

# COMPARING TRADITIONAL SCHEDULING TECHNIQUES WITH RECENT SCHEDULING ALGORITHMS

Ștefan Andrei  
Lamar University  
Department of Computer Science  
Beaumont, TX, USA  
Email: sandrei@cs.lamar.edu

Albert M.K. Cheng  
University of Houston  
Computer Science Department  
Houston, TX, USA  
Email: cheng@cs.uh.edu

Vlad Rădulescu  
Cuza University of Iași  
Department of Computer Science  
Iași, Romania  
Email: rvlad@infoiasi.ro

Nitin Joshi  
Lamar University  
Department of Computer Science  
Beaumont, TX, USA  
Email: njoshi@lamar.edu

**Abstract.** *With the burgeoning demand of embedded systems with extensive time sensitive processing throughout the globe, the research communities from the computing industry are continuously raising the bars by introducing better and faster process scheduling algorithms.*

*In this paper we try to compare the performances of one of the best known traditional scheduling algorithms such as EDF, LLF with some of the most recent works in the field of multi-processor, non-preemptive scheduling algorithms like Algorithm C and Algorithm D.*

*The tabulated results presented in this paper are obtained after running a java implementation of EDF, LLF and the algorithms presented in [ACR2011] and [ACR2013].*

**Keywords:** Multiprocessor platform; scheduling algorithm

## I. Introduction

In the study of real-time embedded systems, schedulability problem remains the most researched problem and over the course of several decades many pioneering researchers have presented scheduling algorithms such as EDF and LLF that changed the way scheduling analysis of task sets is done. These scheduling algorithms aim to provide optimal feasible schedules of task sets under various constraints. Despite such a radical increase in study of such algorithms, the schedulability analysis of non-preemptive tasks has received less attention in the real-time embedded research community.

There are many benefits of using non-preemptive tasks over preemptive tasks on multiprocessor platform. This is primarily because in non-preemptive scheduling the task instance runs until its computation time on the same processor where as in preemptive scheduling task migration creates a lot of computational overheads and makes prediction difficult [ACR2011].

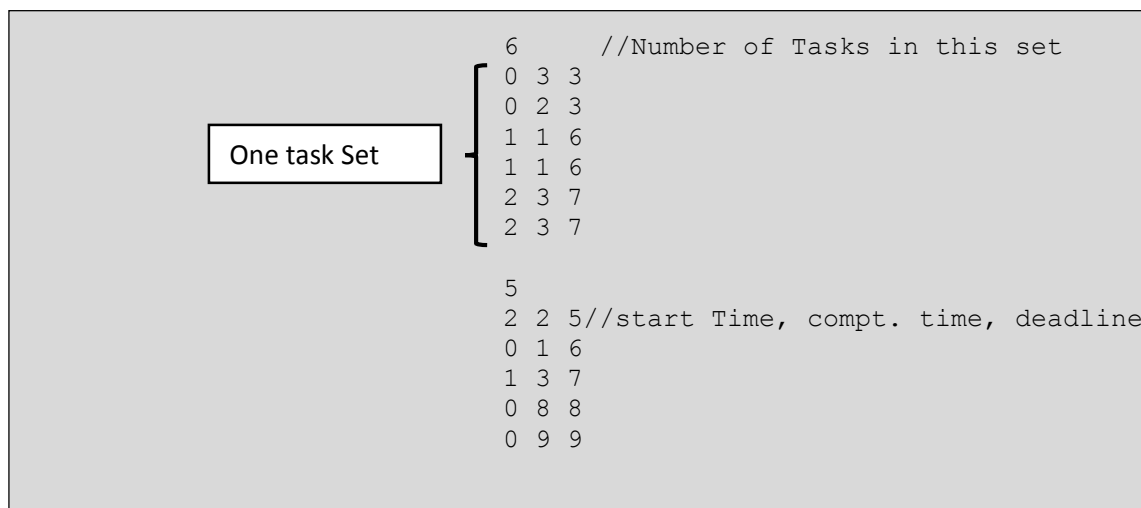
In this paper we prove by simulated implementation that despite EDF with idle task being an optimal method for uniprocessor platform [GMR1995], EDF is no longer optimal for multiprocessor platform. We show this by providing a feasible schedule by algorithm C described in [ACGR2010] and Algorithm D [ACR2013] for all task sets for which EDF failed to provide a feasible schedule. The tabulated result in Section III shows the feasibility of task sets by different scheduling methods.

