a task. Before actually scheduling, the constraint is checked. In EDF algorithm, when the deadline of a task is equal to its period, the Utilization Factor (U) must be less the one. Else, the task set is discarded. Utilization Factor is calculated from the formula,

$$U = \sum_{i=1}^{n} WCETi/Pi \qquad (1)$$

Where $WCET_i$ is the Worst Case Execution Time
P_i is the Time Period

The Worst Case Execution Time is calculated from the formula:

$$WCET = Execution\ Cycle/F \qquad (2)$$

Where, F is the frequency of the processor. Hence before scheduling, the constraint is checked.

Since it's a uniprocessor system, the resource constraints do not affect much. However EDF algorithm is a dynamic priority algorithm where the priority of a task is determined by its deadline. And this may affect the scheduled task set. A task might need some resources, such as data structures, variables, and communication buffers, for its execution. Each resource may have multiple instances. Every task can have two types of accesses to a resource:

(1) Exclusive access, in which case, no other task can use the resource with it or

(2) Shared access, in which case, it can share the resource with another task (the other task also should be willing to share the resource).

Resource conflict exists between two tasks *Ti* and *Tj* if both of them require the same resource and one of the accesses is exclusive.

The resource manager plans actions that include which software to use which resources to achieve the maximum system level benefit and to optimize the real-time performance of sets of application software. As input, it is given the static characteristics of both the hardware system and the software system. Based on these, it makes resource allocation decisions and has the ability to modify certain performance parameters.

3.2 *Dynamic voltage scaling*:

Dynamic voltage scaling and dynamic frequency scaling (DFS) allow adjusting processor voltage and frequency at runtime. Usually, higher processor voltage and frequency leads to higher system throughput while energy reduction can be obtained using lower voltage and frequency. Recent trends in modern processor architecture provide support for these two mechanisms. Dynamic Voltage Scaling is an effective way to minimize CPU energy. The main principle that is exploited in DVS is that the voltage of a processor is proportional to its frequency. When the processor works at lower frequencies than its maximum frequency, it uses less energy. In DVS technique, the processor executes each task at various frequencies in such a way that the task is executed within its deadline and also the energy utilization is minimized. Here we consider frequency levels 2.2 GHz, 2.0 GHz, 1.8 GHz, 1.5 GHz, and 1.0 GHz. After the tasks are scheduled in EDF algorithm, the task's deadline, time period and the frequencies are scanned. Each task is then assigned with a frequency such that it's WCETi < Di. And the Utilization Factor of the task set is calculated with the new WCET and period. If the calculated Utilization Factor exceeds one then the task set is discarded. The task set is thus executed at the maximum frequency of the processor since it cannot be scheduled with the new Execution time of the tasks.

If the Utilization factor is less than one, then the task set can be scheduled using the EDF algorithm. Thus the tasks are scheduled in the increasing order of the deadlines of the tasks. The waiting time, turnaround time and the frequency with which each task must be executed are listed. The total energy consumed by the processor is calculated.

3.3 *Fault tolerant system:*

In Fault Tolerance system, the system is designed in such a way that the faults or unexpected system behavior during system execution are detected and managed in such a way that system failure does not occur.

There are two strategies for software fault tolerance - *error processing* and *fault treatment*. Error processing aims to remove errors from the software state and can be implemented by substituting an error-free state in place of the erroneous state, in two ways. And they are:

a) *Error recovery*: Error recovery can be achieved by either forward or backward error recovery. The second strategy, fault treatment, aims to prevent activation of faults and so action is taken before the error creeps in. The two steps in this strategy are fault diagnosis and fault passivation.

b) *Error compensation*: This is done by compensating for the error by providing redundancy.

In this paper, we focus on Error compensation method to handle software faults i.e. we provide software redundancy to develop a fault tolerant system. We utilize the PB approach with a slight modification in the design.

In general, in PB approach two copies of the task are run on different processors. The primary copy is first started for execution and the Backup is started later. The worst case computation time of a primary copy may be more than that of its backup i.e. the primary copy of a task has the worst case execution time and the backup copy has the actual execution time. The other attributes and resource requirements of both the copies are identical. The backup copy of a task is run only when the primary copy has failed during execution. Each task may encounter failure due to software failure, i.e., if the primary fails, its backup always succeeds.

