Using Task Density as Heuristic to Schedule Real-time Tasks*

Carlos A. Rincon C., Albert M. K. Cheng

Department of Computer Science
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

Technical Report Number UH-CS-17-01

March 13, 2017

Keywords: Task Density, Scheduling, Real-time Systems, Performance;

Abstract

The purpose of this paper is to present the density of a task as a heuristic to schedule real-time tasks. The density of a task is defined as the ratio between the computation time of the task and the minimum of its relative deadline and period. We propose the Highest Task Density First (HTDF) algorithm to schedule tasks in real-time systems. HTDF is a dynamic priority scheduling algorithm that selects the task with the highest task density relative to the earliest deadline to guarantee that all tasks meet their deadlines. We present the feasibility analysis of the proposed algorithm and we compare its performance against the Earliest Deadline First (EDF) scheduling algorithm using as dependent variables the number of context switches and the number of preemptions. We generate 16 test files (with 100 task sets per file) using as independent variables: (a) Utilization (from 70% to 100%), (b) Number of tasks per task set (from 4 to 10). We also test our algorithm with a real test case task set, derived from the NASA X-38 Crew Return Vehicle avionics, using WindRiver Workbench 3.3 to calculate the execution time of the scheduler to compare its performance against EDF. The results show that: (a) HTDF is able to minimize the number of context switches and the number of preemptions when compared against EDF and; (b) HTDF performance in terms of the execution time of the scheduler is similar to EDF for the studied test case.

*Supported in part by the National Science Foundation under Awards No. 0720856 and No. 1219082.

Using Task Density as Heuristic to Schedule Real-time Tasks*

Carlos A. Rincon C., Albert M. K. Cheng

Abstract

The purpose of this paper is to present the density of a task as a heuristic to schedule real-time tasks. The density of a task is defined as the ratio between the computation time of the task and the minimum of its relative deadline and period. We propose the Highest Task Density First (HTDF) algorithm to schedule tasks in real-time systems. HTDF is a dynamic priority scheduling algorithm that selects the task with the highest task density relative to the earliest deadline to guarantee that all tasks meet their deadlines. We present the feasibility analysis of the proposed algorithm and we compare its performance against the Earliest Deadline First (EDF) scheduling algorithm using as dependent variables the number of context switches and the number of preemptions. We generate 16 test files (with 100 task sets per file) using as independent variables: (a) Utilization (from 70% to 100%), (b) Number of tasks per task set (from 4 to 10). We also test our algorithm with a real test case task set, derived from the NASA X-38 Crew Return Vehicle avionics, using WindRiver Workbench 3.3 to calculate the execution time of the scheduler to compare its performance against EDF. The results show that: (a) HTDF is able to minimize the number of context switches and the number of preemptions when compared against EDF and; (b) HTDF performance in terms of the execution time of the scheduler is similar to EDF for the studied test case.

Index Terms

Task Density, Scheduling, Real-time Systems, Performance;

I. Motivation

Rincon and Cheng [1] [2] proposed the mathematical background for using information theory principles to schedule real-time tasks (based on the work proposed by Shannon [3]). A preliminary analysis of performance showed that the execution time of our scheduling solution based on information theory was higher than EDF because of the increase in the computational complexity due to the use of the logarithm function to calculate parameters based on information theory.

After further analysis of our study about using information theory in real-time systems, we found that the proposed parameters are related to the density of the task ($TD_i = c_i/\min(d_i, p_i)$). Therefore, to minimize the computation complexity of the scheduling solution we decided to simplify our approach by using the density of a task as a parameter to schedule real-time tasks.

EDF is a dynamic priority algorithm that uses the deadlines of the tasks to assign task priorities. This research aims to show that by using the density of the tasks as a parameter to schedule real-time tasks, we can minimize the number of context switches and the number of preemptions without increasing the execution time of the scheduler. Our goal is to show that the information from the computation time of each task provided by its density can be used to outperform EDF while using similar resources (CPU time).

This research focuses on a uni-processor environment using two scenarios: (a) synthetic task sets with periodic, independent, and synchronous release tasks with implicit deadlines to measure the performance of the studied algorithms in terms of the number of preemptions and number of context switches and; (b) task sets based on the X-38 avionics [4] with non-implicit deadlines and precedence constraints to measure the execution time of the scheduler and to corroborate the behavior of the studied algorithms considering the number of preemptions and the number of context switches.

Our goal in this paper is to show that a dynamic task density-based parameter can be used to schedule tasks in real-time systems. Using the computation time implicit in the density of the task, this new parameter provides additional information for the scheduler to generate a feasible solution with similar performance in terms of execution time than deadline-based scheduling algorithms while minimizing the number of preemptions and context switches.

*Supported in part by the National Science Foundation under Awards No. 0720856 and No. 1219082.

The contributions of this paper are:

- Presenting the design, feasibility analysis, and implementation of the HTDF algorithm.
- Comparing the performance of the HTDF algorithm against EDF using the number of context switches, the number of preemptions, and the execution time of the scheduler as dependent variables.

The rest of the paper is organized as follows. In the next section, we describe the related work for the studied problem. In section 3, we present the heuristic based on the density of the task as well as the design, feasibility analysis, and implementation of the proposed scheduling algorithm. Section 4 presents the performance comparison between HTDF and EDF scheduling algorithms. We give our conclusions and future work in section 5.

II. Related Works

In this section, we present previous research which used the density of the tasks to solve different scheduling problems in real-time systems. These papers aim to show that the density of the task can be used to achieve a different solution to the studied problem while improving the scheduler's performance.

Aldarmi and Burns [5] in their study Dynamic Value-Density For Scheduling Real-Time Systems present a value-based scheduling heuristic that combines the tasks' values with some of the tasks' dynamic attributes (tasks' remaining execution time), in order to derive dynamic scheduling priorities. Their research focuses on soft real-time task scheduling in a uni-processor environment with the following assumptions: (a) All tasks are aperiodic; (b) Tasks are independent of each other, excluding contention for CPU access; and (c) Scheduling is preemptive. They concluded that the proposed parameter (Dynamic Timeliness Density) is an effective CPU scheduling scheme, and it is more suitable than the traditional Static Value Density and/or Earliest Deadline First, to operate under all operating loads. This research shows that a task density based parameter can outperform the traditional real-time system scheduling algorithms under certain conditions.

