An Algorithm to Reduce the Time Complexity of Earliest Deadline First Scheduling Algorithm in Real-Time System

Jagbeer Singh
Dept. of Computer Science and Engineering
Gandhi Institute of Engg. & Tech.
Gunupur, Rayagada, India-765022
willybokadia@gmail.com

Bichitrananda Patra
Dept. of Information Technology
Gandhi Institute of Engg. & Tech.
Gunupur, Rayagada, India-765022
bnpatra@gmail.comdr

Satyendra Prasad Singh
Dept. of Master of Computer Application
Gandhi Institute of Compt. Studies
Gunupur, Rayagada, India-765022
spsingh1@hotmail.com

***Abstract*—To this paper we have study to Reduce the time Complexity of Earliest Deadline First (EDF), a global scheduling scheme for Earliest Deadline First in Real Time System tasks on a Multiprocessors system. Several admission control algorithms for earliest deadline first are presented, both for hard and soft real-time tasks. The average performance of these admission control algorithms is compared with the performance of known partitioning schemes. We have applied some modification to the global earliest deadline first algorithms to decrease the number of task migration and also to add predictability to its behavior. The Aim of this work is to provide a sensitivity analysis for task deadline context of multiprocessor system by using a new approach of EFDF (Earliest Feasible Deadline First) algorithm. In order to decrease the number of migrations we prevent a job from moving one processor to another processor if it is among the m higher priority jobs. Therefore, a job will continue its execution on the same processor if possible (*processor affinity*). The result of these comparisons outlines some situations where one scheme is preferable over the other. Partitioning schemes are better suited for hard real-time systems, while a global scheme is preferable for soft real-time systems**.

Keywords- Real-time system; task migration, earliest deadline first, earliest feasible deadline first.

I. INTRODUCTION *(HEADING 1)*

Real-time systems are those in which its correct operation not only depends on the logical results, but also on the time at which these results are produced. These are high complexity systems that are executed in environments such as: military process control, robotics, avionics systems, distributed systems and multimedia.

Real-time systems use scheduling algorithms to decide an *order of execution* of the tasks and an *amount of time* assigned for each task in the system so that no task (for hard real-time systems) or a minimum number of tasks (for soft real-time systems) misses their deadlines. In order to verify the fulfillment of the temporal constraints, real-time systems use different exact or inexact *schedulability tests*. The schedulability test decides if a given task set can be scheduled such that no tasks in the set miss their deadlines. Exact schedulability tests usually have high time complexities and may not be adequate for online *admission control* where the system has a large number of tasks or a dynamic workload. In contrast, inexact schedulability tests provide low complexity sufficient schedulability tests.

The first schedulability test known was introduced by Liu and Layland with the Rate Monotonic Scheduling Algorithm [Liu, 1973] (RM). Liu and Layland introduced the concept of *achievable utilization factor* to provide a low complexity test for deciding the schedulability of independent periodic and preemptable task sets executing on one processor.

In Earliest Deadline First scheduling, at every scheduling point the task having the shortest deadline is taken up for scheduling. The basic principle of this algorithm is very intuitive and simple to understand. The schedulability test for *EDF* is also simple. A task is schedule under *EDF*, if and only if it satisfies the condition that total processor utilization ($\boldsymbol{U_i}$) due to the task set is less than **1**.

With scheduling periodic processes that have deadlines equal to their periods, *EDF* has a utilization bound of 100%. Thus, the schedulability test for *EDF* is:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

Where the $\{C_i\}$ are the worst-case computation-times of the n processes and the $\{T_i\}$ are their respective inter-arrival periods (assumed to be equal to the relative deadlines).

The schedulability test introduced by Liu and Layland for RM states that a task set will not miss any deadline if it meets the following condition: $U \leq n(2^{1/n} - 1)$. Liu and Layland provided a schedulability tests that fails to identify many schedulable task sets when the system is heavily overloaded. After the work of Liu and Layland, many researchers have introduced improvements on the schedulability condition for RM for one and multi processors. These improvements include the introduction of additional timing parameters in the schedulability tests and transformations on the task sets. It is a well-known fact that when more timing parameters are

introduced in the schedulability condition better performance can be achieved.