The aim of this paper is to show that although the scheduling problem is NP-complete, the Algorithms C and D were still able to find a feasible schedule for most single-instance, non-preemptive and independent task sets. In the next section we provide the implementation details for the simulation tool used for the analysis.

## II. Implementation Details

The schedulability analysis tool is coded entirely using Java as the programming language. This tool is designed to provide feasibility of single-instance, non-preemptive and independent task sets according to EDF, LLF, Algorithm C and Algorithm D on a multiprocessor platform. In addition to this, the tool also provides the execution times of each task scheduler. This tool takes in as an input a text file, containing the task sets to be analyzed for feasibility and produces an output file (*result.txt*) which tabulates the feasibility analysis for each Algorithm individually along with their respective execution times. This tool is powerful enough to test feasibility of a large task set and any number of task-sets (of course limited by the space of the system).

The following snippet at Figure 2.1 describes a typical input file.



```

6 //Number of Tasks in this set
0 3 3
0 2 3
1 1 6
1 1 6
2 3 7
2 3 7

5
2 2 5 //start Time, compt. time, deadline
0 1 6
1 3 7
0 8 8
0 9 9

```

Figure 2. 1: Sample Input file

### A. Implementation of Scheduling Algorithms

The java implementation of the scheduling algorithms are based on the algorithms presented in [ACR2011]. The Algorithm C define in [ACR2011] is as follows,

```
sort the task set according to the " $\leq$ " relation
set all task chains as empty

while T is non-empty
  for each chain C
    add idle task to C if the start times of all tasks in T exceed  $c(C)$ 
  endfor
  let T be the first task in T such that there is at least one chain C with  $s(T) \leq c(C)$ 
  choose a chain C such that  $c(T) + c(C) \leq d(T)$ 

  if such a chain exists
    remove T from T
    add T to chain C
  else
    choose a chain C such that  $\text{last}(C) \rightarrow_x T$ , where  $x = c(C - \text{last}(C))$ 
    if such a chain exists
      add T to chain C before  $\text{last}(C)$  // so, switch
      remove T from T
    else
      stop // infeasible
    endif
  endif
endwhile
print all chains C // the feasible schedule
```

The Algorithm D [ACR2013] is an improvement of algorithm C, in which the following lines of computation from Algorithm C,

```
choose a chain C such that  $\text{last}(C) \rightarrow_x T$ , where  $x = c(C - \text{last}(C))$ 
  if such a chain exists
    add T to chain C before  $\text{last}(C)$  // so, switch
```

are replaced by

```
choose a chain C such that  $\text{last}(C) \rightarrow_x T$ , where  $x = c(C - \text{last}(C))$ 
 $T' = \text{last}(C)$ ;  $C' = C$ ;
while ( $T' \rightarrow_x T$  &&  $x \geq c(C' - T')$ )
   $C' = C - T'$ ;
   $T' = \text{previous}(T')$ ;
```

**endwhile**

Next, we compare the traditional approach, such as EDF and LLF with the new ones, namely Algorithms C and D respectively.

*Example 2.1:* Let  $T = \{T_1, T_2, T_3\}$  be a single instance and non-preemptive task set given by

$T_1 = (3, 2, 7)$ ,

$T_2 = (2, 3, 9)$  and

$T_3 = (0, 4, 10)$ .

On a uniprocessor platform, we can easily see that EDF fails.

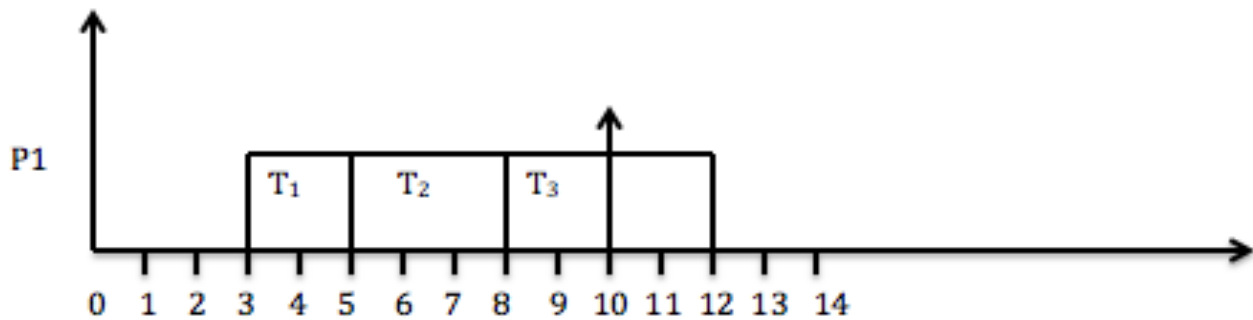


Figure 2. 2: EDF Schedule

LLF schedule also fails as shown below.

