Virtual Laxity Driven Scheduling Algorithm for Multiprocessor Real-Time Systems

Gökhan Seçinti[1], D. Turgay Altılar[1]
1: Department of Computer Engineering, Istanbul Technical University

Abstract

An optimal real time scheduling algorithm has been presented in this paper for multiprocessor systems. It has been assumed that system consists of m identical processors and only contains periodic tasks with implicit deadlines. Despite most of the recent studies focus on the notion of fairness, the proposed algorithm improves average response time of the tasks and decreases the number of the context switches by reducing the fragmented execution of the tasks without any fairness constraints.

I. Introduction

In spite of the common usage of traditional scheduling algorithms such as Earliest Deadline First (EDF) and Least Laxity First (LLF) in single processor systems, it has been shown that they are not suitable for multiprocessor scheduling problems.

There had not been an algorithm that provides solution for real-time multiprocessor scheduling problem for periodic tasks with implicit deadlines until Pfair has been proposed by Baruah et. al in 1996 [3]. Pfair makes scheduling decisions and ensures fairness between all tasks at any quantum (discrete time slots) during the scheduling. However such an approach increases the number of context switches and scheduling overhead significantly. After the concept of fairness for multiprocessor scheduling was proposed, most of the recent studies have focused on fairness to meet the deadlines of periodic tasks.

In the Pfair algorithm, the lag of the tasks are kept within a window of size of two quanta (between -quantum and +quantum) to provide fairness. The lag can be defined as the difference between the expected and actual value of the execution time that has already been consumed. This expected value is computed with regard to fluid scheduling. Early Release Fair [1] relaxed fairness constraints for early use of processors. Thus if there are idle processors and waiting tasks in the system, ERfair algorithm allows these waiting tasks to be dispatched earlier than the expected time for Pfair. Therefore it provides work conserving scheduling and increases the average response time of tasks. ERfair exploits under-utilized systems.

Boundary Fair (BF) algorithm [9], is another variant of Pfair. It aims to ensure fairness among the tasks at the deadlines rather than at each quantum as in the Pfair. Between two deadlines fairness constraints could be bend to provide schedule plans with less context switches than Pfair.

EDF with task splitting and k processors in a group (EKG) [2] has a different approach to the scheduling problem for periodic tasks in multiprocessors systems. If migration of a task is required, EKG splits this periodic task into two sub tasks and assign task to different processors to work on different times, and tries to meet their deadlines separately. EKG algorithm is able to define a bound for the number of the context switches in 100% utilized systems.

LLREF [4] (largest local remaining execution time first) uses fluid scheduling for intervals bounded by deadlines or release times of tasks. These intervals are defined by Time and Local Execution Time Domain Plane Abstraction (TLPA). In the TLPA algorithm, local execution times of tasks in an interval are calculated as the product of utilization factors of these tasks and duration of an interval. Task with the largest local remaining time is assigned processor until its local execution time runs out or it has been preempted by another task which has zero laxity.

A derivation of LLREF algorithm, Extended-Time and Nodal Execution Time Domain Plane Abstraction (E-TNPA) [5], provides work conserving algorithm for periodic tasks with implicit deadlines in multiprocessor system. The E-TNPA algorithm increases local remaining execution time of tasks, that is calculated as the product of utilization factors and duration of an interval in LLREF, if there is an idle processor while task queue is not empty. Thus E-TNPA algorithm never left any processors idle when there is at least one tasks waiting. The E-TNPA algorithm, providing the same schedule plan with LLREF for 100% utilized task sets, reduces the number of context switches for task sets utilized fewer than 100%.

As shown by the given short survey, optimal scheduling solutions have been available for years in literature. Although most of the recent works rely on the concept of fairness, we believe that for a 100% utilized systems scheduling with the sole concern of meeting deadlines would provide less context switches and faster response times. In this paper, a new algorithm, Virtual Laxity Driven Scheduling (VLDS), is proposed. The VLDS algorithm ignores fairness for the sake of both less context switches and shorter response times. However, the VLDS algorithm also schedules the task sets that can only be scheduled by fair schedulers as those previously mentioned in this paper.

