RTOS Project Report

Emile Merian
ULB : 518697

18/12/2023

1 Introduction

The primary objective of this project is twofold. Firstly, we aim to conduct a comprehensive analysis and comparison of the success rates associated with various partitioning strategies within the context of partitioned scheduling in a multiprocessor environment. In doing so, we seek to discern the efficiency and effectiveness of different partitioning approaches.

Subsequently, our focus will shift to a comparative evaluation of the success rates observed in the context of what we identify as the most optimal partitioning strategy for a partitioned scheduler. This optimal strategy will be juxtaposed against global scheduling and edf^k schedulers, providing a comparison to their respective performances.

2 Methodology

2.1 Task Generator

The task set generator creates a task set given the following constraints :

1. It is given a number of tasks in the generated task set
2. It is given a system utilization m for which the generated task set's utilization must be included in $[m/2, m]$.
3. the execution time is generated in the range $[1, 40]$
4. the period is generated in the range $[executiontime, 200]$
5. The deadline is equal the period as we are working on implicit deadlines tasks in this project.

2.2 Schedulability Test

2.2.1 Schedulable

There no parameters implemented to deduce schedulability before simulation. Note that the partitioning of cores are counted as simulating, and therefore a partition error returns the exit code 2.

2.2.2 Non-Schedulability

2 simple parameters have been implemented to deduce non-schedulability for a given taskset, which are respectively checking if the total utilization exceeds the number of cores or if any task in the task set exceeds a utilization of 1. If any of these conditions are met, the code will return an exit code 3, as they are detected before simulation starts.

2.3 Experiments

2.3.1 Partioning

A partition is characterized by a pair (h, o) for a given heuristic h and task ordering o. Our experiment will feature 4 heuristics : first fit, next fit, best fit and worst fit, and will feature two orderings: increasing utilization and decreasing utilization. We will generate a dataset of 500 synchronous tasks sets with implicit deadlines and run every possible combinations of heuristic/ordering on them for all m cores in the range $[1, 20]$. We will record the success rates of every pairs and observe the results.

2.3.2 Scheduling

Once we determine which partitioning is the most successful, we will compare it to two mutliprocessor scheduling algorithms: global scheduling and edf^k. We will make observations in the same manner as the previous experiment.

2.4 Experimental setup and result

The experimental setup can be found in the plot.py file to plot the results and the main experiments can be found in the experiment.py file. For every values of the number of cores m, the dataset will possess tasksets containing 4 times m amount of tasks.

Note that the global scheduling and edf^k algorithms seems from the result to not have been implemented correctly. This report will therefore only cover the partitioning phase of the experiments.

For each respective experiments, we obtain the following graphs :

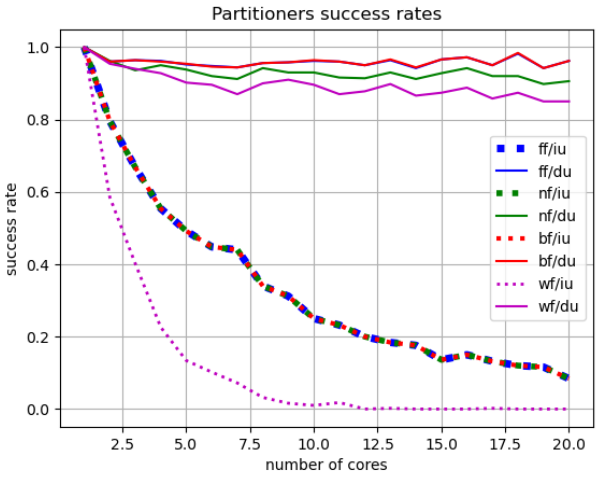

Figure 1: First Experiment result graph

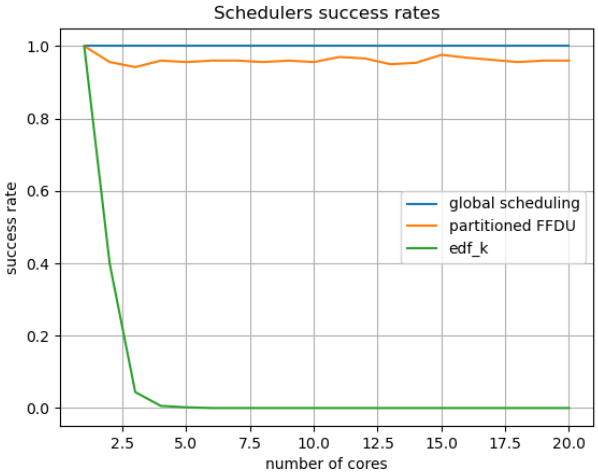

Figure 2: Second Experiment result graph (erronated)

2.5 Discussion

2.5.1 Partitioning

We can observe a few surprising observations from the result of our experiments. Firstly, we can observe that the success rate for the first fit, next fit and best fit ordered by increaasing utilization have exactly the same success rate. This

tells us that the 3 partitions will always perform exactly the same. This also tells us that there main problem is fitting increasingly more consumming tasks when the space has already been filled by smaller tasks.

We can also observe that the success rate for the first fit and best fit ordered by decreasing order are the exact same. This make sense as we can prove that when ordered by decreasing order, the two algorithms will work exactly the same, albeit first fit being a bit less complex. Indeed, lets take a small step by step example :

1. let m1, m2, m3 three cores
2. a task arrives and is fitted in m1 for both algorithms
3. a second task arrives and the following happens : either it fits in m1, so it is fitted in m1 by first fit, but it is also fitted in m1 by best fit, as m1 is currently the bin with least free space, or it doesn't fit m1 and is fitted to m2 for both algorithms.
4. a third task arrives, and we may repeat the previous step exactly the same way. Because the tasks are ordered in decreasing utilization, we know that the space for $m1 <= m2 <= m3$ in the case that the first and second tasks are respectively in m1 and m2. So if the task cannot be fitted in m1, m2 is the next chosen bin for both algorithms, and then it is m3.
5. We notice that in decreasing ordering, the best fits are ordered from first bin to last for the biggest tasks of the taskset.

Finally, the best performing algorithms are the first fit and best fit ordered by decreased utilization and the worst performing algorithm is the worst fit algorithm ordered by increasing utilization. This result is not surprising as fitting the most costly tasks first allows us to make sure we minimize the space we waste when fitting a core with free space that cannot accomodate for the tasks still left to be fitted.