An Empirical Investigation of Eager and Lazy Preemption Approaches in Global Limited Preemptive Scheduling

Abhilash Thekkilakattil[1], Kaiqian Zhu[1], Yonggao Nie[1], Radu Dobrin[1] and Sasikumar Punnekkat[2]

[1]Mälardalen Real-Time Research Center, Mälardalen University, Västerås, Sweden
[2]Birla Institute of Technology and Science, Goa, India

Abstract. *Global limited preemptive real-time scheduling in multiprocessor systems using Fixed Preemption Points (FPP) brings in an additional challenge with respect to the choice of the task to be preempted in order to maximize schedulability. Two principal choices with respect to the preemption approach exist 1) the scheduler waits for the lowest priority job to become preemptible, 2) the scheduler preempts the first job, among the lower priority ones, that becomes preemptible. We refer to the former as the Lazy Preemption Approach (LPA) and the latter as the Eager Preemption Approach (EPA). Each of these choice has a different effect on the actual number of preemptions in the schedule, that in turn determine the runtime overheads.*
In this paper, we perform an empirical comparison of the run-time preemptive behavior of Global Preemptive Scheduling and Global Limited Preemptive Scheduling with EPA and LPA, under both Earliest Deadline First (EDF) and Fixed Priority Scheduling (FPS) paradigms. Our experiments reveal interesting observations some of which are counter-intuitive. We then analyse the counter-intuitive observations and identify the associated reasons. The observations presented facilitate the choice of appropriate strategies when using limited preemptive schedulers on multiprocessor systems.

1 Introduction

Real-time computing systems are increasingly being used in modern mission and safety critical systems. In a real-time system, the events occurring in the environment are typically handled by a set of real-time tasks, with the requirement that the task executions must complete by their respective deadlines. A real-time scheduling algorithm ensures the timely execution of these real-time tasks. Real-time scheduling theory has traditionally focused on fully preemptive and fully non-preemptive scheduling of real-time tasks on uniprocessor [8] and multiprocessor platforms [1]. However, both these paradigms can become prohibitively expensive when considering the effects of preemption related overheads and blocking. Preemptively scheduling real-time tasks implies context switch overheads, cache related preemption and

migration delays, increased bus contention and pipeline delays. Non-preemptive scheduling, on the other hand, can be infeasible at arbitrarily small utilizations [19].

The multi-core revolution has revived the interest among researchers and practitioners, particularly in the field of real-time embedded systems, to leverage on the ability of multiprocessing platforms to provide higher performance. In this paper, we focus on global scheduling on a multiprocessor platform that can be broadly classified into fixed task priority, fixed job priority and dynamic job priority scheduling algorithms. Note that preemptive scheduling on multiprocessor platforms using the global approach also implies that resuming tasks may be migrated to other processors. In this paper, we consider Global Preemptive Fixed Priority Scheduling (G-P-FPS) which is a fixed task priority scheduling paradigm, and Global Preemptive Earliest Deadline First (G-P-EDF) scheduling which is a fixed job priority scheduling paradigm. In general, EDF incurs less preemptions than FPS since EDF assigns priorities according to absolute deadlines, because of which preemptions occurring towards the end of the task executions, specifically due to higher priority tasks having later deadlines, are avoided (and no new ones occur) [9].

High preemption and migration related overheads are an emerging issue in many real-time applications [2]. Recent experiments show that, among the various operating system data structures, the stack is the most susceptible to faults [4], and that preemptive scheduling makes the system significantly susceptible to faults [19]. Moreover, there is a clear link between the run-time preemptions on higher criticality tasks and the associated actual execution times that in turn decide the scheduling of lower criticality tasks in mixed criticality systems (see [7] for an overview). The foundation of mixed-criticality systems is based on the premises that tasks have different levels of assurances on their WCETs depending on their criticality [7]. Higher criticality tasks which have/require high levels of assurances, typically have over-approximated WCETs to account for the different unpredictable overheads, a significant one being the overhead associated with preemptions. At any given time instant the system is assumed to be executing at a criticality level L, which is given by the criticality of the currently executing lowest criticality tasks. Every task in the system has a budgeted time for the criticality level L, the overrun of which triggers a *criticality switch* during which all tasks with criticality L or lower are discarded. It is clear that the schedulability of lower criticality tasks depends on the actual execution time of higher criticality tasks [7]. If the higher criticality tasks are preempted very often, the probability of the system switching to a higher criticality level is high since there is a greater chance for the higher criticality tasks to overrun their budgeted time for that criticality level because of the associated preemption related overheads.