The rest of the paper is organized as follows. Section 2 provides detailed information about the VLDS algorithm and the proof of optimality of the algorithm. In Section 3, the simulation environment is explained briefly and the comparative test results of the algorithms are given. Section 4 consists of the conclusion.

II. System Model and Proposed Algorithms

II.1. System Model

Periodic task model proposed by Liu and Layland [7] has been used in this paper and it has been assumed that actual execution time of all tasks is equal the worst case execution times. Periodic tasks (T_i) are defined by their period and execution time. Arrival times of the first instances for all tasks are considered as 0. Hereafter absolute deadline of active job of the task (T_i) would be represented as d_{T_i} and the remaining execution time of this job would be represented as e_{T_i} It is assumed that system consists n periodic tasks and m identical processor. Global Task Set (GTS) that contains all active jobs belong to tasks (T_i) in the system is defined as follows. Because there can be only one active job belong to a task, variables that defines these jobs would be used as T_i in defined task sets.

$$T_i \in GTS \;\; i = 1,2 \dots n \qquad (1)$$

The system model does not contain any aperiodic or sporadic tasks. The utilization and laxity value of a task is defined as below. (t: current time)

Utilization $$u_{T_i} = \frac{executionTime}{period}$$

Laxity $$l_{T_i} \;=\; d_{T_i} - e_{T_i} - t$$

Processors which have been defined in the system model are identical. The costs of migrations and context switches are ignored in the system model.

II.2. The Proposed Algorithms

The VLDS is composed of two algorithms that would work consequently. One of the algorithms is used to create sub-work sets from GTS, another one is used to schedule these sub-work sets.

Basic principal of the VLDS algorithm is to create sub work sets according to the nearest deadline. All tasks in this sub-work set must share the same deadline. Then the scheduling problem can be resolved over the sub-work set with Least Laxity First (LLF). However a derivation of the LLF algorithm which constraints preemption operations has been proposed to reduce the number of context switches in scheduling.

II.2.1. ***Least Laxity First with Preemption Constraints (LLFPC):*** With the introduction of preemption constraints to Least Laxity First (LLF), the number of context switches and the overhead of the scheduling are reduced by removing the necessity of the laxity calculations on each quantum.

LLFPC algorithm will assign least laxity tasks first and define next preemption point according to laxity value of the unassigned task which has the least laxity. This will prevent context switches until an unassigned task has no (zero) laxity. When preemption occurs algorithm will reassign tasks by their new laxity values and define the new preemption time. State transition diagram of LLFPC algorithm is given in Figure 1.

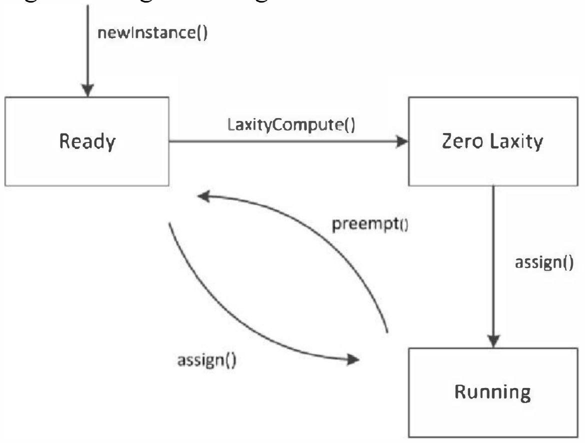

Figure 1: State transition diagram of the Least Laxity First with Preemption Constraints Algorithm

A task set has been given in Table 1 to examine improvements which LLFPC offers over LLF.

Schedule plan created by the LLF algorithm is given in Figure 2 for the given task set. As can be seen, number of context switches increases dramatically even for such a small scale system compose of two processor and four tasks when there exists tasks which have equal laxity values.

Table 1: Task Set 1

Task Name	Execution time	Period (Deadline)
T1	6	10
T2	6	10
T3	6	10
T4	2	10

As seen in Figure 2, the number of context switches in the schedule plan created by the LLFPC algorithm is lower than the LLF algorithm.

