Variants of Zero-Laxity based Scheduling Algorithm for Hard Real-Time Systems

Pulkit Agrawal
Electronics and Electrical Department
BITS Pilani University
Goa, India
f20150523@goa.bits-pilani.ac.in

***Abstract*—In this paper, we propose three scheduling algorithm namely, Earliest Deadline first till Zero-Laxity (EDZL), Fixed Priority till Zero-Laxity, and Least Laxity Group First (LLGF). Hard Real-Time Systems are all about meeting deadlines. We need various algorithms to make sure that all the tasks meet their deadlines. The algorithms proposed are improved version of well-known pre-existing algorithms. These algorithms are modified versions of the parent algorithm. Throughout the paper we considers global multiprocessor scheduling only.**

The proposed algorithms are written using ADA Language in a Low-Level Program. This is done so as to run simulation in Cheddar. Since we are writing a low-level program, hence, further information about the system is required so as to implement the algorithms. All the required information for implementation of the algorithm are discussed in this paper.

***Index Terms*—Laxity; absolute deadline; dynamic priority; multi-processor.**

I. Introduction

In this paper, all the proposed algorithms are implemented on a multiple identical processor. Approaches to multiprocessor real-time scheduling, can be categorised into two broad classes: partitioned and global. Partitioned approaches allocate each task to a single processor, dividing the multiprocessor scheduling problem into one of task allocation followed by uniprocessor scheduling. In contrast, global approaches allow tasks to migrate from one processor to another at run-time. All the algorithms proposed in this paper consider global multiprocessor scheduling. The main motive of this paper is to propose new scheduling algorithms which are derived from well-known scheduling algorithms. The new scheduling algorithms are supposed to perform better than the existing algorithms. Since these are considered for Hard Real-Time systems, hence meeting the deadlines is of the highest concern.

The new algorithms are proposed using low level language, hence there are few assumptions involved which are stated in respective sections of all the algorithms. In this paper, we have also considered special task sets as an example which performs better in new algorithms proposed than the existing algorithms. The performance of the new algorithms can be compared in terms of meeting the deadlines, number of pre-emptions and number of context switching.

The following paper consist of four more sections. Each sections takes up one new scheduling algorithm and each section consists of the overview of the scheduling algorithm, the guidelines and assumptions for implementation of algorithm, and an example task sets. Section I discuss about Earliest Deadline First till Zero-Laxity (EDZL), Section II discuss about Fixed Priority till Zero-Laxity (FPZL) and Section III discuss about Least Laxity Group First (LLGF) followed by Section IV Conclusion.

II. Earliest Deadline till Zero-Laxity (EDZL)

Scheduling algorithm proposed in [1] is a hybrid scheduling algorithm which is based on the guidelines of Earliest Deadline First. This algorithm gives dynamic priorities to each task sets according to their absolute deadline. The variation in proposed algorithm is that whenever a job in a task reaches a state of zero laxity (i.e. slack time equal to zero) this algorithm gives that corresponding task the highest priority. So the decreasing order of priorities of the jobs (present in the ready queue) at any instance are, jobs with zero slack time, then, jobs with earliest deadline. It is a dynamic priority scheduling algorithm.

A. Guidelines and Assumptions in EDZL

EDZL algorithm proposed in [1] includes various guidelines:

- The scheduling algorithm is pre-emptive, meaning, at any unit of time a higher priority task can pre-empt the lower priority task.
- There is no specific tie-breaker rule for jobs with same priorities, if there are more than one job with same priority any jobs among them can execute.
- Parallel execution of the jobs are not allowed, i.e., if a job starts its execution in any one processor then it cannot simultaneously start its execution in another processor.
- Job-Level Migration is allowed within multiple processor of the system.

EDZL is a high level scheduling algorithm. For performing simulation in Cheddar, this algorithm is implemented using a low level language. In order to simulate Multi-Processor environment, we take Uniprocessor Multi-Core System with identical cores Thus, for implementation of algorithm there are few assumptions made, which are as follows:

- We know all the temporal parameters of the task sets. Temporal parameters including release time of the jobs, worst case execution time of the jobs, and period of the jobs (if periodic tasks are included in the task sets). We

require these values for the calculation of the slack time and absolute deadlines of all the tasks.
- We consider only schedulable task sets.
- Negative slack time is not possible. This assumptions states that no task will miss its deadline. This assumption is a consequence of previous assumption. This assumption also states that if there are *'m'* identical processors then, not more than *'m'* tasks can simultaneously reach zero laxity state.
- Since, this algorithm is designed for multiple processors, we assume that we know the total number of processors involved in performing the simulation.
- Although the cores are completely identical, but the instances of scheduling jobs with distinct priority, higher priority tasks are scheduled is first core being considered for scheduling and so on.