One way of minimizing preemption overheads while controlling the effects of blocking is to limit preemptions to predefined locations in the tasks (see [10] for a survey). Among the different methods to limit preemptions, fixed preemption points [6] enable offline analysis since the points of preemptions are

known beforehand. Limited preemptive scheduling on multiprocessor platforms presents an additional problem with respect to the choice of the running lower priority tasks to be preempted. The scheduler can choose to wait for the lowest priority task to become preemptible or preempt the first lower priority running task that becomes preemptible. The former is a Lazy Preemption Approach (LPA) and the later is an Eager Preemption Approach (EPA).

Each approach may have a different effect on the actual number of preemptions at run-time, that in turn determines the overheads in the schedule. Consequently, there is a need for a thorough study of the preemptive behavior of the different approaches. Note that we consider tasksets in which the optimal fixed preemption points are already known. Consequently, the overheads depend on whether or not a preemption actually occurs at these points. In other words, since preemptions are possible only at these optimal preemption points, the system performance will depend on the actual number of preemptions instead. In this paper, we investigate 1) Which approach among EPA and LPA generates fewer number of preemptions at run-time? 2) Which scheduling paradigm among G-LP-FPS and G-LP-EDF generates fewer number of preemptions at run-time? We make several observations, and in particular show that limited preemptive scheduling on multiprocessors may generate more preemptions than fully preemptive scheduling.

The rest of the paper contains the system model in Section 2, some background in section 3 experiments in Section 4, and analysis of results in Section 5. Finally, we present the conclusions in Section 6.

2 System Model

We consider a set of n sporadic real-time tasks $\Gamma = \{\tau_1, \tau_2, \ldots \tau_n\}$ scheduled on m identical processors. Each task τ_i is characterized by a minimum inter-arrival time T_i, and a relative deadline $D_i \leq T_i$, and is assumed to contain $q_i \geq 0$ optimal preemption points specified by the designer/programmer [18]. Let $b_{i,j}$, $j = 1...(q_i + 1)$ denote the worst case execution time of task τ_i between its $(j-1)^{th}$ and j^{th} preemption points. We use the notation $b_{i,j}$ to also refer to the corresponding Non-Preemptive Region (NPR). The Worst Case Execution Time (WCET) of each task τ_i can be calculated as $C_i = \sum_{j=1}^{q_i+1} b_{i,j}$.

3 Global Limited Preemptive Scheduling

In this section, we describe the two approaches to preemption under global LP scheduling *viz.* the eager and lazy preemption approaches. Please note that in all the figures in this paper an up arrow represents the release time and a down arrow represents the deadline of the associated task.

3.1 Lazy preemption approach

In the Lazy Preemption Approach (LPA), if a higher priority task is released and all lower priority jobs are executing their NPRs, the scheduler waits for

the lowest priority job to complete its NPR (*i.e.,* to become *preemptible*) before allowing the higher priority job to preempt [5][3]. We illustrate this approach using the following example illustrated in Figure 1.

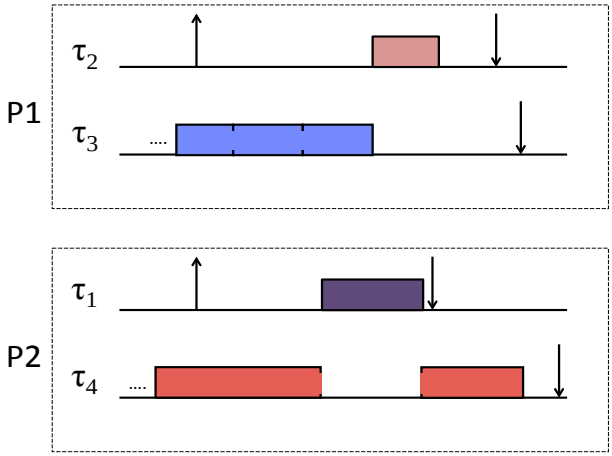

Fig. 1: Example for lazy preemption approach.