In this paper, instead of having two processors, we have only one processor. After scheduling the task set based on EDF algorithm and DVS technique, the tasks are inserted in the task queue. The primary copy of a task is inserted followed by its backup copy. The algorithm discussed in this paper is Distance Myopic algorithm. A notion of d*istance* is introduced, which decides the relative difference in position between primary and backup copies of a task in the task queue. The efficiency of the algorithm depends mainly on the relative position of a task's primary copy from its backup copy.

We follow the following procedure in Distance myopic algorithm.

i) Prior to inserting tasks in the task queue, the tasks are ordered in the increasing order of their deadlines. The DVS technique is applied and the frequency with which each task must be executed is found. Since, the tasks are to be executed at decreased frequency levels; the execution time of the tasks is different from its WCET.

ii) A task's backup copy is inserted in the queue in a position which does not affect the schedule and all the tasks are executed within their deadlines.

iii) Hence the tasks' primary copy and backup copy's position depends largely on their deadlines.

The following is an example task queue with $n = 4$. Pr_i and Bk_i denote the primary copy and backup copy of the task i respectively.

Pr 1	Pr 2	Pr 3	Bk 1	Bk 2	Bk 3	Pr 4	Bk 4

The positioning of backup copies in the task queue relative to their primaries can easily be achieved with minimal cost:

i) By having two queues, one for primary copies (*n* entries) and the other for backup copies (*n* entries), and

ii) Merging these queues before invoking the scheduler, based on the deadline to get a task queue of 2*n* entries.

4. Results:

The proposed system is simulated in windows environment with the code implemented in C. The hardware profile describes the multiple operating points of the processor, each with a separate frequency and power consumption. Unless otherwise specified, the following hardware profile {(1.6, 1.484), (1.4, 1.420), (1.2, 1.276), (1.0, 1.164), (800, 1.036), (600, 0.956)}, is used where 1.6, 1.4, 1.2, 1.0, 800, 600 denote the processor frequency in GHz and MHz, and 1.484, 1.420, 1.276, 1.164, 1.036, 0.956 denote the corresponding voltage.

For simulation, the scheduling task set is generated by the following approach

- Tasks are periodic and independent.
- The number of tasks in a set is given as the input to the system.
- The deadline, Time period and the worst case execution time are randomly generated.
- With these parameters, the scheduling is done using Earliest Deadline First algorithm and the frequency for each task is obtained.

The proposed system considers the resource constraints of the tasks and is also reliable. The performance metric used for comparison are percentage of power consumption and level of fault tolerance. Since the system is developed to handle hardware faults, these faults do not result in system failure which is very much essential in a hard real- time system. These features are not present in the existing systems. Since variable frequency is assigned to each bin of a task, the time overhead is more and energy consumption is reduced by 60% in the existing system. In the proposed system, we use variable frequency to each task in the task set and hence the time overhead is minimal and the energy consumption is reduced by 70%. It can be seen from the results that the proposed system had less power consumption than the existing one, thus leading to more energy efficiency.

The following graph Fig. 4 illustrates the % of power saved for various numbers of tasks. The % of power saved decreases with the increase in number of tasks in general. However certain deviations in the graph are due to the random generation of execution cycles and the deadlines of the tasks.

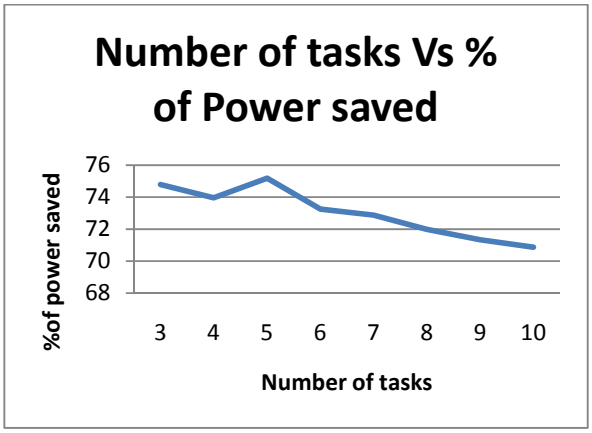

Fig. 4 Graph illustrating the relationship between the number of tasks and % of power saved.

The following graph Fig. 5 is drawn against the number of tasks and miss ratio. Miss ratio is given by

Miss ratio=1-hit ratio

Miss ratio is the number of times the task set has been discarded since the utilization factor has exceeded one. The Miss ratio is expressed in %. The graph shows that the miss ratio when the number of tasks increase. However a deviation is found where the number of tasks is equal to 7 and 8. When the number of task is 7 the miss ratio is 40% and when its 8, its 20% and which again raises to 40% t number of tasks is equal to 9 and 10.