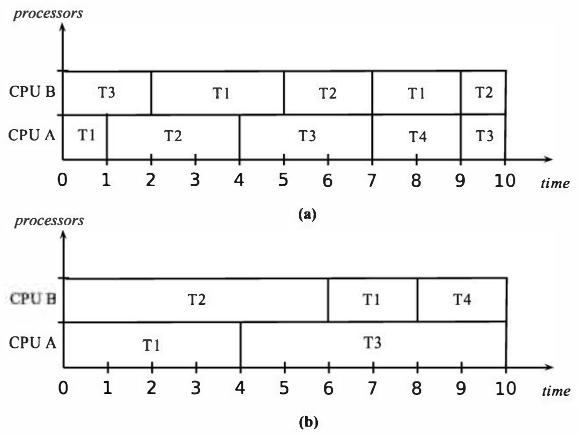

Figure 2: Schedule plan of LLF (a) and LLFPC (b) for given Task Set I.

II.2.2. ***Creating Sub-Work set***: First step in the VLDS algorithm while creating sub-work-set is determining the nearest deadline (D_n).

$$D_n = \min\{d_{T_i} \mid T_i \in GTS\} \qquad (2)$$

The duration between the current time and D_n is called interval and all calculations are bounded with this interval. Once D_n is determined, total number of time slots provided by the processors which is contained by the interval is given in Equation 3.

$$\text{totalTimeSlots} = \mathrm{m} \cdot (\mathrm{D_n} - \mathrm{t}) \qquad (3)$$

Tasks with deadline D_n are going to be scheduled within the interval. However there might be the additional time slots available within the interval for use of other tasks. To distribute these time slots among the tasks, they are grouped as High Priority and Low Priority Task Sets are described in Equations 4 and 5.

$$HP = \{T_i \mid d_{T_i} = D_n\}$$
$$LP = \{T_i \mid d_{T_i} > D_n\} \qquad (4)$$

$$HP \cup LP = GTS, HP \cap LP = \emptyset \qquad (5)$$

As can be seen, the deadlines of the HP tasks are equal to D_n and their executions have to be completed in the current interval. So the remaining execution time of each HP tasks must be included in the created work set to meet deadlines. The total time slots assigned to HP tasks is given in Equation 6.

$$\begin{matrix}\text{assigned}\\ \text{TimeSlots forHP}\end{matrix} = \sum_{j=0}^{|HP|} e_{T_j}^{HP} \qquad (6)$$

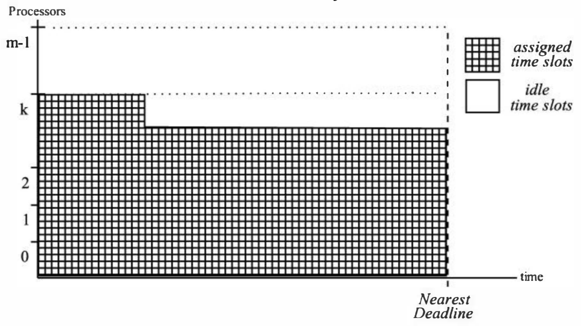

Figure 3: Work Set Time Diagram Context

From Equations 3and 6, the remaining idle time slots for LP tasks can be defined as in Equation 7.

$$\begin{matrix}\text{idle}\\ \text{TimeSlots}\end{matrix} = \mathrm{m} \cdot (\mathrm{D_n} - \mathrm{t}) - \sum_{j=0}^{|HP|} e_{T_j}^{HP} \qquad (7)$$

After the HP tasks included in the created work set, remaining idle time have to be distributed among the LP tasks. LP tasks that have lower laxities than the interval could miss their deadlines if these tasks have not been scheduled in the current interval. To prevent negative laxity (deadline miss) at least $(D_n - t - l_{T_i})$ time slots must be assigned for each LP task which its laxity is lower than the nearest deadline. The number of the total time slots required by the LP tasks in an interval (t, D_n) to prevent deadline misses is given in Equation 8.