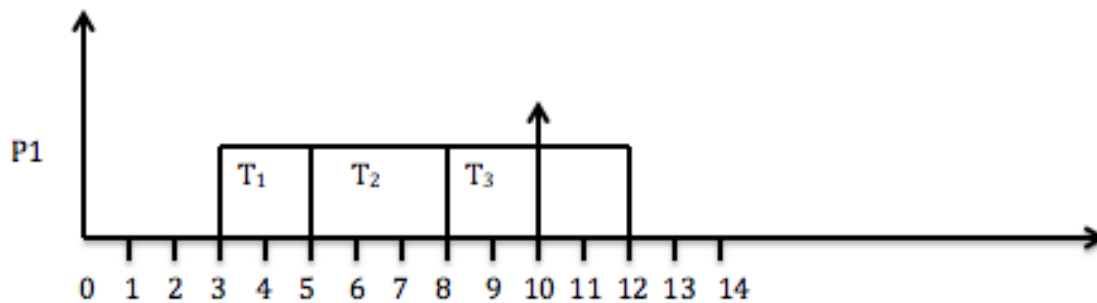


Figure 2. 3: LLF Schedule

Algorithm C also fails as shown below.

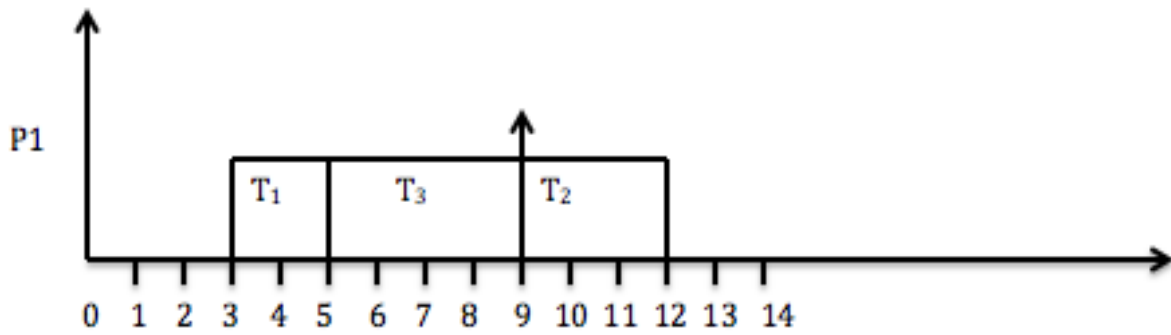


Figure 2. 4: Algorithm C schedule

Algorithm D is successful in finding a feasible schedule, as shown below.

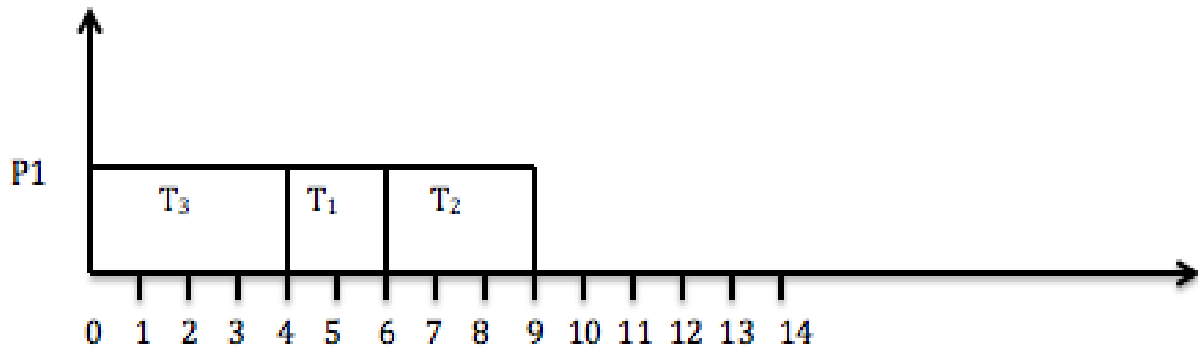


Figure 2. 5: Algorithm D schedule

### *B. Putting it all together – The tool*

The analysis tool combines all the scheduling algorithms into one tool, thus providing a single solution for performance analysis of these algorithms. In this sub-section we define the class diagram for the tool.