For example let us Consider 3 periodic processes scheduled using *EDF*, the following acceptance test shows that all deadlines will be met.

Table 1: Task Parameters

Process	Execution Time = C	Period = T
P1	1	**8**
P2	2	5
P3	4	10

The utilization will be:

$$\frac{1}{8}+\frac{2}{5}+\frac{4}{10}=0.925=92.5\%$$

The theoretical limit for any number of processes is 100% and so the system is schedulable.

EDF has been proven to be an optimal uniprocessor scheduling algorithms [8].This means that if a set of tasks is unschedulable under *EDF*, then no other scheduling algorithm can feasible schedule this task set. The *EDF* algorithm chooses for execution at each instant in the time currently active job(s) that have the nearest deadlines. The *EDF* implementation upon uniform parallel machines is according to the following rules [2], No Processor is idled while there are active jobs waiting for execution, when fewer then ***m*** jobs are active, they are required to execute on the fastest processor while the slowest are idled, and higher priority jobs are executed on faster processors.

A formal verification which guarantees all deadlines in a real-time system would be the best. This verification is called feasibility test.

Three different kinds of tests are available:-

- Exact tests with long execution times or simple models [11], [12], [13].
- Fast sufficient tests which fail to accept feasible task sets, especially those with high utilizations [14], [15].
- Approximations, which are allowing an adjustment of performance and acceptance rate [1], [8].

For many applications an exact test or an approximation with a high acceptance rate must be used. For many task sets a fast sufficient test is adequate.

EDF is an appropriate algorithm to use for online scheduling on uniform multiprocessors. However, their implementation suffers from a great number of migrations due to vast fluctuations caused by finishing or arrival of jobs with relatively nearer deadlines. Task migration cost might be very high. For example, in loosely coupled system such as cluster of workstation a migration is performed so slowly that the overload resulting from excessive migration may prove unacceptable [3]. Another disadvantage of *EDF* is that its behavior becomes unpredictable in overloaded situations. Therefore, the performance of *EDF* drops in overloaded condition such that it cannot be considered for use. In this paper we are presenting a new approach, call the Earliest Feasible Deadline First *(EFDF)* which is used to reduce the time complexity of earliest deadline first algorithm by some assumptions.

II. Background and Review of Related Works

Each processor in a uniform multiprocessor machine is characterized by a speed or Computing capacity, with the interpretation that a job executing on a processor with speed ***s*** for ***t*** time units completes (**s * t**) units of execution. The Earliest-Deadline First scheduling of real-time systems upon uniform multiprocessor machines is considered. It is known that online algorithms tend to perform very poorly in scheduling such real-time systems on multiprocessors; resource-augmentation techniques are presented here that permit online algorithms in general (*EDF* in particular) to perform better than may be expected given these inherent limitations.

Generalization the definition of utilization from periodic task to nonperiodic tasks has been studies in [23] and [24]. In deriving the utilization bound for rate monotonic scheduler with multiframe and general real time task models, Mok and Chen in [25] and [26] proposed a *maximum average utilization* which measures utilization in an infinite measuring window. To derive the utilization bound for nonperiodic tasks and multiprocessor system, the authors in [23] and [24] proposed a utilization definition that is based on relative deadlines of tasks, instead of periods. It is shown that *EDF* scheduling upon uniform multiprocessors is robust with respect to both job execution requirements and processor computing capacity.

III. Scheduling on Multiprocessor System

Meeting the deadlines of a real-time task set in a multiprocessor system requires a scheduling algorithm that determines, for each task in the system, in which processor they must be executed (*allocation problem*), and when and in which order, with respect to other tasks, they must start their execution (*scheduling problem*). This is a problem with a difficult solution, because (i) some research results for a single processor not always can be applied for multiple processors [17], [18], (ii) in multiple processors different scheduling anomalies appear [19], [21], [20] and (iii) the solution to the allocation problem requires of algorithms with a high computational complexity.

The scheduling of real-time tasks on multiprocessors can be carried out under the *partitioning scheme* or under the *global scheme*. In the partitioning scheme (Figure 1.a) all the instances (or jobs) of a task are executed on the same processor. In contrast, in the global scheme (Figure 1.b), a task can migrate from one processor to another during the execution of different instances. Also, an individual job of a task that is preempted from some processor, may resume execution in a different processor. Nevertheless, in both schemes parallelism is prohibited, that is, no job of any task can be executed at the same time on more than one processor.