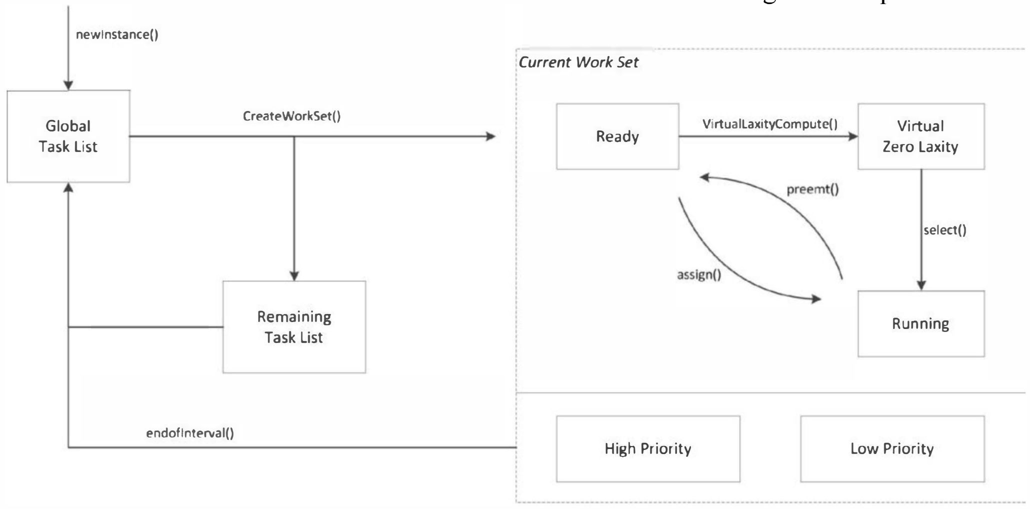

Figure 4: State transition diagram of the VLDS Algorithm

$$\begin{matrix}\text{totalTimeSlots}\\ \text{RequiredforLP}\end{matrix} = \sum_{j=0}^{|LP|}\left[(\mathrm{D_n} - \mathrm{t}) - l_{T_j}^{LP}\right] \qquad (8)$$

There have to be additional remaining idle time to assign LP tasks, after necessary distribution for LP and HP tasks to prevent deadline misses in the current interval has been made. It has been shown in proof of optimality that it is impossible to schedule given task set if there is no remaining idle time just after the necessary distribution.

After the necessary distributions were made, remaining idle time has been distributed among LP tasks according to their laxity values. Until no idle time is left in the current interval or no waiting task is left in GTS, maximum idle time that could be assigned to a LP task has been assigned by starting the LP task with the lowest laxity. This last distribution provides improvements that algorithm offers. Idle time slots left after the necessary distribution is assigned to LP tasks without any fairness constraints. This reduces the fragmented execution of the tasks and so decreases the number of context switches.

Once all possible distribution of the time slots in the interval among tasks has been performed, LP Tasks which have been assigned the current work set could be considered as virtually divided into two parts. The first part of the task contains the information that is going to be used in the scheduling of the current sub work set by LLFPC algorithm. Second part holds the information that belongs to the remaining part of the task. A few examples of the division given in Table 2 In these examples; Task 1 has been divided into two parts for the nearest deadline is equals to 5. For Task 2, it has been assumed that the nearest deadline is equals to 6.

Algorithm 1 CreateWorkSet(GTS, D_n)

```
Calculate $totalTimeSlots$
Assign time slots to HP tasks
Calculate $idleTimeSlots$
if $idleTimeSlots > 0$ then
  foreach(Task $T$ in LP)
  if $laxity < D_n$ then
    Assign $(D_n - t - laxity)$ time slot to $T$
    Update $idleTimeSlots$
  end if
end if
if $idleTimeSlots > 0$ then
  Distribute $idleTimeSlots$ to LP tasks by real laxities
end if
```

Pseudo code of CreateWorkSet function is given in Algorithm 1. Assigning time slots to the tasks could be considered as dividing task into two parts and assigned values will define virtual values of that task.