Example 1. Consider four tasks τ_1, τ_2, τ_3 and τ_4, where τ_1 has the highest priority and τ_4 has the lowest, scheduled on 2 processors $P1$ and $P2$. Consider the scenario in figure 1 in which τ_1 and τ_2 are released during the execution of NPRs of τ_3 and τ_4 as shown. If the scheduling algorithm used is G-LP-FPS with lazy preemptions, τ_1 and τ_2 will be blocked until τ_4 finishes executing its NPR, after which τ_1 is scheduled on $P2$. However, τ_2 cannot be scheduled because τ_3 has already started executing its NPR at this point. Once τ_3 completes execution of its NPR, τ_2 is allowed to execute on $P1$. Although we have considered tasks with fixed priorities, it can be easily seen from the example that the same conclusions can be made about the preemptive behavior under G-LP-EDF with lazy preemption.

3.2 Eager preemption approach

Under the Eager Preemption Approach (EPA), if a higher priority task is released and all lower priority executing jobs are executing their NPRs, the scheduler preempts the first lower priority that becomes *preemptible*. We illustrate this approach using the following example that is illustrated in Figure 2.

Example 2. Consider the same four tasks τ_1, τ_2, τ_3 and τ_4 presented in example 1, and the same scenario in which τ_1 and τ_2 are released during the execution of NPRs of τ_3 and τ_4. If the scheduling algorithm used is G-LP-FPS

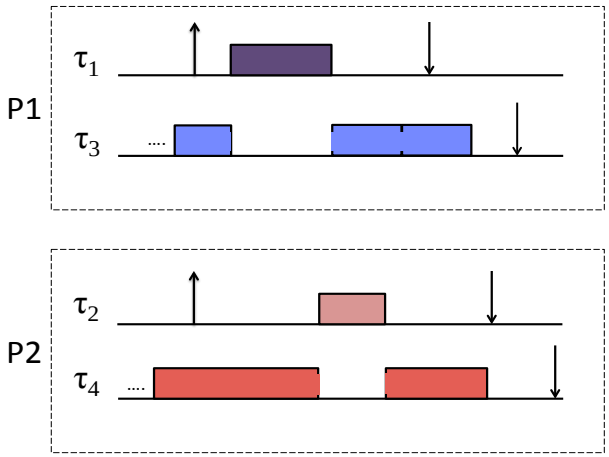

Fig. 2: Example for eager preemption approach.

with eager preemptions, the scheduler will allow τ_1 to preempt τ_3 rather than wait for τ_4 since it is the first available opportunity to preempt (see figure 2). Once τ_4 completes its NPR, τ_2 is scheduled on $P2$. Although we have considered tasks with fixed priorities, the same conclusions can be made about the preemptive behavior under G-LP-EDF with lazy preemption.

3.3 Related Works

All of the major works on *multiprocessor* limited preemptive scheduling [14][12][16][3][21][20][11] focused on schedulability and not the preemptive behavior. We clarify that in our previous work [21] we considered schedulability under (only) limited preemptive FPS with Fixed Preemption Points (FPP). In this paper, we instead consider the "preemptive behavior" of both limited preemptive EDF and FPS under FPP. Marinho [15] presented some observations regarding the preemptive behavior under EPA and LPA; however he did not present detailed empirical comparisons. Previously, a preliminary study of the preemptive behavior of G-LP-FPS [17] and that of G-LP-EDF [23] was conducted with respect to the number of preemptions under both EPA and LPA. In this paper, we perform a comprehensive comparison of the preemptive behavior using a weighted metric similar to weighted schedulability [2] and vary different parameters that may have an impact on the number of preemptions. To our knowledge, this is the first such effort towards investigating the runtime preemptive behavior of limited preemptive scheduling on multiprocessors.

4 Empirical Investigation of the Preemptive Behavior

In this paper, we investigate how the decision of choosing a scheduling paradigm *viz.* EDF and FPS, and the preemption approach *viz.* EPA and LPA affect the number of preemptions at run-time. The number of run-time preemptions influence the preemption overheads in the schedule, which in turn influence, among others, schedulability of lower criticality tasks in mixed criticality systems and resource contention. Note that the performance of different combinations of schedulers and the approaches to preemption differ based on the underlying hardware and the specific application (task parameters, cache access patterns etc). As an exhaustive study on real cases that allows us to sufficiently generalize our results is resource and time demanding, we had to restrict the current experiments to the use of synthetic tasksets. A more detailed version of this paper can be found online at [22].