B. Comparison EDZL versus EDF versus LLF

The taskset used for the comparison is given in Table I. We use this taskset for analysis of its performance in the proposed algorithm in [1], LLF and EDF. We have taken the number of cores as two. We can clearly see in Figure 1 that for this taskset, EDZL outperforms both EDF (Figure 3) and LLF (Figure 2).

TABLE I
TASKSET PARAMETERS

	p	e	d
T1	15	11	15
T2	6	3	6
T3	5	3	5

We can clearly see in Figure 1, at $t = 7units$, laxity of the Task 1.1 becomes equal to zero, and hence it pre-empts Task 3.2 and Task 1.1 gets scheduled in processor $c2/p1$. Since the absolute deadline of Task 3.2 is $10units$ and absolute deadline of Task 2.2 is $12units$, Task 3.2 gets higher priority as compared to Task 2.2 and it further pre-empts and Task 3.2 is scheduled in processor $c1/p2$. Again at $t = 27units$ we see that Task 1.2 pre-empts Task 3.6 and is scheduled in processor $c2/p1$ (based on the last assumption stated). Rest of the scheduling is done, based on Earliest Deadline First algorithm.

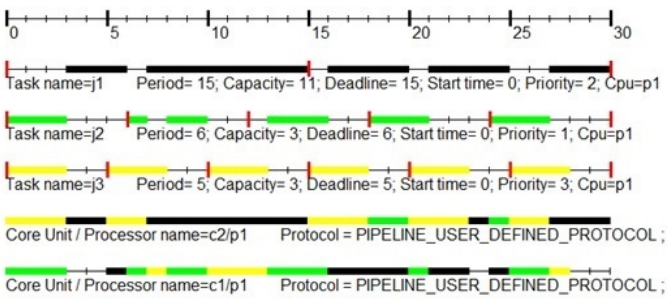

Fig. 1. EDZL Scheduling

We can clearly see in Figure 2, that two jobs in Task 2 miss their deadline, and hence we can say that EDZL performs better that LLF in this particular example taskset. We can also see that along with missing deadline number of context switching and pre-emption are more than that in EDZL.

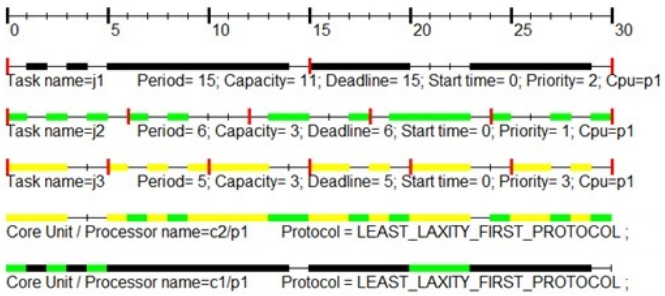

Fig. 2. LLF Scheduling

We can clearly see in Figure 3, that EDZL is superior to EDF scheduling as Task 1.1 missed its deadline. If there is no case of zero laxity, then both algorithms perform exactly same, but when there is a case of zero laxity EDZL performs better.

Fig. 3. EDF Scheduling

III. FIXED PRIORITY TILL ZERO LAXITY (FPZL)

Scheduling algorithm proposed in [2] is a modified scheduling algorithm which schedules the jobs according to the fixed priority assigned to them. The modification in this algorithms is that when any job reaches the zero laxity this algorithm assigns the highest priority to that corresponding task and schedules it immediately. FPZL is a dynamic priority based scheduling algorithm. Also this is a minimally dynamic algorithm as the priority of any task can change at most one time (when zero laxity state come in). Hence this algorithm reduces number of pre-emptions and context switching. If any system has a high overhead associated in pre-emption then, this algorithm can be preferred.

A. Guidelines and Assumptions in FPZL

FPZL algorithm proposed in [2] includes various guidelines:

- The fixed priority of various jobs are assigned by the user, and there is no restriction on deciding the fixed priority ordering.

- All the priorities assigned to various tasks are distinct. No two tasks can have same priority.
- The scheduling algorithm is pre-emptive, meaning, at any unit of time a higher priority task can pre-empt the lower priority task.
- Parallel execution of the jobs are not allowed, i.e., if a job starts its execution in any one processor then it cannot simultaneously start its execution in another processor.
- Job-Level Migration is allowed within multiple processor of the system.