The following figure shows the class diagram of our tool.

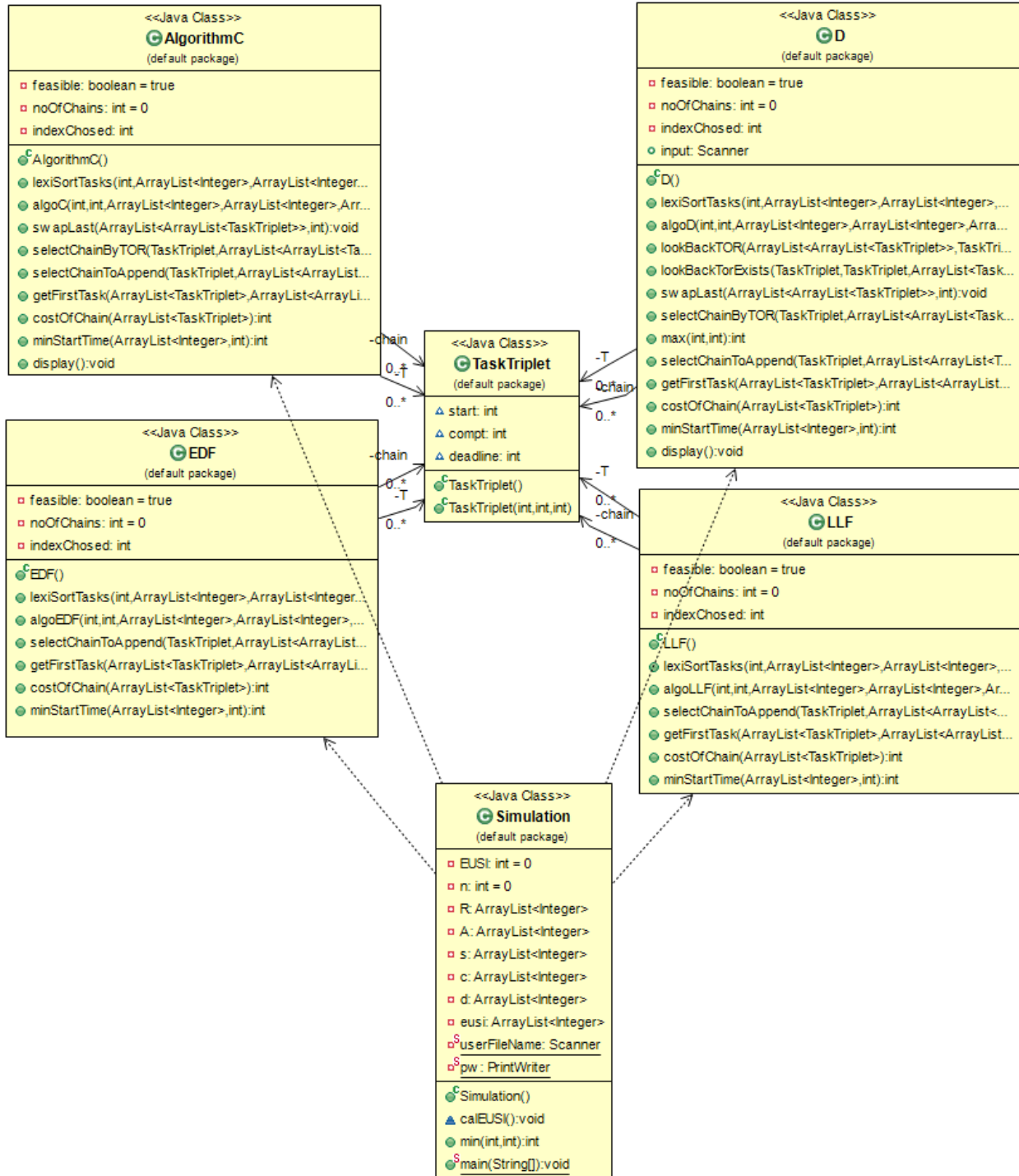


Figure 2. 6: The Class diagram for the Analysis tool

### III. Experimental Results

The analysis tool creates a table in a file (result.txt) with the feasibility and execution time for each algorithm for all the task sets in the input file. The execution time is calculated in nanoseconds.