Method for the Experimental Design: The experimental methodology is similar to the one adopted by Buttazzo [9]. In order to perform the experiments, due to the very limited availability of simulators implementing limited preemptive scheduling under eager and lazy preemption approaches, we developed a simulator that takes as input the task parameters and generate the different schedules for a user defined time duration. The task parameters were generated using well accepted techniques and is described in the following: We used the UUnifast-discard algorithm [13] to generate individual task utilizations that were varied between U_{min} and U_{max}. For the case of FPS, we adopted the deadline-monotonic priority assignment– note that we are interested in the number of preemptions and not deadline misses. The scheduling algorithms, whose preemption behaviors we want to compare are in fact incomparable with respect to schedulability [14]. Therefore, to investigate the preemptive behavior, building on the speed-up bounds [1] and schedulability experiments, *e.g.,* [5], that indicate high schedulability for tasksets that utilize up to 50% of the platform under both preemptive EDF and FPS, we set, $U_{max} = \frac{m}{2}$. Note that a fully preemptive schedule can be obtained using a limited preemptive scheduler by enabling preemptions after every unit of execution, therefore, if a taskset is preemptively schedulable, it is also LP schedulable. However, in one set of the experiments, we consider tasksets with utilization up to 100% of the processing platform in order to investigate the preemptive behavior for high utilization tasksets. The task periods were randomly generated between $T_{min} = 5$ and $T_{max} = 500$ (this represents tasks with periods $5 - 500ms$ as found in many typical real-time systems), and execution times were computed using the generated utilizations. We assumed deadlines to be equal to periods; note that this assumption does not affect the generality of the results since we consider the preemptive behavior and not schedulability. The largest NPR of each task was generated as a percentage of its execution time, with the ceiling function applied to obtain integer values (*i.e.,* in case of a decimal NPR we approximate it to the smallest integer greater than the decimal number)– in our experiments, this

was set to 10% (*i.e.,* each task has no more than 9 preemption points). Note that we also vary the NPR lengths in one of the experiments. We counted the number of preemptions generated for one hundred tasksets under each of the following paradigms: 1) G-P-FPS 2) G-P-EDF 3) G-LP-FPS with eager preemptions (EPA-FPS) 4) G-LP-FPS with lazy preemptions (LPA-FPS) 5) G-LP-EDF with eager preemptions (EPA-EDF) 6) G-LP-EDF with lazy preemptions (LPA-EDF), by simulating the respective schedules for a duration of 10000 time units.

Weighted Metric: In order to understand how the number of preemptions vary with a second parameter, *e.g.,* number of tasks, in addition to utilization, we adopted a weighted measure similar to weighted schedulability [2]. We weighted the number of preemptions N_i, for a taskset Γ_i with respect to a parameter p over a simulation run for Δ time units, with the taskset utilization U_i as follows:

$$W_p = \frac{\sum_{\forall \Gamma_i} U_i N_i}{\sum_{\forall \Gamma_i} U_i}$$

We investigated how the number of preemptions in the actual schedule varies with 1) total utilization 2) number of processors 3) number of tasks and 4) length of the NPR.

4.1 Varying the Total Utilization

In this set of experiments, we investigated the preemptive behavior of the scheduling algorithms for increasing utilizations. We considered a 4 processor platform and generated tasksets with 25 tasks and utilizations ranging from 1 to 4. We calculated the average number of preemptions per 100 time units, after simulating the schedule for a duration of 10000 time units– the results are reported in Figure 3. We observe that G-LP-EDF with EPA generates the maximum number of preemptions that is greater than G-LP-FPS with eager preemptions. Perhaps surprisingly, G-P-EDF and G-P-FPS generate fewer preemptions than their limited preemptive counterparts with eager preemptions. Moreover, we observed that the uniprocessor behavior of EDF with respect to generating fewer number of preemptions than FPS [9] generalizes to the multiprocessor case; G-P-EDF generated less preemptions than G-P-FPS. The least number of preemptions is generated by G-LP-FPS with LPA that generated slightly fewer number of preemptions even when compared to G-LP-EDF with LPA.