On both schemes, the *admission control* mechanism not only decides which tasks must be accepted, but also it must create a feasible allocation of tasks to processors (i.e., on each

processor, all tasks allocated must met their deadlines). For the partitioning and global schemes, task sets can be scheduled using static or dynamic schedulers. In any case, the computational complexity associated to the admission control must remain as low as possible, especially for the dynamic case.

The partitioning scheme has received greater attention than the global scheme, mainly because the scheduling problem can be reduced to the scheduled on single processors, where at the moment a great variety of scheduling algorithms exist. It has been proved by Leung and Whitehead [18] that the partitioned and global approaches to static-priority scheduling on identical multiprocessors are *incomparable* in the sense that (i) there are task sets that are feasible on μ identical processors under the partitioned approach but for which no priority assignment exists which would cause all jobs of all tasks to meet their deadlines under global scheduling on the same μ processors, and (ii) there are task sets that are feasible on μ identical processors under the global approach, which cannot be partitioned into μ distinct subsets such that each individual partition is feasible on a single static-priority uniprocessor.

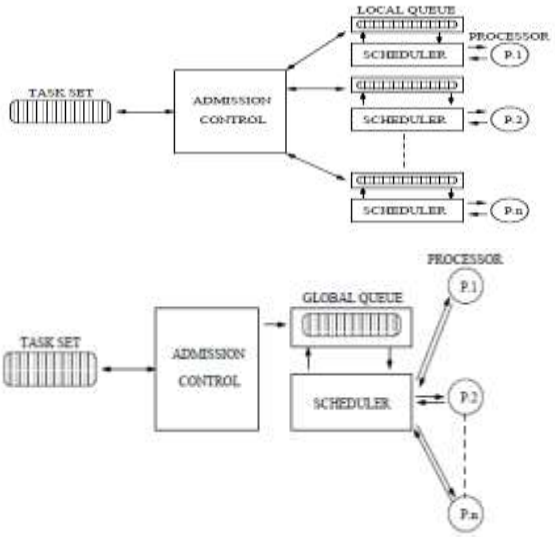

Fig. 1. (a). Partitioning and (b). Global Scheduling Schemes

IV. Our Proposed Grid Approximation Strategy

We have applied some modification to the global Earliest Deadline First algorithms to decrease the number of task migration and also to add predictability to its behavior. In order to decrease the number of migrations we prevent a job from moving to another processor if it is among the ***m*** higher priority jobs. The scheduling algorithms can be classified in static and dynamic. In a static scheduling algorithm, all scheduling decisions are provided a priori. Given a set of timing constraints and a schedulability test, a table is constructed, using one of many possible techniques (e.g., using various search techniques), to identify the start and completion times of each task, such that no task misses their deadlines. This is a highly predictable approach, but it is static in the sense that when the characteristics of the task set change the system must be re-started and its scheduling table re-computed.

In a dynamic scheduling algorithm, the scheduling decision is executed at run-time based on task's priorities. The dynamic scheduling algorithms can be classified in algorithms with *fixed* priorities and algorithms with *variable* priorities. In the scheduling algorithms with fixed priorities, the priority of each task of the system remains static during the complete execution of the system, whereas in an algorithm with variable priorities the priority of a task is allowed to change at any moment.

The schedulability test in static scheduling algorithms can only be performed *off-line*, but in dynamic scheduling algorithms it can be performed off-line or *on-line*. In the o®-line scheduling test, there are complete knowledge of the set of tasks executing in the system, as well as the restrictions imposed to each one of the tasks (deadlines, precedence restrictions, execution times), before the start of their execution. Therefore no new tasks are allowed to arrive in the system. Therefore, a job will continue its execution on the same processor if possible (***processor affinity***[1]).