Table 2: Samples of Divided Tasks After CreateWorkSet Function Called.

Task Name	Virtual Values		Values of the Remaining Part		Real Values	
	exec. time	deadline	exec. time	deadline	exec. time	deadline
Task1	3	5	2	10	5	10
Task2	4	5	2	8	6	8

State transition diagram of the VLDS algorithm contains both LLFPC and CreateWorkSet algorithm is given in Figure 4. In current work set tasks could be distinguished from each other under two different categories. In first category, Tasks have the states of LLFPC algorithm, a task could be in any state such as Ready, Virtual Zero Laxity and Running at any time during the interval which the current work set is created for. Second category depends on the real deadline value of the task as defined in Equation 4. In this category, tasks belong to the High Priority State or Low Priority State and there is not going to be any transition between these states until the end of interval. This category has no effect on decisions of the LLFPC algorithm.

II.2.3. ***Proof of Optimality:*** If there is a condition that the total number of time slots required by the Low Priority tasks is equal or greater than the idle time left in the interval after assignment of the High Priority Tasks has done ($totalTimeSlotsRequiredforLPTask \geq idleTime$)then the scheduler could not assign the required time to LP tasks. Missing deadlines will be unavoidable.

Step1: In this step, with the given assumption at $t = 0$ (initial work set), it will be shown that where this condition exists, total utilization of GTS is greater than the number of processors.

Assuming tasks $T_j^{HP} \in HP \wedge T_k^{LP} \in LP$ with Equation 4, we could write the following equation for the scenario that deadlines cannot be satisfied by the scheduling algorithm.

$$\sum_{k=0}^{|LP|}\left[\mathrm{D_n} - l_{T_k}^{LP}\right] \geq m \cdot \mathrm{D_n} - \sum_{j=0}^{|HP|} e_{T_j}^{HP} \qquad (9)$$

$$\sum_{k=0}^{|LP|}\left[\frac{\mathrm{D_n} - d_{T_k}^{LP} + e_{T_k}^{LP}}{\mathrm{D_n}}\right] \geq \frac{m \cdot \mathrm{D_n} - \sum_{j=0}^{|HP|} e_{T_j}^{HP}}{\mathrm{D_n}} \qquad (10)$$

$$\sum_{k=0}^{|LP|}\left[1 + \frac{e_{T_k}^{LP} - d_{T_k}^{LP}}{\mathrm{D_n}}\right] \geq m - \sum_{j=0}^{|HP|}\frac{e_{T_j}^{HP}}{\mathrm{D_n}} \qquad (11)$$

$$\sum_{k=0}^{|LP|}\left[1 + \frac{(u_{T_k}^{LP} - 1)d_{T_k}^{LP}}{\mathrm{D_n}}\right] + \sum_{j=0}^{|HP|} u_{T_j}^{HP} \geq m \qquad (12)$$

From the definition of the Low Priority Task Set Definition (Eq. 4), we can write that.

$$d_{T_k}^{LP} > \mathrm{D_n} \Longrightarrow \frac{d_{T_k}^{LP}}{\mathrm{D_n}} = 1 + \epsilon \ for \ \epsilon > 0 \qquad (13)$$

$$\sum_{k=0}^{|LP|} \left[1 + (u_{T_k}^{LP} - 1)(1 + \epsilon)\right] = \sum_{k=0}^{|LP|} \left[u_{T_k}^{LP} + \epsilon \cdot (u_{T_k}^{LP} - 1)\right] \qquad (14)$$

$$\sum_{k=0}^{|LP|} \left[u_{T_k}^{LP} + \epsilon \cdot (u_{T_k}^{LP} - 1)\right] < \sum_{k=0}^{|LP|} u_{T_k}^{LP} \qquad (15)$$

From Equations 8 and 9, we derive that

$$\sum_{k=0}^{|LP|} u_{T_k}^{LP} + \sum_{j=0}^{|HP|} u_{T_j}^{HP} > m \qquad (16)$$

$$\sum_{i=0}^{|GTS|} u_{T_i} > m \qquad (17)$$

Step2: In the second step, the given proof has been generalized for all created work sets. In the fluid scheduling when work loads of tasks have been distributed fairly, it can be seen that it is possible schedule all periodic tasks without any deadline misses. As it can be seen in Figure 5, VLDS algorithm just replaces the workloads of the periodic tasks that belongs to LP task set in order to improve response times and reduce the number of context switches.