Liu et. al. [6] present a new scheduling algorithm for real-time service-oriented problems (real-time services over the distributed computing infrastructure). Their solution uses two parameters (a profit utility function and a penalty utility function) based on the density of a task. Their results show that the proposed algorithm can outperform the current scheduling algorithm used to solve the real-time service-oriented problems.

Chapter 7 of Fisher [7] work proposes the Partitioned Scheduling and Schedulability Analysis of Sporadic Task Systems, presenting a preemptive multiprocessor scheduling of sporadic real-time systems under the partitioned paradigm. He implements a partitioning solution based on the maximum job density and demand-based load metrics to reduce the multiprocessor scheduling problem to a series of uni-processor problems. He shows that the demand-based load and maximum job density metrics may be exactly computed in pseudo-polynomial time for general task systems and approximated in polynomial time for sporadic task systems. We will use these findings as the foundation for our future solution based on the density of a task to schedule real-time tasks in a multiprocessor environment.

The researchers from these previous studies used the additional information provided by the density of the task to present a new approach to solve the studied scheduling problem on both uni-processor and multiprocessor platforms. Aldarmi [5] showed that the computation time provides the information needed by the scheduler to minimize the number of preemptions.

Other works like [8], [9], [10] and [11] use different variations of the density of a task as a heuristic to solve different problems in real-time systems.

In our research, we propose a different approach based on the results of our work related to information theory principles and real-time systems scheduling to present a new heuristic that can be used in both uni-processors and multiprocessors platforms.

III. Task Density and the Highest Task Density First algorithm

The density of a task (TD_i) is defined by the term c_i/min(d_i, p_i) [12], where task i has a release time = r_i, computation time = c_i, deadline = d_i, period = p_i, the hyper-period ($hperiod$) of the system = least common multiple of the periods, and the utilization of the task set equal to ($U = \sum_{i=1}^{m} c_i/p_i$) [13],

The heuristic that we use to assign the priority of the tasks is represented by the minimum amount of task density that must be contained in a studied time interval. This interval is represented by the closest deadline that has to be met in a scheduling problem based on the current scheduling time t.

The task density per studied interval ($TD_{i,t}$) is equal to:
$TD_{i,t}$ = c_i/min(d_i, p_i) * min$(d, p)_t$ / min(d_i, p_i)
where min$(d, p)_t$ is the closest deadline to the scheduling time t.

Our approach uses the information provided by the computation time implicit in the density of the tasks to improve the performance of our solution while using the same time as EDF. For a task set with two (one-instance) tasks represented by the notation: J_i=(r_i,c_i,d_i), if J_1 = (0,1,3) and J_2=(0,2,4), then ($TD_{1,3}$) will be equal to 0.333 and ($TD_{2,3}$) = 0.375, therefore our scheduler will run J_2 first and then J_1 even though J_1 has the earliest deadline. Because of the information provided by $TD_{i,t}$ our algorithm knows that: (a) both J_1 and J_2 can be scheduled by t=3; and (b) there is no need to preempt J_2.

A. HTDF algorithm design

The HTDF algorithm uses the task density per studied interval ($TD_{i,t}$) as a parameter to assign the priorities of the tasks. At each scheduling point, the scheduler will select the tasks with the highest ($TD_{i,t}$). Our algorithm is a dynamic priority solution because it updates the parameters of the tasks (c_i, d_i) before calculating their ($TD_{i,t}$).

To minimize the number of context switches and the number of preemptions, we propose to use an event-driven method based on the density of a task to the improve the performance of the algorithm by adding a decision point to the normal scheduling points (a new task arrives, a running task ends) using the following criteria:

1) Select the task with the task density for the studied interval ($TD_{i,t}$).
2) Select the next scheduling point based on:
   If the computation time of the selected task (c_i) is less than min$(d, p)_t$ -$\sum_{k=1}^{m,\forall k\neq i}\lceil c_k * d_t/d_k\rceil$, then:
   the next scheduling point is the current time plus c_i.
   Else, the next scheduling point is the current time plus min$(d, p)_t$ - $\sum_{k=1}^{m,\forall k\neq i}\lceil c_k * d_t/d_k\rceil$.

Based on these additional criteria to calculate the next scheduling point, we present the design of the HTDF scheduling algorithm (See Algorithm 1).

B. Feasibility Analysis

Based on the design of HTDF in terms of the event-driven method to calculate the next scheduling point and taking into consideration the highest task density value based on the studied interval, we present the following mathematical analysis to show the feasibility of HTDF:

Assuming that we have a task set S with implicit deadlines and $U = \sum_{i=1}^{m}(c_i/d_i) \leq 1$. If S is not schedulable by HTDF, this means that for any deadline d_i, the sum of the computation time of all the scheduled tasks for the studied interval must be greater than the amount of time represented by this interval (d_i).

Mathematically, we have:
c_i + $\sum_{k=1}^{m,\forall k\neq i}(c_k * Instances_{k,t}) > d_i$
where $Instances_{k,t}$ = $TD_{k,t}$ / TD_k Then:
$Instances_{k,t}$ = min$(d, p)_t/d_k$
$c_i + \sum_{k=1}^{m,\forall k\neq i}(c_k$*(min$(d, p)_t/d_k$)) $> d_i$,
considering that: min$(d, p)_t$ = d_i, we have:
$c_i + d_i * \sum_{k=1}^{m,\forall k\neq i}(c_k/d_k) > d_i$,
and because: $\sum_{k=1}^{m,\forall k\neq i}(c_k/d_k)$ = $U - c_i/d_i$, we have:
$c_i + d_i * (U - c_i/d_i) > d_i$
$d_i * (U - c_i/d_i) > d_i - c_i$
$(U - c_i/d_i) > (d_i - c_i)/d_i$ = $(U - c_i/d_i) > 1 - c_i/d_i$, then
$U > 1$

This result shows that task set S is not schedulable by HTDF only if U of the studied task set is greater than 100%.