In the following, we conduct experiments with varying number of processors after adopting the weighted metric described above.

4.2 Varying Number of Processors

We generated tasksets with 25 tasks having utilizations ranging from 1 to $U_{max} = m/2$ for $m = 4$ to $m = 20$ in steps of 2. The results of the experiments are

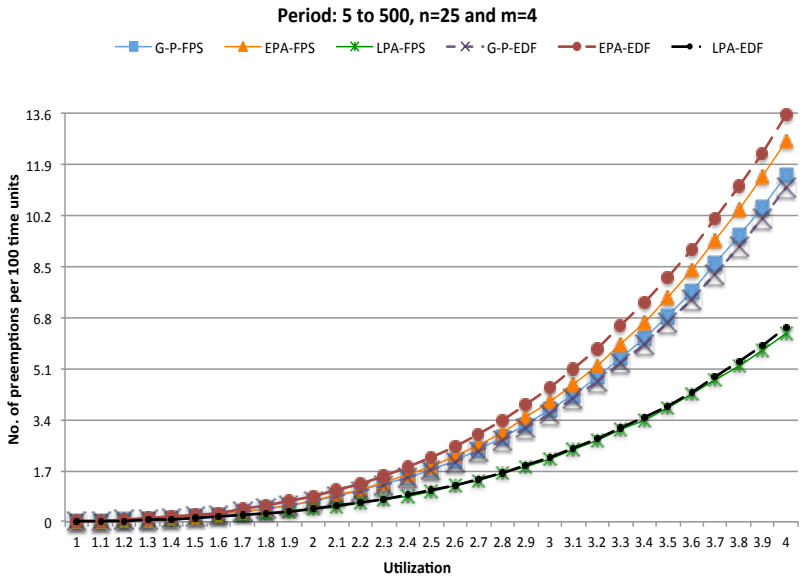

Fig. 3: Average number of preemptions per 100 time units under varying utilizations for NPR length=10% of WCET.

reported in Figure 4. We can see that the number of preemptions, in general, decreases with increasing number of processors (since more processors for the same number of tasks imply a reduced need for preemption). The interesting observation here is that G-LP-EDF with EPA generated more preemptions than G-LP-FPS with EPA. The fully preemptive variants of EDF and FPS generated fewer preemptions than their limited preemptive counterparts with EPA. The least number of preemptions are generated by G-LP-EDF and G-LP-FPS; both under LPA. Here again, G-LP-FPS with LPA generates slightly fewer number of preemptions compared to G-LP-EDF with LPA. The use of the weighted metric described above makes it less visible from the graph (note: G-LP-EDF with EPA generates significantly more preemptions than G-LP-FPS with EPA).

4.3 Varying Number of Tasks

We varied the number of tasks per taskset from $n = 6$ to $n = 26$ in steps of 2 and counted the number of preemptions for tasksets with utilizations between 1 and $U_{max} = m/2$. The results are reported in Figure 5. We observed a similar trend observed by Buttazzo [9]. The fully preemptive variant of EDF generated fewer preemptions than G-P-FPS. Moreover, the number of preemptions increased with increasing number of tasks. We expect that increasing the number of tasks further will lead to a decrease in the number of preemptions because individual task execution times will decrease to keep the utilization constant as noted by

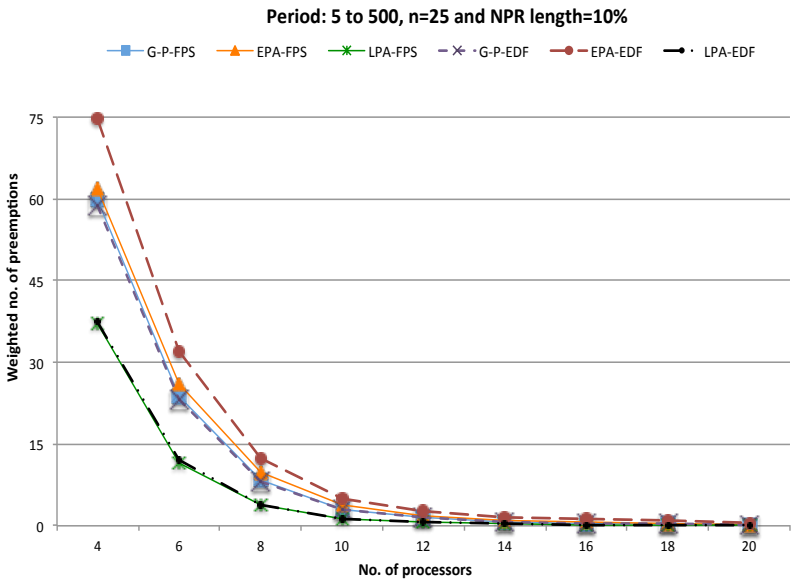

Fig. 4: Weighted number of preemptions under varying number of processors.