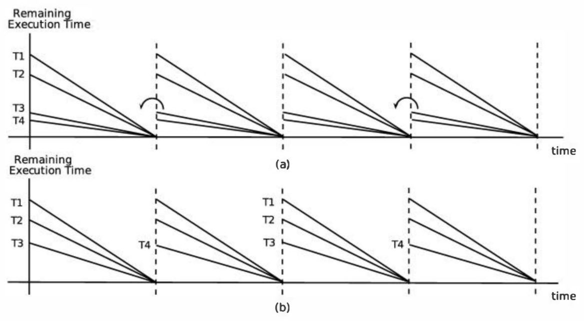

Figure 5: Work load replacement in Fluid Scheduling (a: Fair distribution, b: After replacement)

III. Performance Evaluation

III.1. Simulation Environment

STORM (Simulation TOol for Real time Multiprocessor scheduling) [8] has been used to develop and test algorithms. STORM provides discrete time simulation environment for multiprocessor systems that may contain periodic and aperiodic tasks.

III.2. Test Results

As test results two cases have been given below. First case shows the VLDS algorithm could provide the exact schedule plan with fair schedulers for the worst case scenario. In second case contributions of the algorithm has been examined.

III.2.1. ***Case 1 (The Worst Case Scenario):*** Task set in this scenario has been chosen to force every LP task to require time slots in created work sets. Only possible solution for given task set is to fairly distribute time slots among all tasks according to their utilization. Properties of the designed task set which is used in this test case are given in Table 3.

For the defined task set, it has been shown in Figure 6 that VLDS algorithm provides a fair schedule plan and satisfies all deadlines as the fair schedulers [3][6] could have achieved. In Figure 6, the order of a task is stated by the number of apostrophes. For example the third instance of T_B3 is shown by the notation T_B3'''.

Table 3: Task Set for Worst Case Scenario

Task Name	Execution time	Period (Deadline)
T_A1	1	2
T_A2	1	2
T_B1	2	3
T_B2	2	3
T_B3	2	3

III.2.2. ***Case 2 (Improved Solution Scenario):*** In this test case scenario, task set has been chosen to lower the amount of required time slots by LP tasks in created work sets. Improvements provided by VLDS algorithm have been stated. Designed task set for this scenario is given in Table 4.

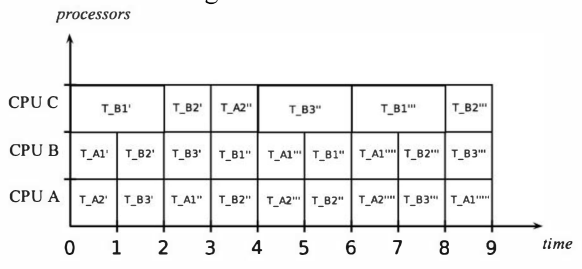

Figure 6: Schedule for Given Task Set in Case 1

Schedule plan that is produced by the VLDS algorithm has been compared the implementation of the Boundary Fair Scheduling [9].

VLDS algorithm tries to reduce partitioning of the tasks which their real deadlines are not equal the nearest deadline by relaxing fairness until a remaining task reaches its zero laxity.