Algorithm 1 Highest Task Density First scheduling algorithm (HTDF)

Input: Task set S

Output: Scheduling Diagram

schedulingTime=0
m=SIZE(S)
for i=1 ; $i \leq m$ **do**
  initialize r_i, c_i, d_i, p_i
end for
repeat
  for i=1 ; $i \leq m$ **do**
    $TD_{i,t}$ = c_i/min(d_i, p_i) * min$(d, p)_t$ / min(d_i, p_i).
  end for
  HTD=$\{$max$(TD_{i,t}(S)) \mid 1 \leq i \leq m\}$
  if SIZE(HTD) = 1 **then**
    SelectedTask = HTD(1)
  else
    if CurrentRunningTask $\in HTD$ **then**
      SelectedTask=CurrentRunningTask
    else
      SelectedTask=LowPID(HTD)
    end if
  end if
  TRemTask=$\sum_{k=1}^{m,\forall k \neq i} \lceil c_k * d_t / d_k \rceil$
  if $c_i <$ (min$(d, p)_t$ - TRemTask) **then**
    NextSchedPoint=c_i
  else
    NextSchedPoint=(min$(d, p)_t$ - TRemTask)
  end if
  schedulingTime=schedulingTime+NextSchedPoint
  for i=1 ; $i \leq m$ **do**
    d_i=d_i - NextSchedPoint
    p_i=p_i - NextSchedPoint
    if i = SelectedTask **then**
      c_i=c_i - NextSchedPoint
    end if
    if d_i = 0 and $c_i > 0$ **then**
      Print task i missed its deadline at $schedulingTime$
      exit
    end if
    if p_i = 0 **then**
      initialize r_i, c_i, d_i, p_i
    end if
  end for
until schedulingTime = $hperiod$
Print Scheduling Diagram

C. Executing the HTDF algorithm

Given the following example (with implicit deadlines) consisting of 3 tasks (J_1, J_2 and J_3) with the following parameters (r_i, c_i, d_i, t_i):

J_1= (0,1,4,4), J_2= (0,2,6,6), J_3= (0,2,6,6)

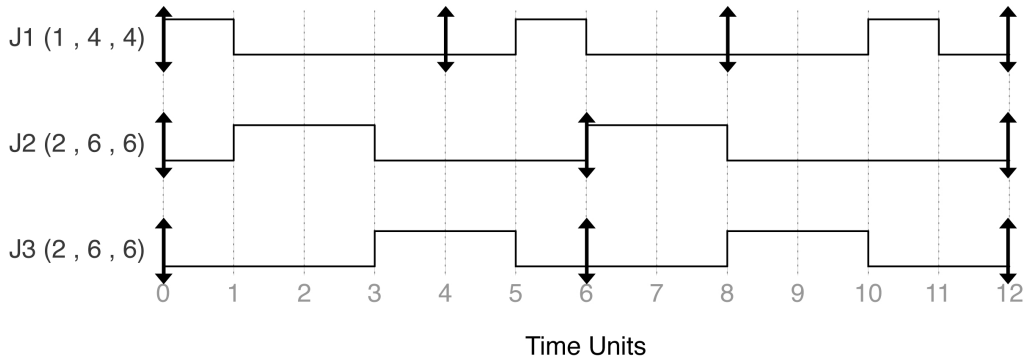


Fig. 1: HTDF scheduling diagram for the example task set

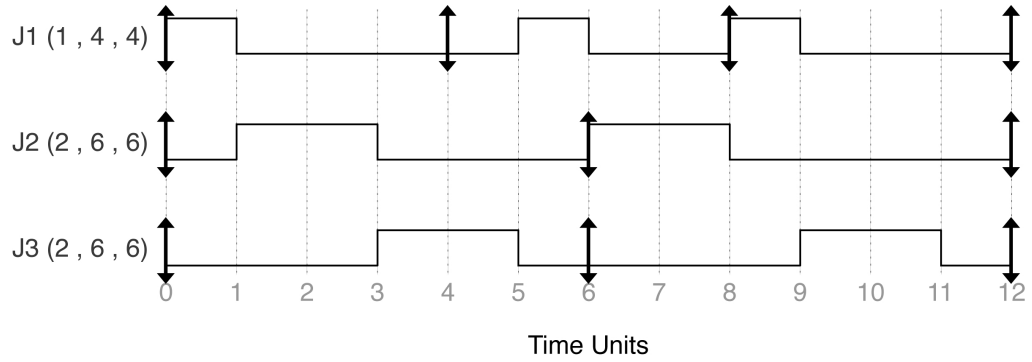


Fig. 2: EDF scheduling diagram for the example task set

We show the schedules derived by the HTDF algorithm (Figure 1), and the EDF algorithm (Figure 2):

For this example, both HTDF and EDF generate 6 context switches and 0 preemptions (but the schedules are different). The results from the HTDF algorithm are a consequence of the implemented event-driven method to select the next scheduling point (that tries to minimize preemptions). At time t= 8, HTDF selects task J_3 because its heuristic considers the computation time of the task (c_i=2) as part of the equation to set the priorities while EDF only considers the deadline of the tasks.

IV. Performance comparison between HTDF and EDF

A. Number of Context Switches and Number of Preemptions

1) Methods:

We evaluate the performance of HTDF by comparing with EDF using periodic, independent, and synchronous release tasks with implicit deadlines. We selected as dependent variables the number of context switches and the number of preemptions and as independent variables the utilization (70%, 80%, 90% and 100%) and the number of tasks per task set (4, 6, 8 and 10).

We use the *UUniFast* algorithm proposed by Bini and Buttazo [14], to generate the task sets. This algorithm generates independent tasks with randomly unbiased utilization factors. To generate the periods and computation time of the tasks based on these utilization factors, we randomly selected the tasks' periods from a set of predefined values of time periods (including the least common multiple of the set), making sure the least common multiple was always selected to guarantee that all the task sets have the same hyper-period. Lastly, we calculate the computation time per task to meet the utilization factor generated by the *UUniFast* algorithm.

Based on the values of the independent variables we generated 16 test files with 100 task sets per file and we executed the implementations of HTDF and EDF to obtain the average number of context switches and the average number of preemptions.

2) Results:

Average Number of Context Switches:

Table I shows the performance of HTDF and EDF in terms of the number of context switches based on the independent variables number of tasks per task set and utilization for each test file.