A. The Strategy

In Earliest Deadline First scheduling, at every scheduling point the task having the shortest deadline is taken up for scheduling. The basic principle of this algorithm is very intuitive and simple to understand. The schedulability test for Earliest Deadline First is also simple. A task is schedule under *EDF*, if and only if it satisfies the condition that total processor utilization due to the task set is less than 1. For a set of periodic real-time task $\{T1, T_2, T_n\}$, *EDF* schedulibility criterion can be expressed as:-

$$\sum_{i=1}^{n} \frac{e_i}{p_i} = \sum_{i=1}^{n} U_i \leq 1$$

Where $\boldsymbol{e_i}$ is the execution time, $\boldsymbol{p_i}$ is the priority of task and $\mathbf{u_i}$ is the average utilization due to the task $\boldsymbol{T_i}$ and $\boldsymbol{n}$ is the total number of task in set. *EDF* has been proven to be an optimal uniprocessor scheduling algorithm [8]. This means that if a set of task is unschedulable under Earliest Deadline First , then no other scheduling algorithm can feasible schedule this task set. In the simple schedulability test for *EDF* we assumed that the period of each task is the same as its deadline. However in practical problem the period of a task may at times be different from its deadline. In such cases, the schedulability test needs to be changed. If $\boldsymbol{p_i > d_i}$, then each task needs $\boldsymbol{e_{we\ are}}$ount of computing time every min($\boldsymbol{p_i}$, $\boldsymbol{d_i}$) duration time. Therefore we can write:

$$\sum_{i=1}^{n} \frac{e_i}{\min(p_i, d_i)} \leq 1$$

However, if $\boldsymbol{p_i < d_i}$, it is possible that a set of tasks is *EDF* schedulable , even when the task set fail to meet according to expression

B. Mathematical Representation

Our motivation for exploiting ***processor affinity*** drive from the observation that, for much parallel application, time spent bringing data into the local memory or cache is significant source of overhead, ranging between *30%* to *60%* of the total execution time [3]. While migration is unavoidable in the

global schemes, it is possible to minimize migration caused by a poor assignment of task to processors.

By scheduling task on the processor whose local memory or cache already contains the necessary data, we can significantly reduce the execution time and thus overhead the system. It is worth mentioning that still a job might migrate to another processor when there are two or more jobs that were last executed on the same processor. A migration might also happen when the numbers of ready jobs become less than the number processors. This fact means that our proposed algorithm is a work conserving one.

In order to give the scheduler a more predictable behavior we first perform a ***feasibility check*** to see whether a job has a chance to meet its deadline by using some exiting algorithm like Yao's [16]. If so, the job is allowed to get executed. Having known the deadline of a task and its remaining execution time it is possible to verify whether it has the opportunity to meet its dead line. More precisely, this verification can be done by examining a *task's laxity*[3]. The *laxity* of a real-time task T_i at time t, $L_i(t)$, is defined as follows:-

$$L_i(t) = D_i(t) - E_i(t)$$

Where $D_i(t)$ is the dead line by which the task T_i must be completed and $E_i(t)$ is the amount of computation remaining to be performed. In other words, Laxity is a measure of the available flexibility for scheduling a task. A laxity of $L_i(t)$ means that if a task T_i is delayed at most by $L_i(t)$ time units, it will still has the opportunity to meet its deadline.

A task with zero laxity must be scheduled right away and executed without preemption or it will fail to meet its deadline. A negative laxity indicates that the task will miss the deadline, no matter when it is possible picked up for execution. We call this novel approach the ***Earliest Feasible Deadline First*** *(EFDF)*

C. EFDF Scheduling Algorithm

Let m denote the number of processing nodes and n, ($n \geq m$) denote the number of Available tasks in a uniform parallel real-time system. Let $s_1, s_2, \ldots s_m$ denote the computing capacity of available processing nodes indexed in a non-increasing manner: $s_j \geq s_j + 1$ for all j, $1<j<m$. We assume that all speeds are positive i.e. $s_j > 0$ for all j. In this section we are presenting five steps of *EFDF* algorithm. Obviously, each task which is picked for up execution is not considered for execution by other processors. Here we are giving following methods for our new approach:

1. Perform a feasibility check to specify the task which has a chance to meet their deadline and put them into a set A, Put the remaining tasks into set B. We can partition the task set by any existing approach.
2. Sort both task sets A and B according to their deadline in a non-descending order by using any of existing sorting algorithms. Let k denote the number of tasks in set A, i.e. the number of tasks that have the opportunity to meet their deadline.
3. For all processor j, ($j \leq min(k,m)$) check whether a task which was last running on the j^{th} processor is among the first min(k,m) tasks of set A. If so assign it to the j^{th} processor. At this point there might be some processors to which no task has been assigned yet.
4. For all j, ($j \leq min(k,m)$) if no task is assigned to the j^{th} processor , select the task with earliest deadline from remaining tasks of set A and assign it to the j^{th} processor. If $k \geq m$, each processor have a task to process and the algorithm is finished.
5. If $k<m$, for all j, ($k<j \leq m$) assign the task with smallest deadline from B to the j^{th} processor. The last step is optional and all the tasks from B will miss their deadlines.

D. Experimental Evaluation

We conducted simulation-based experimental studies to validate our analytical results on *EFDF* overhead. We consider an SMP machine with four processors. We consider four tasks running on the system. Their execution times and periods are given in Table 2. The total utilization is approximately 1.5, which is less than 4, the capacity of processors. Therefore, LLREF can schedule all tasks to meet their deadlines. Note that this task set's α (i.e., $\max^N \{u_i\}$) is 0.818, but it does not affect the performance of *EFDF*, as opposed to that of global EDF [22].

Table 2: Task Parameters (4 Task Set)

Process P_i	Execution Time C_i	Period T_i	U_i
P1	9	11	0.818
P2	5	25	0.2
P3	3	30	0.1
P4	5	14	0.357

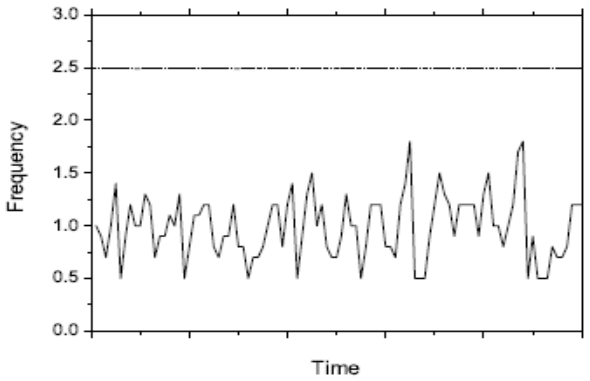

Figure 1: Scheduler Invocation Frequency with 4 Tasks

In Figure 1, the upper-bound on the scheduler invocation frequency and the measured frequency are shown as a dotted

line and a fluctuating line, respectively. We observe that the actual measured frequency respects the upper bound.

Table 3: Task Parameters (8 Task Set)

Process P_i	Execution Time C_i	Period T_i	U_i
P1	3	7	0.429
P2	1	16	0.063
P3	5	19	0.263
P4	4	5	0.8
P5	2	26	0.077
P6	15	26	0.577
P7	20	29	0.69
P8	14	17	0.824

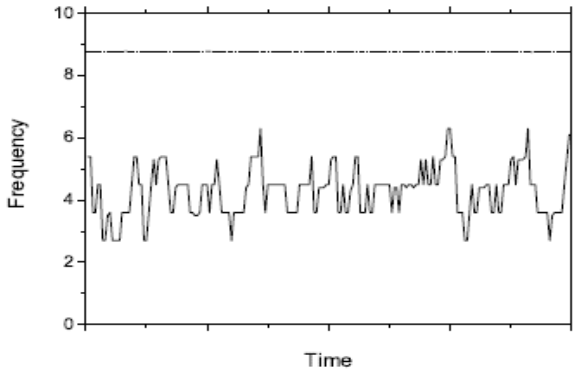

Figure 2: Scheduler Invocation Frequency with 8 Tasks

Figure 2 shows the upper-bound on the invocation frequency and the actual frequency for the 8-task set. Consistently with the previous case, the actual frequency never moves beyond the upper-bound. We also observe that the average invocation frequencies of the two cases are approximately 1.0 and 4.0, respectively. As expected the number of tasks proportionally affects E*FDF* overhead.