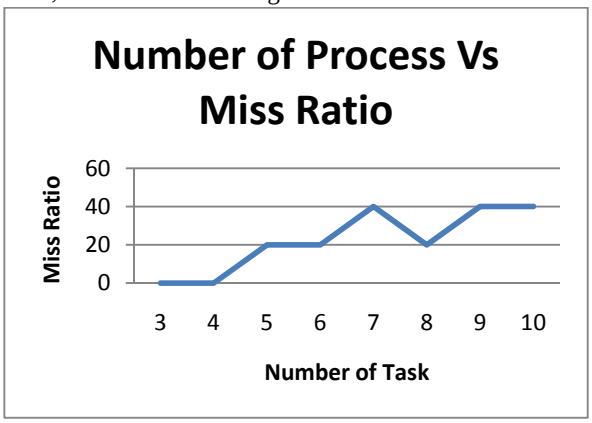

Fig. 5 Graph illustrating the relationship between the number of tasks and Miss Ratio.

5. Conclusion and future works:

In this work, a reliable energy efficient system is considered. This work schedules independent periodic tasks and handles resource constraints and faults in the system. The major contribution of this work is the development of a fault tolerant system that can effectively schedule the tasks and can handle resource constraints.

For future work, security issues can be considered for such real-time distributed systems. Another important future work is extending this technique for a heterogeneous distributed real-time system with precedence constraints as well.

References

[1] Changjiu Xian, Yung-Hsiang Lu and Zhiyuan Li (2008), Dynamic voltage scaling for multitasking real-time systems with uncertain execution time, IEEE Transactions on computer aided design of integrated circuits and systems, Vol. 27, No.8, pp 1467-1478.

[2] Abdullah M. Elewi, Medhat H. A. Awadalla and Mohamed I. Eladawy (2008), Energy Efficient Real-Time Scheduling of dependent tasks sharing resource, Proceedings of the 2008 High Performance Computing & Simulation Conference ©ECMS, pp 107-113.

[3] Vincent N´elis_, Joel Goossens, Raymond Devillers and Dragomir Milojevic (2008), Power-Aware Real-Time Scheduling upon Identical Multiprocessor Platforms, IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing.

[4] Jiong Luo, Niraj K. Jha and Li-Shiuan Peh (2007), Simultaneous Dynamic Voltage Scaling of Processors and Communication Links in Real-Time Distributed Embedded Systems, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 15, No. 4, pp 427-437.
[5] Claudio Scordino and Giuseppe Lipari (2006), A Resource Reservation Algorithm for Power-Aware Scheduling of Periodic and Aperiodic Real-Time Tasks, IEEE Transactions on Computers, Vol. 55, No. 12, pp 1509-1522.
[6] Gang Quan and Xhaobo Sharon (2006), Energy efficient DVS schedule for fixed-priority real-time systems, ACM transactions on design automation of electronic systems, Vol. V, No. N.
[7] Jihong Kim and Dongkun Shin (2006), Dynamic voltage scaling of mixed task sets in priority-driven systems, IEEE Transactions on computer aided design of integrated circuits and systems, Vol. 25, No.3, pp 438-453.
[8] Ying Zhang and Krishnendu Chakrabarty (2006), A Unified Approach for Fault Tolerance and Dynamic Power Management in Fixed-Priority Real-Time Embedded Systems, IEEE transactions on computer-aided design of integrated circuits and systems, VOL. 25, NO. 1.
[9] Jian-Jia Chen and Tei-Wei Kuo (2005), Voltage-Scaling Scheduling for Periodic Real-Time Tasks in Reward Maximization, Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05).
[10] Jianfeng Mao, Qianchuan Zhao and Christos G. Cassandras (2004), Optimal Dynamic Voltage Scaling in Power-Limited Systems with Real-Time Constraints, 43rd IEEE Conference on Decision and Control, pp 1472-1477.
[11] Haka Aydin, Rami Melhem, Daniel Mosse and Pedro Meija-Alvarez (2004), Power aware scheduling for periodic real-time task, IEEE transactions on computers, Vol. 53, No. 5, pp 584-600.
[12] Yang Yu and Viktor k. Prasanna (2003), Resource allocation for independent real-time tasks in heterogeneous systems for energy minimization, Journal of Information Science and Engineering 19, pp 433-449.
[13] G. Manimaran and C. Siva Ram Murthy (1998), A Fault-Tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems and Its Analysis, IEEE transactions on parallel and distributed Systems, Vol. 9, No 1