TABLE I: Number of Context Switches: HTDF vs EDF

		Utilization							
		70%		**80%**		**90%**		**100%**	
		HTDF	EDF	HTDF	EDF	HTDF	EDF	HTDF	EDF
# Tasks	**4**	63.68	65.07	76.99	78.25	93.91	95.76	114.55	116.32
	6	109.12	109.94	116.7	118.28	126.99	129.94	142.72	144.66
	8	152.17	153.28	167.12	168.68	183.58	186.63	193.62	196.02
	10	194.57	195.3	258.12	259.24	271.33	271.93	287.37	289.28

Figures 3 and 4 present the average number of context switches based on the number of tasks per task set and the utilization respectively.

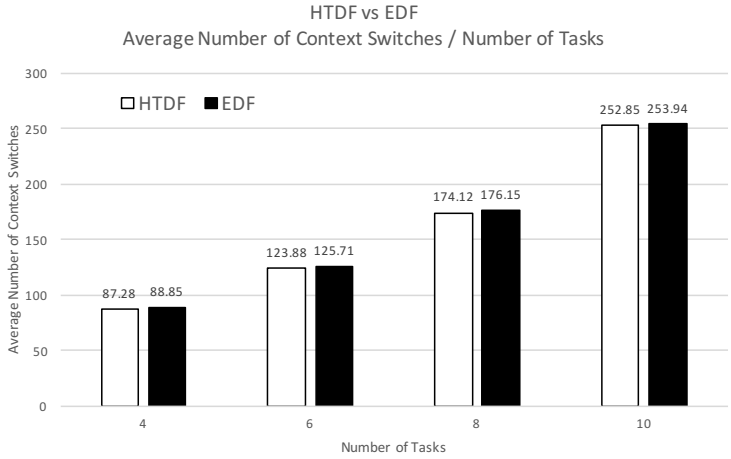


Fig. 3: Average Number of Context Switches based on the number of tasks per task set

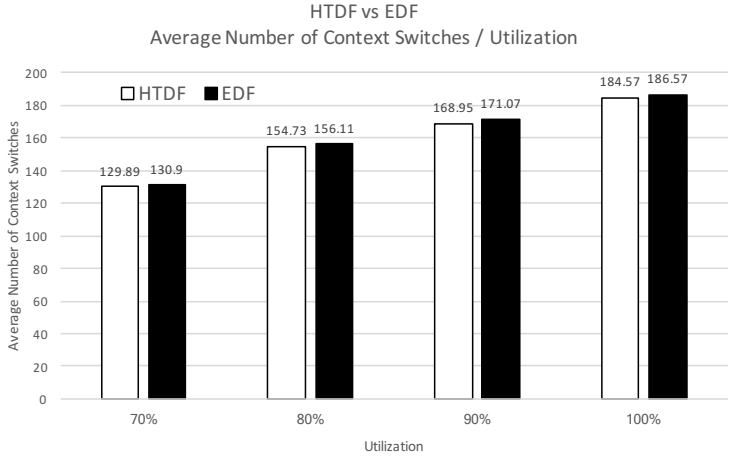


Fig. 4: Average Number of Context Switches based on the utilization

Average Number of Preemptions:

Table II shows the performance of HTDF and EDF in terms of the number of preemptions based on the independent variables number of tasks per task set and utilization for each test file.

TABLE II: Number of Preemptions: HTDF vs EDF

		Utilization							
		70%		**80%**		**90%**		**100%**	
		HTDF	EDF	HTDF	EDF	HTDF	EDF	HTDF	EDF
# Tasks	**4**	14.99	15.53	21.84	22.38	31.9	32.45	41.95	42.36
	6	19.8	20.64	25.48	26.27	34.04	35.46	42.51	43.8
	8	23.51	24.45	31.86	32.87	41.87	43.49	51.64	53.34
	10	25.29	26.29	33.04	34.16	43.79	44.39	57.37	59.28

Figures 5 and 6 present the average number of preemptions based on the number of tasks per task set and the utilization respectively.

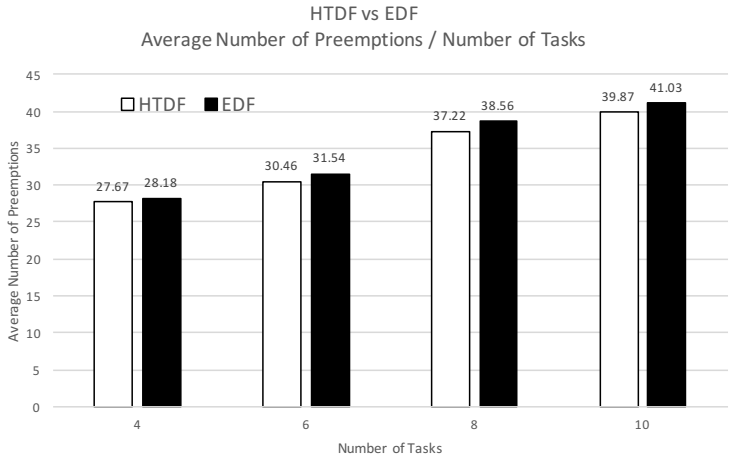


Fig. 5: Number of Preemptions based on the number of tasks per task set

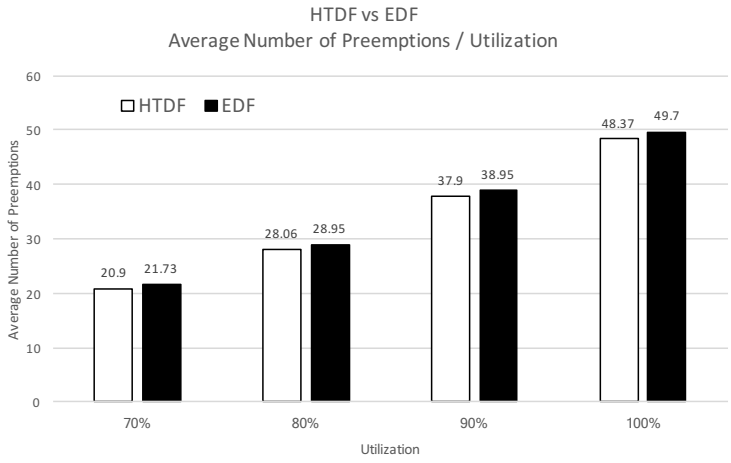


Fig. 6: Number of Preemptions based on the utilization

Results Analysis:

Based on the previous results, we can conclude that:

1) The number of context switches and the number of preemptions are directly proportional to both the number of tasks per task set and the utilization for the studied algorithms. This behavior is a consequence of the relationship between the complexity of the scheduling problem (due to the increase of the number of tasks and the utilization) and the probability of a context switch or a preemption.