E. Complexity and Performance of the Partitioning Algorithms

In Table 2 we are taking the compression of given standard and simulated complexities of different algorithms given below and we are comparing these complexities to our purposed algorithm, the complexity and performance of the partitioning algorithms is introduced. Note that the algorithms with lowest complexity are RMNF-L&L, RMGT/M, and EDF-NF, while the algorithm with highest complexity is RBOUND-MP. The rest of the algorithms have complexity $O(n \log n)$. The algorithms with best theoretical performance are RM-FFDU, RMST, RMGT, RMGT/M, EDF-FF and EDF-BF.[16]

TABLE 2 :COMPLEXITY AND PERFORMANCE OF THE MULTIPROCESSOR PARTITIONING ALGORITHMS

Algorithm	Condition	Complexity	$\Re_A$
RMNF [14]	IP	$O(n \log n)$	2.67
RMFF [35]	IP	$O(n \log n)$	2.33
RMBF [35]	IP	$O(n \log n)$	2.33
RM-FFDU [33]	UO	$O(n \log n)$	5/3
FFDUF [12]	L&L	$O(n \log n)$	2.0
RMST [10]	PO	$O(n \log n)$	$1/1\text{-}\alpha$
RMGT [10]	PO and Le	$O(n \log n)$	7/4
RBOUND-MP [21]	RBOUND	$O(n(m + \log n))$	N/A
RMNF-L&L [34]	L&L	$O(n)$	2.88
RMFF-L&L [34]	L&L	$O(n \log n)$	2.33
RMBF-L&L [34]	L&L	$O(n \log n)$	2.33
RMGT/M [9]	PO	$O(n)$	$\frac{1}{(1-\alpha)} + \frac{1}{1-(\ln 2)/M}$
EDF-FF [15]	$U \leq 1$	$O(n \log n)$	1.7
EDF-BF [15]	$U \leq 1$	$O(n \log n)$	1.7
EDF-WF [31]	$U \leq 1$	$O(n \log n)$	N/A
EDF-NF [31]	$U \leq 1$	$O(n)$	N/A

F. Complexity Analysis

The Earliest Deadline First algorithm would be maintaining all tasks that are ready for execution in a queue. Any freshly arriving task would be inserted at the end of queue. Each task insertion will be achieved in *O(1)* or constant time, but task selection (to run next) and its deletion would require *O(n)* time, where n is the number of tasks in the queue. *EDF* simply maintaining all ready tasks in a sorted priority queue that will be used a heap data structure. When a task arrives, a record for it can be inserted into the heap in $O(\log_2 n)$ time where ***n*** is the total number of tasks in the priority queue. Therefore, the time complexity of Earliest Deadline First is equal to that of a typical sorting algorithm which is $O(n \log_2 n)$. While in the *EFDF* the number of distinct deadlines that tasks is an application can have are restricted.

In our approach, whenever a task arrives, its absolute deadline is computed from its release time and its relative deadline. A separate first in first out (*FIFO*) queue is maintained for each distinct relative deadline that task can have. The schedulers insert a newly arrived task at the end of the corresponding relative deadline queue. So tasks in each queue are ordered according to their absolute deadlines. To find a task with the earliest absolute deadline, the scheduler needs to search among the threads of all FIFO queues. If the number of priority queue maintained by the scheduler in ***n***, then the order of searching would be *O(1).* The time to insert a task would also be *O(1)*. So finally the time complexity of five steps of Earliest Feasible Deadline First *(EFDF)* are *O(n)*, $O(n \log_2 n)$, *O(m)*, *O(m)*, *O(m)*, respectively.

V. CONCLUSION AND FUTURE WORK

This work focused on some modification to the global Earliest Deadline First algorithms to decrease the number of task migration and also to add predictability to its behavior. Mainly Earliest Feasible Deadline First algorithms are presented the least complexity according to their performance analyzed. Experimental result of Earliest Feasible Deadline First *(EFDF)* algorithm reduced the time complexity in compression of Earliest Deadline First algorithm on real time system scheduling for multiprocessor system and perform the feasibility checks to specify the task which has a chance to meet their deadline.

When Earliest Feasible Deadline First is used to schedule a set of real-time tasks, unacceptable high overheads might have to be incurred to support resource sharing among the tasks without making tasks to miss their respective deadlines, due to this it will take again more time. Our future research will investigate other less complexity Algorithm and also reduced the overhead for different priority assignments for global scheduling which will, consequently, lead to different bounds.