For the following input task sets, the analysis table is as follows:

Input task sets:

T1 = { (0, 3, 3), (0, 2, 3), (1, 1, 6), (1, 1, 6), (2, 3, 7), (2, 3, 7) }  
T2 = { (2, 2, 5), (0, 1, 6), (1, 3, 7), (0, 8, 8), (0, 9, 9) }  
T3 = { (0, 3, 3), (1, 1, 6), (2, 3, 7), (1, 7, 8), (1, 8, 9) }  
T4 = { (3, 2, 7), (2, 3, 9), (0, 4, 10) }  
T5 = { (0, 1, 2), (0, 2, 4), (0, 4, 5) }  
T6 = { (0, 1, 3), (0, 2, 3), (0, 4, 4) }  
T7 = { (0, 2, 6), (0, 3, 8), (0, 2, 8), (0, 8, 10), (0, 2, 8), (0, 1, 9), (0, 4, 8),  
(0, 7, 19), (0, 7, 16), (0, 2, 8) }  
T8 = { (0, 2, 6), (1, 3, 8), (0, 2, 8), (0, 8, 10), (0, 2, 8), (0, 1, 9), (2, 4, 8),  
(3, 7, 19), (0, 7, 16), (1, 2, 8) }  
T9 = { (0, 8, 10), (0, 4, 10) }  
T10 = { (0, 5, 9), (0, 9, 14), (0, 7, 10), (0, 2, 7), (0, 3, 13), (0, 6, 21) }  
T11 = { (0, 1, 4), (2, 2, 6), (0, 7, 11) }  
T12 = { (0, 1, 5), (0, 1, 5), (2, 2, 6), (0, 10, 11) }  
T13 = { (0, 1, 5), (0, 1, 5), (2, 2, 6), (0, 11, 11) }  
T14 = { (1, 1, 9), (2, 2, 10), (3, 5, 12), (0, 17, 17), (0, 15, 15) }  
T15 = { (3, 5, 8), (0, 3, 8), (4, 1, 9) }  
T16 = { (3, 5, 8), (0, 3, 12), (4, 1, 9) }  
T17 = { (0, 2, 6), (0, 7, 9) }  
T18 = { (2, 4, 9), (0, 7, 12), (1, 5, 12), (0, 2, 13), (1, 2, 15), (0, 9, 22), (0, 7, 26),  
(10, 10, 36) }  
T19 = { (1, 1, 4), (0, 4, 7), (2, 4, 7), (3, 10, 18), (5, 5, 13) }  
T20 = { (2, 3, 7), (3, 2, 9), (0, 4, 9), (0, 15, 15) }

Table 3.1 clearly indicates that Algorithm D was able to find a feasible schedule for all 20 task sets, while EDF, LLF and C were not able to provide a feasible schedule for all the task sets.

The execution time for each algorithm is estimated by using the system clock. In Table 3.1, each algorithm's execution time is represented in nanoseconds. On a close analysis for execution times of each algorithm in Figure 3.1 and 3.2, we can say that Algorithm D takes slightly longer time to execute than other algorithms, but Algorithm D is able to find a feasible schedule in all 20 task sets.

Table 3.1: Analysis Results

---TaskSet---   -----EDF-----   -----LLF-----   -----C-----   -----D-----									
Task Set 1	NO	752989	NO	215580	YES	274608	YES	283846	
Task Set 2	YES	160145	NO	130888	NO	182729	YES	150392	
Task Set 3	NO	119595	NO	116515	YES	142693	YES	134994	
Task Set 4	NO	74939	NO	80073	NO	123189	YES	96498	
Task Set 5	YES	87258	YES	104710	YES	90338	YES	98551	
Task Set 6	YES	93417	YES	96498	YES	90852	YES	90851	
Task Set 7	YES	217119	YES	219173	YES	233031	YES	219686	
Task Set 8	YES	240730	YES	252022	YES	250996	YES	254076	
Task Set 9	YES	66214	YES	63647	YES	68267	YES	68267	
Task Set 10	YES	100604	YES	109842	YES	109843	YES	105224	
Task Set 11	YES	79046	NO	66726	YES	111383	YES	116745	
Task Set 12	YES	78533	NO	78533	YES	85718	YES	94958	
Task Set 13	YES	79559	NO	81098	NO	88285	YES	95984	
Task Set 14	YES	105223	NO	97524	NO	184782	YES	118568	
Task Set 15	YES	68267	NO	67754	NO	114462	YES	80073	
Task Set 16	NO	63134	YES	81612	YES	74426	YES	73913	
Task Set 17	YES	61594	NO	55435	YES	68267	YES	69214	
Task Set 18	YES	113436	YES	272554	YES	219173	YES	226358	
Task Set 19	YES	180162	YES	157065	YES	162711	YES	184783	
Task Set 20	NO	71346	NO	81099	NO	101630	YES	180072	