2) For the generated task sets, HTDF outperforms EDF in terms of the number of context switches with an average difference of 1.20% for the number of tasks per task set and 0.99% for the utilization.

3) For the generated task sets, HTDF outperforms EDF in terms of the number of preemptions with an average difference of 2.93% for the number of tasks per task set and 3.11% for the utilization.

B. Scheduler execution time

1) Methods:

To measure the performance of the HTDF algorithm against EDF, we use WindRiver Workbench 3.3 on a server with an Intel i7-3770 processor running at 3.4 GHz, with 16 GB of RAM and 2 TB hard drive to implement the studied algorithms, using the system viewer to capture the scheduling diagrams.

a) The task sets:

We generate 4 task sets with utilization approximately equal to 70%, 80%, 90% and 100% respectively. These task sets are based on the X-38 avionics task set proposed by Rice and Cheng [4]. This task set has 13 periodic tasks with precedence constraints and non-implicit deadlines. Table III presents the parameters of the tasks and figure 7 presents the precedence constraints of the X-38 task set.

TABLE III: Original X-38 task set

No.	**Name**	c_i	d_i	t_i	**No.**	**Name**	c_i	d_i	t_i
1	ICP-I50FC-SENSOR	2	10	20	8	FCP-P10FC	40	50	100
2	FCP-I50FC	1	10	20	9	FCP-O10FC	1	50	100
3	FCP-P50FC	5	10	20	10	ICP-I10FC-CMDS	1	50	100
4	FCP-O50FC	1	10	20	11	ICP-I50NFC-SENSOR	5	100	100
5	ICP-I50FC-CMDS	1	10	20	12	FCP-I50NFC	1	100	100
6	ICP-I10FC-SENSOR	2	50	100	13	FCP-P50NFC	2	100	100
7	FCP-I10FC	1	50	100					

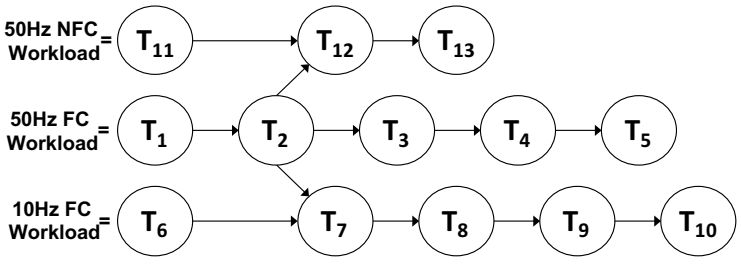


Fig. 7: X-38 task set precedence constraints

The X-38 201 avionics architecture is a four string, two-fault tolerant avionic system [15]. It uses four Flight Critical Computers (FCC) for redundancy. The central part of the architecture is the FCC. Each FCC has two processors (Flight Control Processor - FCP and the Instrumentation Control Processor - ICP). Figure 8 [15] shows the X-38 Vehicle 201 Avionics Architecture.

To test the performance of HTDF and EDF, we change the deadlines and periods of the tasks (maintaining the precedence constraints), making the X-38 task set feasible on a uniprocessor environment ($U \leq 1$). To comply with

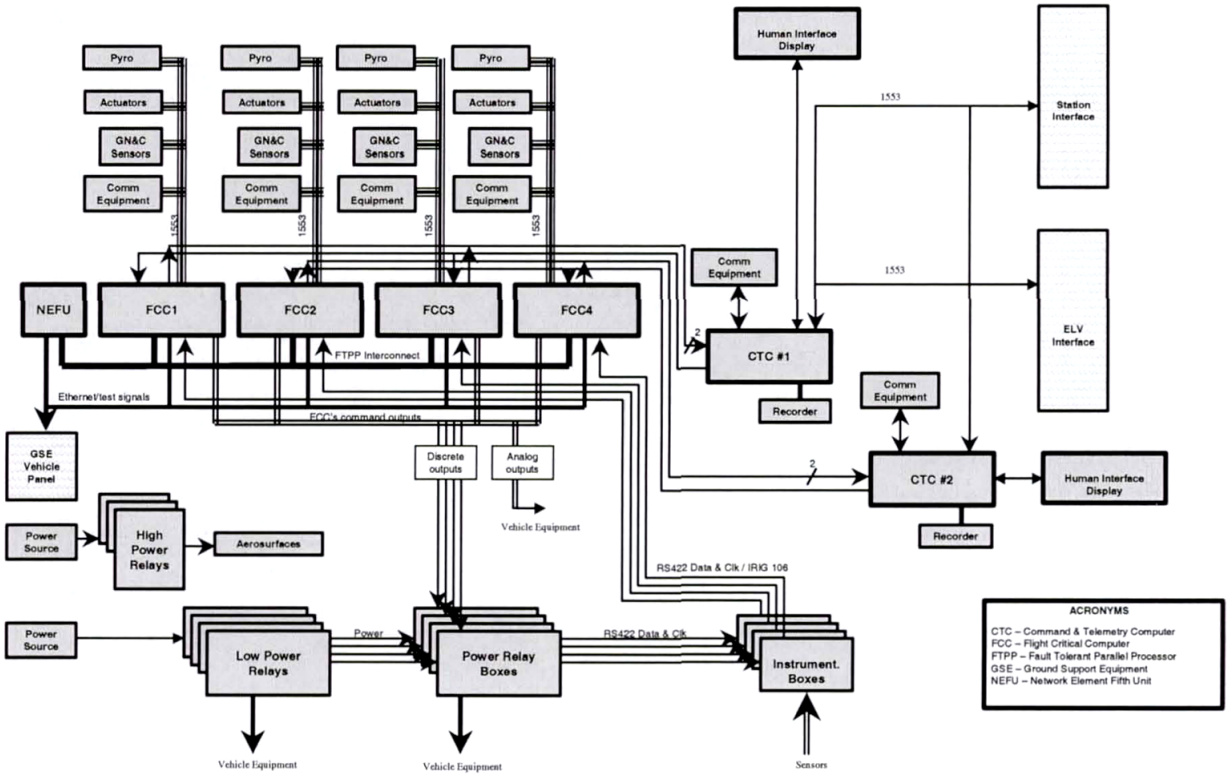


Fig. 8: X-38 Vehicle 201 Avionics Architecture

the precedence constraints from the original X-38 task set, we apply the Rendezvous model [16] to generate our independent task sets based on the selected values of utilization. Even though our simulations are not using the original X-38 task set, these experiments are trying to replicate the implementation of the X-38 avionics on a less powerful hardware, because we are using almost the same parameters as the original task set (number of tasks, precedence constraints and computation time of the tasks).