We believe that such studies should be conducted regularly by collecting data continuously so that skill demand patterns can be understood properly. This understanding can lead to informed curricula design that can prepare graduates equipped

with necessary skills for employment. Once such studies are carried out, students can use the findings to select courses that focus on those skills which are in demand. Academic institutions can use the findings so that those skills in demand can be taken into account during curriculum design.

As an advance to our work, in future, we have desire to work on different deployment approaches by developing more strong and innovative algorithms to solve the time complexity of Earliest Deadline First. Moreover, as our proposed algorithm is a generalized one, we have planned to expand our idea in the field of Real Time System existing Rate Monotonic Algorithm for calculating minimum Time Complexity. Moreover, we have aim to explore some more methodologies to implement the concept of this paper in real world and also explore for Fault Tolerance Task Scheduling Algorithms to finding the Task Dependency in single processor or multiprocessor system for reducing the time for fault also reduce the risk for fault and damage.

ACKNOWLEDGMENTS

The authors thank the reviewers of drafts of this paper. It is profound gratitude and immense regard that we acknowledge to Dr. S.P. Panda, Chairman, GGI, Prof. N.V.J. Rao Dean (Admin), GGI for their confidence, support and blessing without which none of this would have been possible. Also a note to all professors here in GIET for the wisdom and knowledge that they given us, all of which came together in the making of this paper. We express our gratitude to all my friends and colleagues as well for all their help and guidance.

REFERENCES

[1] S. Baruah, S. Funk, and J. Goossens , "Robustness Results Concerning *EDF* Scheduling upon Uniform Multiprocessors",IEEE Transcation on computers, Vol. 52, No.9 pp. 1185-1195 September 2003.

[2] E.P.Markatos, and T.J. LeBlanc, " *Load Balancing versus Locality Management in Shared-Memory Multiprocessors*", *The 1992 International Conference on Parallel Processing*, August 1992.

[3] S. Lauzac, R. Melhem, and D. Mosses,"*Compression of Global and Partitioning Scheme for Scheduling Rate Monotonic Task on a Multiprocessor*", *The 10th EUROMICRO Workshop on Real-Time Systems, Berlin*,pp.188-195, June 17-18, 1998.

[4] Vahid Salmani ,Mohsen Kahani , " *Deadline Scheduling with Processor Affinity and Feasibility Check on Uniform Parpllel Machines*", Seventh International Conference on Computer and Information Technology, *CIT.121.IEEE*,2007.

[5] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem. Operations Research", 26(1):127–140, 1978.

[6] Y. Oh and S. Son. "*Allocating fixed-priority periodic tasks on multiprocessor systems*", Real-Time Systems Journal, 9:207–239, 1995.

[7] J. Lehoczky, L. Sha, and Y. Ding. "*The rate monotonic Scheduling: Exact characterization and average case behavior*", *IEEE Real-time Systems Symposium*, pages 166–171, 1989.

[8] C.M. Krishna and Shin K.G. Real-Time Systems. Tata McGrawiHill,1997.

[9] S. Chakraborty, S. Künzli, L. Thiele. *Approximate Schedulability Analysis*. 23rd IEEE Real-Time Systems Symposium (RTSS), IEEE Press, 159-168, 2002.

[10] J.A. Stankovic, M. Spuri, K. Ramamritham, G.C. Buttazzo. *Deadline Scheduling for Real-Time Systems EDF and Related Algorithms*. Kluwer Academic Publishers, 1998.

[11] S. Baruah, D. Chen, S. Gorinsky, A. Mok. *Generalized Multiframe Tasks.* The International Journal of Time-Critical Computing Systems, 17, 5-22, 1999.

[12] S. Baruah, A. Mok, L. Rosier. *Preemptive Scheduling Hard-Real-Time Sporadic Tasks on One Processor*. Proceedings of the Real- Time Systems Symposium, 182-190, 1990.

[13] K. Gresser. *Echtzeitnachweis Ereignisgesteuerter Realzeitsysteme*. Dissertation (in german), VDI Verlag, Düsseldorf, 10(286), 1993.

[14] M. Devi. *An Improved Schedulability Test for Uniprocessor Periodic Task Systems*. Proceedings of the 15th Euromicro Conference on Real-Time Systems, 2003.