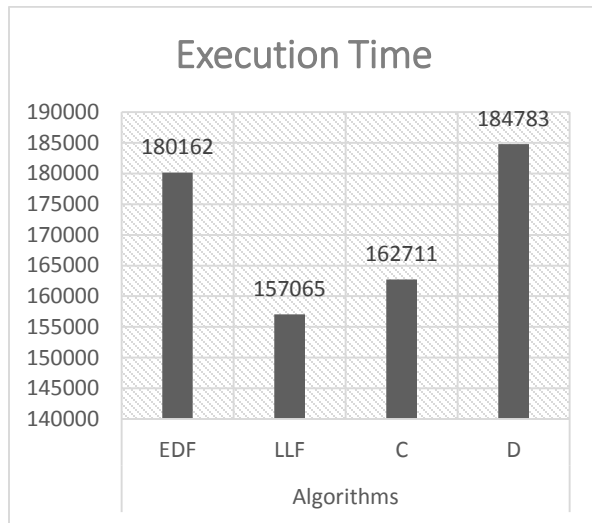


Figure 3. 1: The Execution time when all algorithms find a feasible schedule

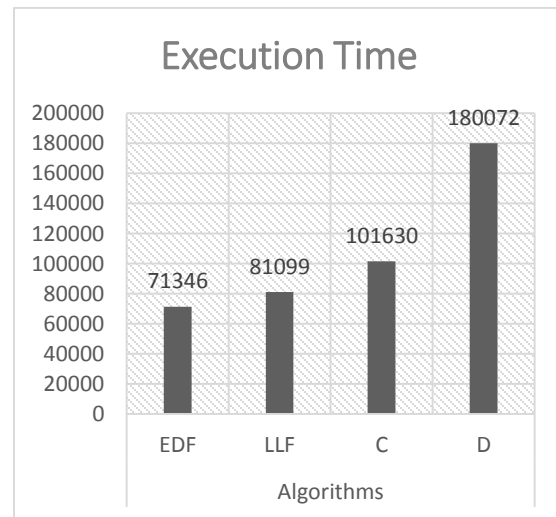


Figure 3. 2: The Execution time when only Algorithm D can find a feasible schedule



## IV. Conclusions

After analyzing the experimental results, we conclude that the latest scheduling algorithms such as Algorithms C and D outperformed the classic scheduling algorithms like EDF and LLF. In many cases where classical algorithms failed to provide a feasible schedule for some task sets, scheduling Algorithms C and D successfully provided a feasible schedule. This analysis also depicts that Algorithm D is the most powerful algorithm, since it was able to schedule every task others algorithms were able to, as well provided feasible schedule for those tasks in which all other algorithms failed to provide. However, the performance of Algorithm D is achieved at the cost of higher execution time. Hence, Algorithm D represents a tradeoff between execution time and performance.

## REFERENCES

- [ACR2011] Andrei S., Cheng A., Radulescu V.. Estimating the number of processors towards an efficient non-preemptive scheduling algorithm. *Proceedings of 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'11), IEEE Computer Society, Timisoara, Romania, September 26-29, 2011*
- [ACGR2010] Andrei S., Cheng A., Grigoras G., Radulescu V.. An Efficient Scheduling Algorithm for the Multiprocessor Platform. In *Proceedings of 12<sup>th</sup> International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'10)*, pages 245–252, IEEE Computer Society, Timisoara, Romania, September 26-29, 2010.
- [GMR1995] George L., Muhlethaler P., Rivierre N. (1995), Optimality and non-preemptive real-time scheduling revisited; Rapport de Recherche RR-2516, INRIA, Le Chesnay Cedex, France.
- [AND 2013] Andrei S., Lecture Notes of Real Time Systems Course (COSC-5340\_03\_3), Department of Computer Science, Lamar University.