Task set with U = 70%: For this task set, we change the parameters (d_i and t_i) of two workloads from the original X-38 task set. For the 50Hz FC workload (Task 1 thru 5), we change the deadlines to 25ms and the periods to 50ms. For the 10Hz FC workload (Task 6 thru 10), we change the deadlines to 75 ms and the periods remained the same 100ms. The 50Hz NFC workload (Task 11 thru 13) parameters remain the same (100 ms). The utilization of the generated task set is equal to 73%.

Task set with U = 80%: For this task set, we change the parameters (d_i and t_i) of all the workloads from the original X-38 task set. For the 50Hz FC workload (Task 1 thru 5), we changed the deadlines to 25ms and the periods to 45ms. For the 10Hz FC workload (Task 6 thru 10), we change the deadlines to 75 ms and the periods to 90ms. For the 50Hz NFC workload (Task 11 thru 13), we change the deadlines to 90 ms and the periods to 90ms. The utilization of the generated task set is equal to 81.1%.

Task set with U = 90%: For this task set, we change the parameters (d_i and t_i) of all the workloads from the original X-38 task set. For the 50Hz FC workload (Task 1 thru 5), we change the deadlines to 25ms and the periods to 40ms. For the 10Hz FC workload (Task 6 thru 10), we change the deadlines to 75 ms and the periods to 80ms. For the 50Hz NFC workload (Task 11 thru 13), we change the deadlines to 80 ms and the periods to 80ms. The utilization of the generated task set is equal to 91.25%.

Task set with U = 100%: For this task set, we change the parameters (d_i and t_i) of all the workloads from the original X-38 task set. For the 50Hz FC workload (Task 1 thru 5), we change the deadlines to 25ms and the periods to 37ms. For the 10Hz FC workload (Task 6 thru 10), we change the deadlines to 70 ms and the periods to 74ms. For the 50Hz NFC workload (Task 11 thru 13), we change the deadlines to 74 ms and the periods to 74ms. The utilization of the generated task set is equal to 98.64%.

b) Implementing the studied algorithms in WindRiver WorkBench:

Figure 9 presents the model used to implement the studied scheduling algorithms in Workbench 3.3.

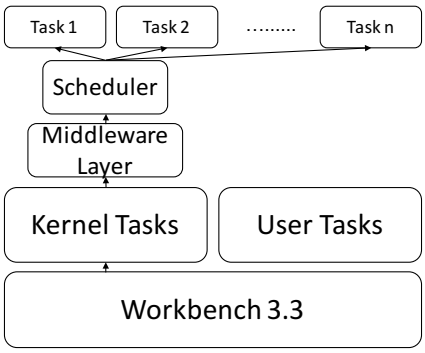


Fig. 9: Implementation of the schedulers in WindRiver WorkBench 3.3

We decided to implement our solution as a Workbench 3.3 kernel task (middleware layer). This layer is responsible for measuring the execution time of the tasks created in our environment. It also creates and executes the scheduler task to start the execution of the tasks. The scheduler task is responsible for loading the tasks table into the system and to create and activate the tasks depending on their priorities.

To make the tasks run for a specific number of milliseconds, we use the delaylib.c function [17] that provides hard delay routines for micro and millisecond periods. This function has a +/- 10% of error (based on our tests), therefore, to collect the data for our analysis, we executed the simulations several times taking the first 10 observations per experiment where the execution time of the scheduler was between +/- 1 ms of the theoretical hyper-period.

2) Results:

a) Scheduling Diagrams:

Tables IV, V, VI, VII present the generated task sets after applying the Rendezvous model to modified X-38 task sets with U = 70%, 80%, 90% and 100% respectively.

TABLE IV: Generated Task set with U = 70%

No.	**Name**	c_i	d_i	t_i	**No.**	**Name**	c_i	d_i	t_i
1	Task1	2	17	50	8	Task8	40	73	100
2	Task2	1	18	50	9	Task9	1	74	100
3	Task3	5	23	50	10	Task10	1	75	100
4	Task4	1	24	50	11	Task11	5	97	100
5	Task5	1	25	50	12	Task12	1	98	100
6	Task6	2	32	100	13	Task13	2	100	100
7	Task7	1	33	100					

TABLE V: Generated Task set with U = 80%

No.	**Name**	c_i	d_i	t_i	**No.**	**Name**	c_i	d_i	t_i
1	Task1	2	17	45	8	Task8	40	73	90
2	Task2	1	18	45	9	Task9	1	74	90
3	Task3	5	23	45	10	Task10	1	75	90
4	Task4	1	24	45	11	Task11	5	87	90
5	Task5	1	25	45	12	Task12	1	88	90
6	Task6	2	32	90	13	Task13	2	90	90
7	Task7	1	33	90					

When comparing the number of context switches and number of preemptions, we found that for U = 70% and 100% both HTDF and EDF generate the same number of context switches and number of preemptions. For U =