Buttazzo [9]– the decreasing trend is observable when number of tasks change from 22 to 26.

We can see trends that are similar to the one observed in the previous experiments reported in this paper: G-LP-EDF with EPA generated most preemptions while G-LP-FPS (together with G-LP-EDF) with LPA generated the least. Here again G-LP-FPS with LPA generated slightly fewer number of preemptions than G-LP-EDF with LPA (not visible due to scaling issues– see appendix of [22]). Also, note that G-LP-FPS with EPA generated significantly fewer preemptions than G-LP-EDF with EPA.

4.4 Varying Length of Non-preemptive Regions

Lastly, we wanted to investigate if the preemption behavior would be any different if we had chosen a different size for the non-preemptive regions. We considered a 4 processor platform and counted the number of preemptions generated when the NPR lengths changed from 5% to 100% of the task WCETs (no. of tasks per taskset=25). The experimental results are reported in Figure 6. The graph indicates that when using EPA (under EDF or FPS), the number of preemptions increases if the length of the NPRs increase without decreasing the number of preemptions points. This is observed by the increased number of preemptions when the NPR lengths increase from 35% to 40% (the number of preemptions remains unchanged at 2) and from 50% to 80% (when the number of preemptions remains unchanged at 1). Once past 80%, most of

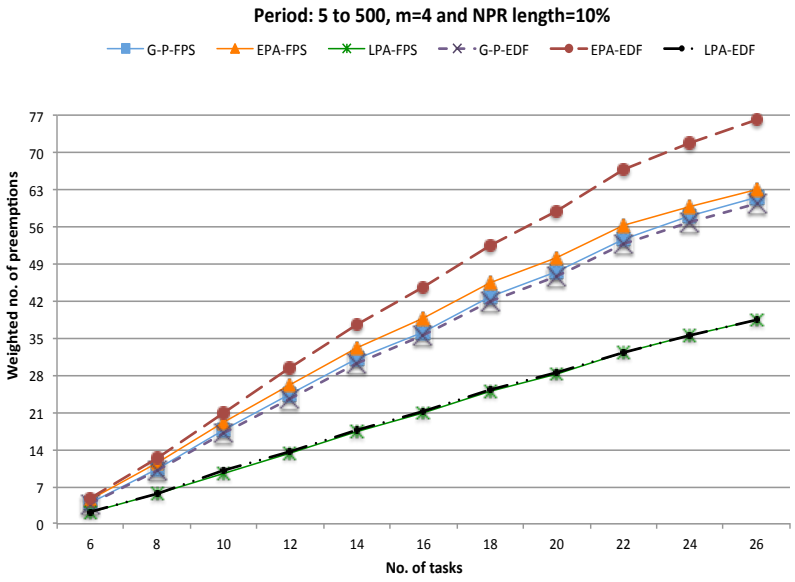

Fig. 5: Weighted number of preemptions under varying number of tasks.

the tasks become non-preemptive since we apply the ceiling function, and hence there is a decrease in the number of preemptions. Similar trends are also observed in the case of EPA although less pronounced.

In all the cases, LPA outperformed all the other variants of the scheduling algorithms. Moreover, G-LP-EDF with EPA continued to have the highest number of preemptions for shorter NPR lengths, but showed similar performance as G-LP-FPS with EPA for larger NPR lengths (from around 45% as seen from Figure 6). Notably, for NPR lengths larger than 20%, EPA (under both EDF and FPS) generated fewer preemptions than the fully preemptive counterparts. An enlarged version of the above graph available in the Appendix of [22] illustrates that for shorter NPR lengths, G-LP-FPS generated fewer preemptions than G-LP-EDF with both EPA and LPA, while for larger NPR lengths EDF fares better. However, we would like to clarify that this is observed only for NPR lengths larger than 50% (*i.e.,* for tasks with a single preemption point). If there are more preemption points, then clearly FPS outperforms EDF. Moreover, note that very large NPR lengths may imply unschedulability due to the "long task problem" [19].