[15] C. Liu, J. Layland. Scheduling Algorithms for Multiprogramming in Hard Real-Time Environments. Journal of the ACM, 20(1), 46-61, 1973

[16] Omar U. Pereira Zapata, Pedro Mej´ıa Alvarez "EDF and RM Multiprocessor Scheduling Algorithms: Survey and Performance Evaluation" Report No. CINVESTAV-CS-RTG-02. CINVESTAV-IPN, Sección de Computación.

[17] S. K. Dhall and C. L. Liu, "On a Real-Time Scheduling Problem", *Operation Research*, vol. 26, number 1, pp. 127-140, 1978.

[18] J. Y.-T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks, *Performance Evaluation*, number 2, pp. 237-250,1982.

[19] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies",*SLAM* Journal of *Applied Mathematics*, 416-429, 1969.

[20] R. Ha and J. Liu, "Validating Timing Constraints in Multiprocessor and Distributed Real-Time Systems", *Int'l Conf. on Distributed Computing system*, pp. 162-171, June 21-24, 1994.

[21] B. Andersson, "Static Priority Scheduling in Multiprocessors", PhD *Thesis*, Department of Comp.Eng., Chalmers University, 2003.

[22] J. Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen,"An Optimal Real-Time Scheduling Algorithm for Multiprocessors", IEEE Conference Proceedings, SIES 2007: 9-16.

[23] B. Anderson,"Static-priority scheduling on multiprocessors," PhD dissertation, Dept. of Computer eng., Chalmers Univ. of Technology,2003.

[24] T.Abdelzaher and C.Lu,"Schedubility Analysis and Utilization Bound of highly Scalable Real-Time Services," *Proc. 15th Euro-micro Conf. Real Time Systems*,pp.141-150,july 2003.

[25] A.K.Moc and D.Chen," A General Model for Real Time Tasks," Technical Report TR-96-24,Dept. of Computer Sciences, Univ.of Texas at Austin, Oct.1996.

[26] A.K.Moc and D.Chen," A multiframe Model for Real Time Tasks,"*IEEE Trans. Software Eng*., vol. 23 ,no.10,pp.635-645,Oct 1997.

AUTHORS PROFILE

Jagbeer Singh has received a bachelor's degree in Computer Science and engineering, from the Dr. B.R.A. University Agra 2000, Uttar Pradesh (India). In 2006, he received a master's degree in computer science from the Gandhi Institute of Engineering and Technology Gunupur , under Biju Patnaik University of Technology Rourkela ,Orissa(India) He has been a

Asst. Professor Gandhi Institute of Engineering and Technology Gunupur in the Department of Computer Science since 2004. His research interests are in the areas of Real Time Systems under the topics "Fault Tolerance Tasks Scheduling in single processor or multiprocessor system," he has published 3 peer-reviewed, and 6 scientific papers, organized 5 national research papers in international/ national conferences and organized national conferences /workshops, and serves as a reviewer for 3 journals, conferences, workshops, and also having membership for different professional bodies like ISTE,CSI,IAENG etc.

Bichitrananda Patra He is an assistant professor at the Department of Information Technology Engineering, Gandhi Institute of Engineering Technology, Gunupur, Orissa, India, He received his master degree in Physics and Computer Science from the Utkal University, Bhubaneswar, Orissa, India. His research interests are in Soft Computing, Algorithm analysis, statistical, neural nets. He has published 8 research papers in international journals and conferences organized national workshops and conference and also having membership for different professional bodies like ISTE, CSI etc.

Satyendra Prasad Singh having M. Sc., MCA and Ph. D. in Statistics and working as a Professor and Head of department of MCA, Gandhi Institute of Computer Studies, Gunupur, Rayagada, Orissa, India since 2007. He has worked as a Research Associate in Defence Research and Development Organisation, Ministry of Defence, Government of India, New Delhi for 2 years and also worked in different universities. Also he has received a Young Scientist Award in year 2001 by International Academy of Physical Sciences for the best Research Paper in CONIAPS-IV, 2001. He has published more than 10 papers in reputed International/National journals and presented 15 Papers in International/National Conferences in the field of Reliability Engineering, Cryptology and Pattern Recognization. He has guided many M. Tech and MCA project thesis.