TABLE VI: Generated Task set with U = 90%

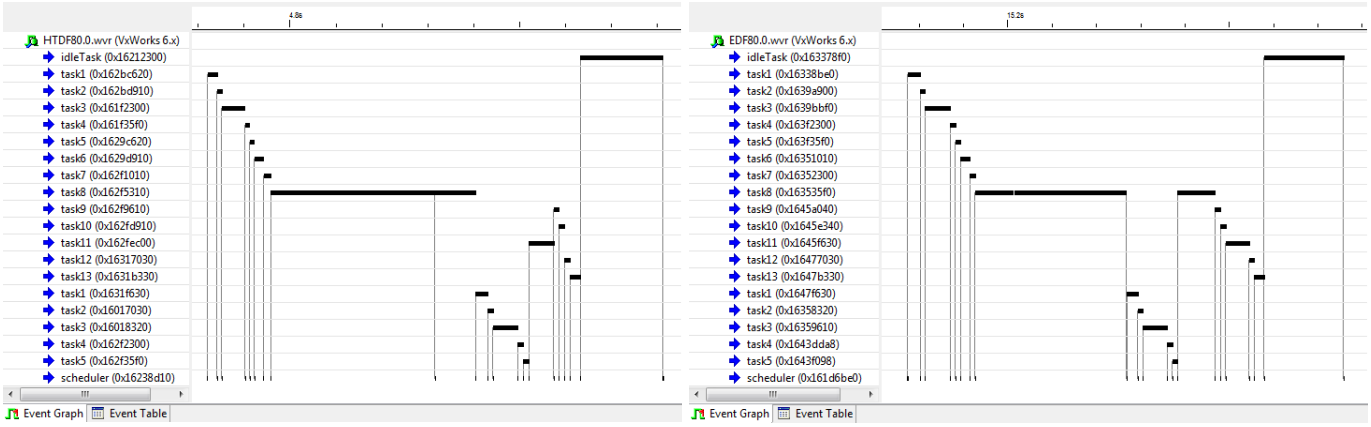
No.	**Name**	c_i	d_i	t_i	**No.**	**Name**	c_i	d_i	t_i
1	Task1	2	17	40	8	Task8	40	73	80
2	Task2	1	18	40	9	Task9	1	74	80
3	Task3	5	23	40	10	Task10	1	75	80
4	Task4	1	24	40	11	Task11	5	77	80
5	Task5	1	25	40	12	Task12	1	78	80
6	Task6	2	32	80	13	Task13	2	80	80
7	Task7	1	33	80					

TABLE VII: Generated Task set with U = 100%

No.	**Name**	c_i	d_i	t_i	**No.**	**Name**	c_i	d_i	t_i
1	Task1	2	17	37	8	Task8	40	68	74
2	Task2	1	18	37	9	Task9	1	69	74
3	Task3	5	23	37	10	Task10	1	70	74
4	Task4	1	24	37	11	Task11	5	71	74
5	Task5	1	25	37	12	Task12	1	72	74
6	Task6	2	27	74	13	Task13	2	74	74
7	Task7	1	28	74					

80% and 90%, HTDF was able to outperform EDF because it minimized by one the number of context switches and the number of preemptions.

For the task set with U = 80%, figure 10 shows that at t=45, task8 is running but the scheduler is called (because a new instance of task1 becomes active). For HTDF (figure 10.a), task8 doesn't get preempted because the scheduler knows that task8 can finish without making the other tasks in the system miss their deadlines (using the information provided by the density of the tasks). For EDF (figure 10.b), task8 gets preempted because the deadline of task1 is lower (t=62) than the deadline of task8 (t=73). For the task set with U = 90% we saw the same behavior but at a different scheduling point time (t=40).



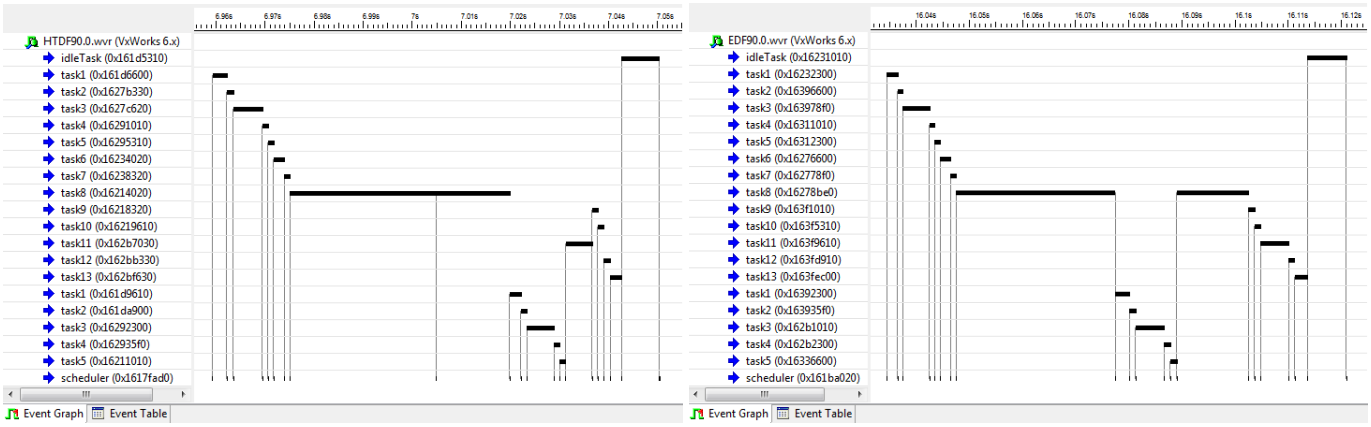
(a) HTDF (b) EDF

Fig. 10: Workbench System Viewer Scheduling Diagrams for U=80%

These results follow the same behavior shown in our previous experiments in terms of the minimization of the number of context switches and the number of preemptions. Therefore, we can conclude that for the studied task sets based on the X-38 avionics, HTDF outperforms EDF in terms of the number of context switches and the number of preemptions.

b) Scheduler execution time analysis:

Figure 12 presents the performance analysis of HTDF and EDF in terms of the execution time of the scheduler. The results show that for the task set with U=70% the average execution time of the scheduler for HTDF was



(a) HTDF (b) EDF

Fig. 11: Workbench System Viewer Scheduling Diagrams for U=90%

246.882 μsec and for EDF was 248.032 μsec. For the task set with U=80% the average execution time of the scheduler for HTDF was 241.84 μsec and for EDF was 248.815 μsec. For the task set with U=90% the average execution time of the scheduler for HTDF was 243.69 μsec and for EDF was 249.138 μsec. For the task set with U=100% the average execution time of the scheduler for HTDF was 247.366 μsec and for EDF was 249.178 μsec.