Note that graphs 4 to 6 shows the *weighted number of preemptions* as described above, instead of directly showing the number of preemptions.

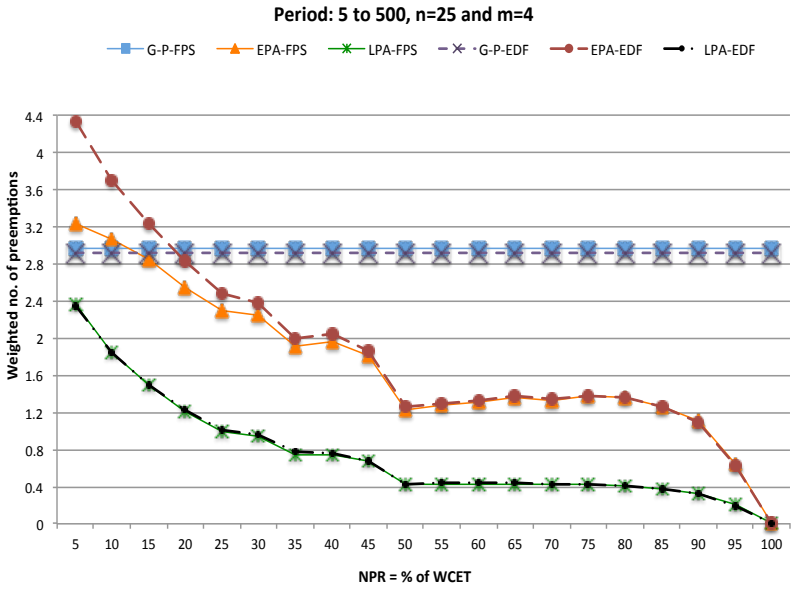

Fig. 6: Weighted number of preemptions under varying NPRs.

5 Analysis of the Experimental Results

In this section, we discuss the (counter-intuitive) experimental results in detail and identify reasons behind the observed behaviors.

5.1 Preemptive scheduling vs. limited preemptive scheduling with EPA and LPA

As seen from the evaluations presented in Section 4, global LP scheduling with eager preemptions generates more preemptions than global fully preemptive scheduling. This is because more preemptions are required to resolve the priority inversions occurring due to the eager preemption of the lower priority executing task (not necessarily the lowest) that first completes executing its NPR. This is detailed in the following.

When using the eager preemption approach, if the first executing lower priority task that becomes preemptible is preempted by a higher priority job, what essentially happens is that the higher priority task *transfers* the priority inversion to the preempted task if it is not the lowest priority one (since there are lower priority tasks still executing on other processors). The preempted task, which is in the ready queue, may preempt another lower priority task that first completes its NPR (again not necessarily the lowest priority one) thereby transferring priority inversion. This could potentially continue like a domino effect until no more priority inversion exists in the system. In order to

resolve priority inversion for each task, there is a need for preemption, consequently increasing the number of preemptions. The absence of the above described domino effect explains the better performance of global LP scheduling with lazy preemptions when compared to eager preemptions. These are clarified in the following using a simple example.

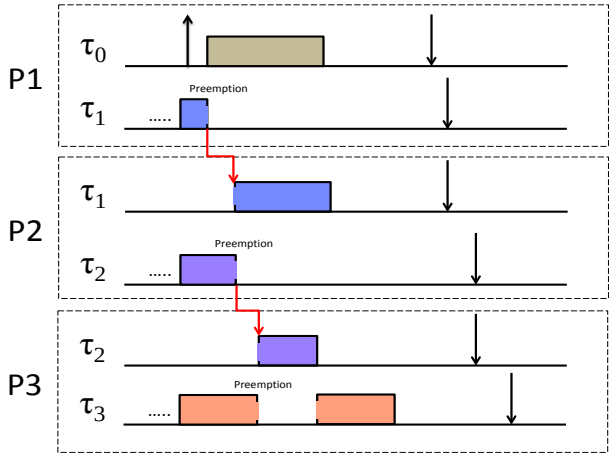

Fig. 7: Preemptions under Eager Preemption Approach.