FPZL is a high level scheduling algorithm. For performing simulation in Cheddar, this algorithm is implemented using a low level language. In order to simulate Multi-Processor environment, we take Uniprocessor Multi-Core System with identical cores Thus, for implementation of algorithm there are few assumptions made, which are as follows:

- We know all the temporal parameters of the task sets. Temporal parameters including release time of the jobs, worst case execution time of the jobs, and period of the jobs (if periodic tasks are included in the task sets). We require these values for the calculation of the slack time of all the tasks.
- We consider only schedulable task sets.
- Negative slack time is not possible. This assumptions states that no task will miss its deadline. This assumption is a consequence of previous assumption. This assumption also states that if there are *'m'* identical processors then, not more than *'m'* tasks can simultaneously reach zero laxity state.
- Since, this algorithm is designed for multiple processors, we assume that we know the total number of processors involved in performing the simulation.
- Although the cores are completely identical, but the instances of scheduling jobs with distinct priority, higher priority tasks are scheduled is first core being considered for scheduling and so on.

B. Comparison FPZL versus FP versus LLF

The taskset used for the comparison is given in Table II. We use this taskset for analysis of its performance in the proposed algorithm in [2], LLF and FP. We have taken the number of cores as two. We can clearly see in Figure 4 that for this taskset, FPZL outperforms both FP (Figure 5) and LLF (Figure 6).

TABLE II
TASKSET PARAMETERS

	p	e	d	π
T1	15	7	15	3
T2	6	3	6	2
T3	6	4	6	1

We can clearly see in Figure 4, at $t = 2units$ Task 3.1 reaches zero laxity state and hence it is given highest priority and gets scheduled in processor $c2/p1$ and Task 1.1 pre-empts Task 2.1 in processor $c1/p1$. Again at $t = 5units$ Task 2.1 reaches zero laxity state and is scheduled in processor $c2/p1$. Again at $t = 8units$ Task 3.2 reaches zero laxity state and hence scheduled in processor $c2/p1$. Similarly, at $t = 20units$ Task 3.4 reaches zero laxity state. Rest scheduling is done on the basis of fixed priority in which 3 being highest priority.

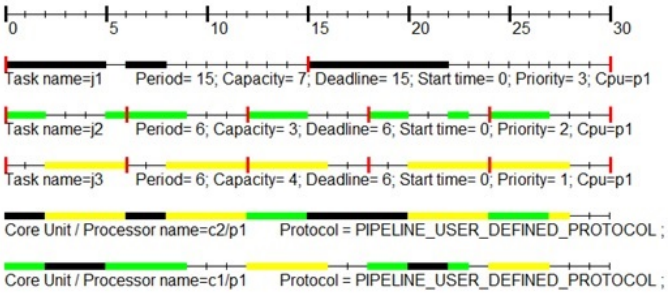

Fig. 4. FPZL Scheduling

We can clearly see in Figure 5, that Task 3 as a whole misses their deadlines in FP scheduling, and hence FPZL performs better that FP scheduling.

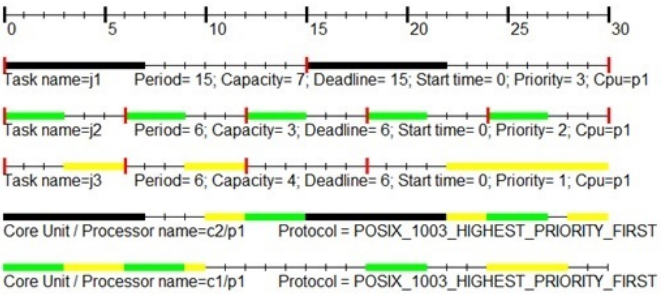

Fig. 5. FP Scheduling

We can clearly see in Figure 6, that Task 1.1 misses its deadline, hence FPZL performs better than LLF scheduling.

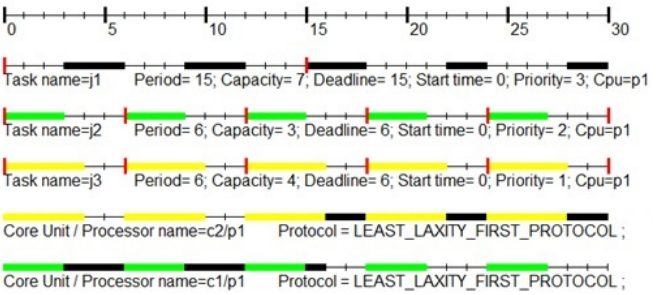

Fig. 6. LLF Scheduling

IV. LEAST LAXITY GROUP FIRST (LLGF)

In [3], a variant of Least Laxity First (LLF) scheduling is proposed, which they call it as "Least Laxity Group First" (LLGF). This is especially designed so as to decrease the

number of pre-emptions that occur when LLF algorithm is used for scheduling the tasks in a Real-Time system. We can form a laxity group Π, where each laxity group has a integer value assigned as a priority. The value of Π can be calculated by some function $f(L)$ where L denotes the laxity of the tasks. The function implemented in this paperwork is:

$$\Pi = f(L) = \lceil \frac{L}{\alpha} \rceil \tag{1}$$

where α represents the laxity group size.