The analysis for the execution time of the scheduler for HTDF and EDF shows a similar performance for U = 70% and 100% because the scheduling solutions for both algorithms generated the same number of context switches and preemptions. For U = 80% and 90%, HTDF was able to reduce the execution time of the scheduler when compared against EDF because of the reduction in the number of preemptions. We can conclude that the performance of HTDF is similar to EDF in terms of the execution time of the scheduler because the difference between the studied algorithm is less than 5 μsec for the worst case (80% utilization).

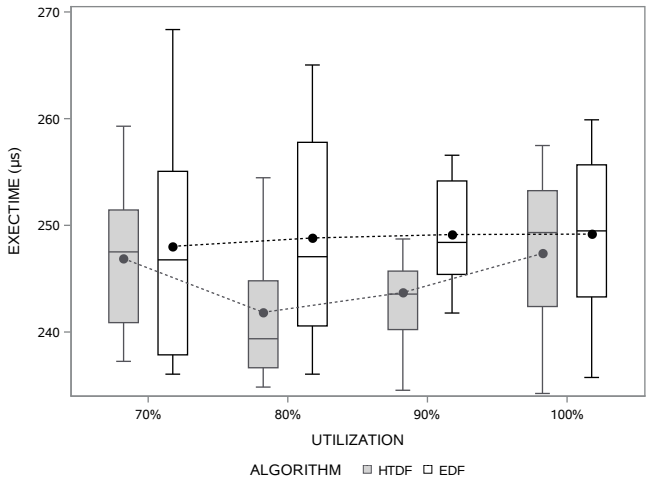


Fig. 12: Scheduler execution time analysis for the modified X-38 task sets

V. Conclusions and Future Work

In this paper, we have presented the task density per studied interval ($TD_{i,t}$) as a parameter to schedule real-time tasks. The HTDF algorithm sets the priorities of the tasks based on the highest $TD_{i,t}$ = c_i/min(d_i, p_i) * min$(d, p)_t$ / min(d_i, p_i), where min$(d, p)_t$ is the closest deadline to the scheduling time t, and uses an event-driven method to calculate the next scheduling time point to be analyzed by the algorithm.

We have presented the design, feasibility analysis and implementation of the HTDF scheduling algorithm for real-time tasks.

A comparison between the HTDF and EDF has been performed using two scenarios to measure the performance of the proposed algorithm in terms of: (a) number of context switches and number of preemptions; and (b) execution time of the scheduler. The first scenario used synthetic task sets with periodic, independent, and synchronous release tasks with implicit deadlines while the second scenario used the X-38 task set [4] to generate 4 task sets with precedence constraints and non-implicit deadlines.

The results from the first scenario show that HTDF outperforms EDF in terms of the number of context switches and the number of preemptions for all the generated test files (16 files, 100 task sets per file). The results from the second scenario show that the performance of HTDF is similar to EDF in terms of the execution time of the scheduler because the difference between the studied algorithm is less than 5 μsec for the worst case (80%).

For future work, we propose: (a) to study the use of the task density per studied interval ($TD_{i,t}$) as a parameter to schedule real-time tasks in multiprocessors (based on our findings and the related works); (b) to design a new scheduling algorithm for real-time systems in multiprocessors based on the HTDF algorithm proposed in this paper; and (c) to test our multiprocessor scheduling algorithm using the original X-38 avionics task set.

REFERENCES

[1] C. A. Rincon and A. M. Cheng, "Using entropy as a parameter to schedule real-time tasks," in *Real-Time Systems Symposium. WiP Session, 2015 IEEE*, pp. 375–375, Dec 2015.

[2] C. A. Rincon and A. M. Cheng in *Conference on Information Sciences and Systems (CISS 2017), Baltimore, Maryland, USA, March 22-24*, (accepted) 2017.

[3] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, 1948.

[4] L. E. P. Rice and A. M. K. Cheng, "Timing analysis of the x-38 space station crew return vehicle avionics," in *Real-Time Technology and Applications Symposium, 1999. Proceedings of the Fifth IEEE*, pp. 255–264, 1999.

[5] S. A. Aldarmi and A. Burns, "Dynamic value-density for scheduling real-time systems," in *Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on*, pp. 270–277, 1999.

[6] S. Liu, G. Quan, and S. Ren, "On-line preemptive scheduling of real-time services with profit and penalty," in *Southeastcon, 2011 Proceedings of IEEE*, pp. 287–292, March 2011.

[7] N. W. Fisher, *The multiprocessor real-time scheduling of general task systems*. PhD thesis, University of North Carolina at Chapel Hill, 2007.

[8] J. Zhuo and C. Chakrabarti, "An efficient dynamic task scheduling algorithm for battery powered dvs systems," in *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, (New York, NY, USA), pp. 846–849, ACM, 2005.

[9] R. Guerrra and G. Fohler, "On-line scheduling algorithm for the gravitational task model," in *2009 21st Euromicro Conference on Real-Time Systems*, pp. 97–106, July 2009.

[10] H. Chen and J. Xia, "A real-time task scheduling algorithm based on dynamic priority," in *Proceedings of the 2009 International Conference on Embedded Software and Systems*, ICESS '09, (Washington, DC, USA), pp. 431–436, IEEE Computer Society, 2009.

[11] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. Gill, "Parallel real-time scheduling of dags," *IEEE Transactions on Parallel & Distributed Systems*, vol. 25, no. 12, pp. 3242–3252, 2014.

[12] A. M. K. Cheng, *Real-Time Systems: Scheduling, Analysis, and Verification*. New York, NY, USA: John Wiley & Sons, Inc., 1 ed., 2002.

[13] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, pp. 46–61, Jan. 1973.

[14] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[15] C. Kouba, D. Buscher, and J. Busa, "The x-38 spacecraft fault-tolerant avionics system," 2003.

[16] A. Mok, "The design of real-time programming systems based on process models.," in *Real-Time Systems Symposium, IEEE*, pp. 5–16, 1984.

[17] "delaylib.c - self-calibrating hard delay routines." http://www.cs.ru.nl/lab/vxworks/www.rt.db.erau.edu/runscheduling/ delayLib.c.