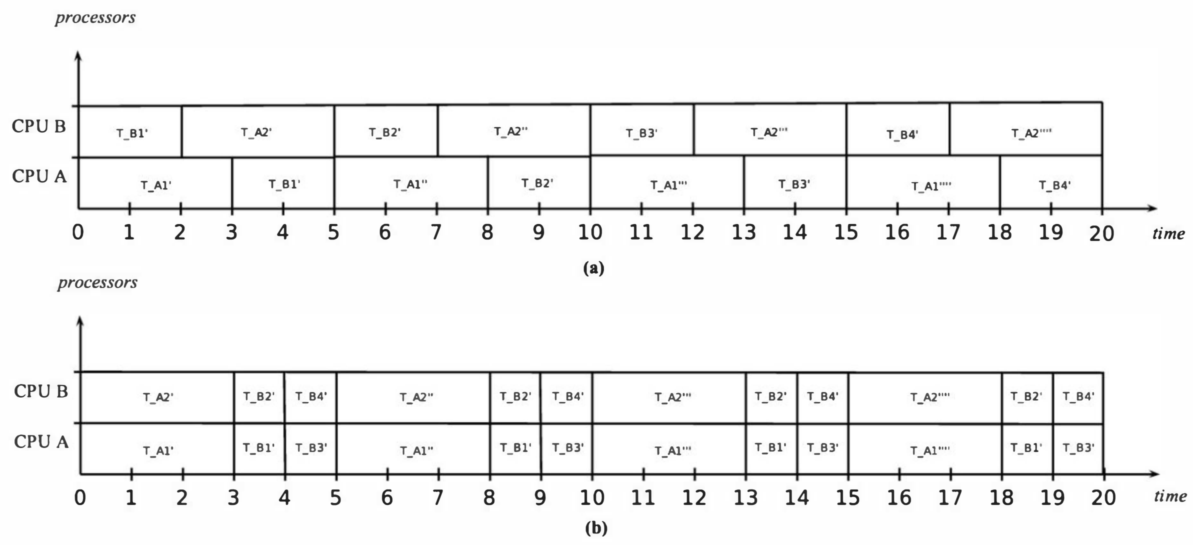

Figure 7: Schedule for Given Task Set in Case 2 (a: VLDS Algorithm, b: Boundary Fairness)

Table 4: Task Set for the Improved Scenario

Task Name	Execution time	Period (Deadline)
T_A1	3	5
T_A2	3	5
T_B1	4	20
T_B2	4	20
T_B3	4	20
T_B4	4	20

In examined case, none of the tasks have reached their zero laxity and VLDS algorithm has reduced partitioning notably. Thus the number of context switches is decreased from 32 to 8 in one super-period as can be seen in Figure 7.

Boundary Fairness algorithm ensures fairness for each deadline. So it completes all tasks which deadlines are equal to 20 in the time duration between 15 and 20. Because the partitioning of the tasks has been reduced, the VLDS algorithms provide smaller response times for these tasks.

IV. Conclusion

A work conserving and optimal scheduling algorithm, VLDS, has been presented and it has been stated that VLDS when compared fair schedule algorithms decreases the number of context switches and improves the average response times of the tasks. This paper shows that fairness could be relaxed or even ignored in some cases to provide better schedules. As a future work, cache awareness could be implemented to even reduce context switches and migrations at the points that one interval finishes and the new interval begin. Thus arbitrary assignments of the tasks to the processors could be replaced with the assignment decisions that take into account the cache contents of the processors.

References

[1] J. H. Anderson and A. Block, "Early-release fair scheduling," *In Proceedings of the 12th Euromicro Conference on Real-Time Systems,* pp. 35–43, 2000.

[2] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," in *IPP Hurray Research Group, Polytechnic Institute of Porto, Portugal HURRAYTR-060811,* 2006.

[3] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, pp. 600–625, 1996.

[4] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *RTSS*, pp. 101–110, 2006.

[5] K. Funaoka, S. Kato, and N. Yamasaki, "Work-conserving optimal realtime scheduling on multiprocessors,", pp. 13–22, 2008.

[6] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Br, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *In Proc of the 2010 22nd Euromicro Conf. on Real-Time Sys*, pp. 3–13, 2010.

[7] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[8] R. Urunuela, A.-M. Deplanche, and Y. Trinquet, *Simulation TOol for Real-time Multiprocessor scheduling*, 2009.

[9] D. Zhu, D. Mosse, and R. Melhem, "Multiple-resource periodic scheduling problem: How much fairness is necessary?" 2003.