A. Guidelines and Assumptions in LLGF

LLGF algorithm proposed in [3] includes various guidelines:

- The scheduling algorithm is pre-emptive, meaning, at any unit of time a higher priority task can pre-empt the lower priority task.
- There is no specific tie-breaker rule for jobs with same priorities, if there are more than one job with same priority any jobs among them can execute.
- Parallel execution of the jobs are not allowed, i.e., if a job starts its execution in any one processor then it cannot simultaneously start its execution in another processor.
- Job-Level Migration is allowed within multiple processor of the system.

LLGF is a high level scheduling algorithm. For performing simulation in Cheddar, this algorithm is implemented using a low level language. In order to simulate Multi-Processor environment, we take Uniprocessor Multi-Core System with identical cores Thus, for implementation of algorithm there are few assumptions made, which are as follows:

- The function $f(L)$ required for forming the Laxity groups is known.
- The value α used in $f(L)$ is known and is fixed.
- We know all the temporal parameters of the task sets. Temporal parameters including release time of the jobs, worst case execution time of the jobs, and period of the jobs (if periodic tasks are included in the task sets). We require these values for the calculation of the slack time of all the tasks.
- We consider only schedulable task sets.
- Negative slack time is not possible. This assumptions states that no task will miss its deadline. This assumption is a consequence of previous assumption. This assumption also states that if there are *'m'* identical processors then, not more than *'m'* tasks can simultaneously reach zero laxity state.
- Since, this algorithm is designed for multiple processors, we assume that we know the total number of processors involved in performing the simulation.
- Although the cores are completely identical, but the instances of scheduling jobs with distinct priority, higher priority tasks are scheduled is first core being considered for scheduling and so on.

B. LLGF Simulation Interpretation

The taskset used for the comparison is given in Table III. We use this taskset for analysis of its performance in the proposed algorithm in [2] and LLF . We have taken the number of cores as two. We can clearly see in Figure 7 and 8 that for this taskset, LLGF outperforms LLF (Figure 8). We have taken the values of α as 2 for the calculation of the laxity groups.

TABLE III
TASKSET PARAMETERS

	p	e	d
T1	15	7	15
T2	6	3	6
T3	6	4	6

We can clearly see in Figure 7 and Figure 8that LLGF gives better schedule than LLF as in LLF Task 1.1 missed its deadline.

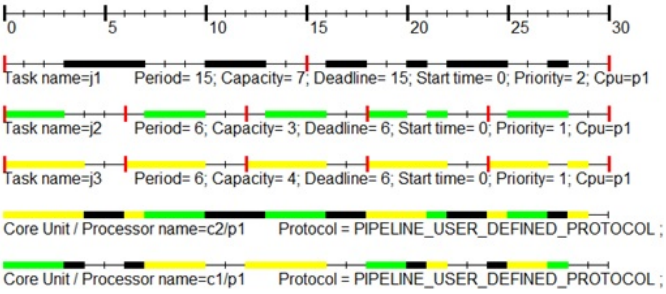

Fig. 7. LLGF Scheduling

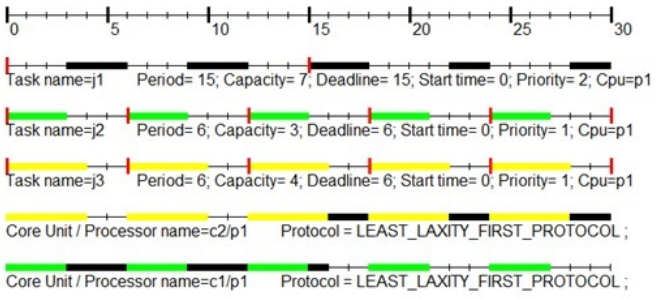

Fig. 8. LLF Scheduling

V. CONCLUSION

We can clearly see that the algorithms proposed in [1]–[3] can be implemented on Multiprocessors Hard Real-Time Systems. We saw task sets where EDZL performed better that LLF and EDL Algorithms. Also, we saw that FPZL performed better that FP and LLF algorithm. We even saw that in EDZL, all tasks met their deadlines and also the number of context switches and number of pre-emptions where less. We also saw that task sets discussed in FPZL did met their deadlines but

context switching and number of pre-emptions where more. But for Hard Real-Time Systems meeting their deadlines is of the highest priority hence, increase in number of pre-emption is not very significant as in other algorithms where the task sets does not even meet their deadlines.

We also saw in LLGF, that if the overhead involved with the pre-emption is significant then, we can implement LLFG, which is especially derived so as to decrease the number of pre-emptions.

REFERENCES

[1] M. Cirinei and T. P. Baker, "Edzl scheduling analysis,"

[2] R. I. Davis and A. Burns, "Fpzl schedulability analysis," 2011.

[3] J. Lee, A. Easwaran, and I. Shin, "Laxity dynamics and llf schedulability analysis on multiprocessor platforms," 2012.