Example 3. Consider the scenario presented in Figure **??** *in which* 3 *tasks* τ_1, τ_2 *and* τ_3, *with priorities in the order* $\tau_1 > \tau_2 > \tau_3$, *are executing on* 3 *processors. Suppose a higher priority task* τ_0 *is released. If the scheduler used is a fully preemptive scheduler,* τ_0 *will preempt* τ_3 *resulting in only a single preemption.On the other hand, under LP scheduling with eager preemptions,* τ_0 *will preempt the first task that becomes preemptible, in this case* τ_1 *(as seen from Figure 7). Note that* τ_1, *has the highest priority among the executing ones. Consequently, there is a priority inversion on* τ_1 *since the other processors are executing lower priority tasks. The task* τ_1 *will wait for one of the lower priority tasks to become preemptible. In our scenario,* τ_2 *is the next task that becomes preemptible first. Consequently,* τ_2 *will be preempted by* τ_1. *However, the priority inversion persists because* τ_2, *which was preempted, has a higher priority than* τ_3 *that is still executing on* $P3$, *and hence there is one more preemption. Note that the number of preemptions in this case is* 3 *instead of* 1 *under fully preemptive scheduling. In the same scenario, under global LP scheduling with lazy preemption, the scheduler will preempt the lowest priority job* τ_3 *and hence the domino effect described earlier will not occur (as seen from Figure 8). Global LP scheduling with lazy preemption performs better than fully preemptive scheduling since the upper-bound on the number of preemptions in a task is determined by the number of preemption points instead of the number of higher priority task releases (that can be significantly larger).*

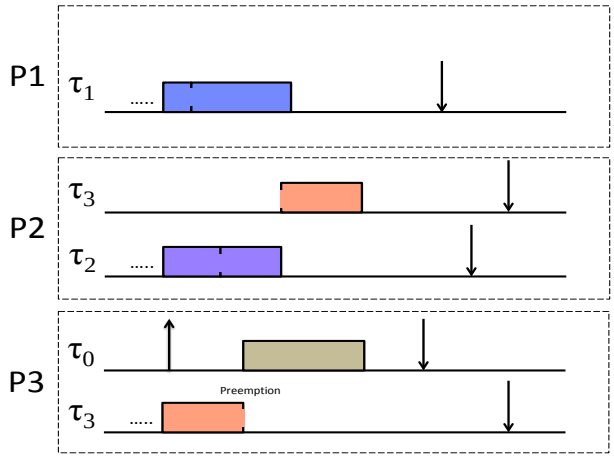

Fig. 8: Preemptions under Lazy Preemption Approach.

The consequences of such a domino effect under EPA on the total number of observed preemptions can be severe on platforms with large number of processors since in the worst case m such priority inversions need to be resolved per high priority release which can be potentially very large especially in many-core systems.

5.2 Global Limited Preemptive FPS vs. EDF

In general, preemptive EDF generates fewer number of preemptions than preemptive FPS [9]. This is because, many of the preemptions necessitated by the fixed priority task assignment, do not occur under EDF. The preemptions required by task releases that may have a higher priority under an FPS scheme, towards the end of the execution of tasks, that may have a lower priority under an FPS scheme, are avoided under EDF because of the deadline based priority ordering. On the other hand, in this scenario under FPS, a preemption occurs. Similarly, it is easily seen that uniprocessor limited preemptive EDF generates fewer preemptions than limited preemptive FPS. As seen from the experiments, similar to the uniprocessor case, G-P-EDF performs better in reducing the actual number of preemptions at run-time when compared to G-P-FPS.

However, under *limited preemptive scheduling on multiprocessors*, G-LP-EDF generates more preemptions than G-LP-FPS (under both EPA and LPA). The reason is that, EDF priority ordering generates more priority inversions consequently "forcing" eager preemptions. For example, under G-LP-FPS with LPA, higher priority tasks released during the execution of the final NPR of the lowest priority task wait for it to complete. This does not happen under EDF since at least one of the executing jobs may have a larger absolute deadline and hence can be preempted. We clarify the reason for the relatively poor performance of G-LP-EDF using the following example.

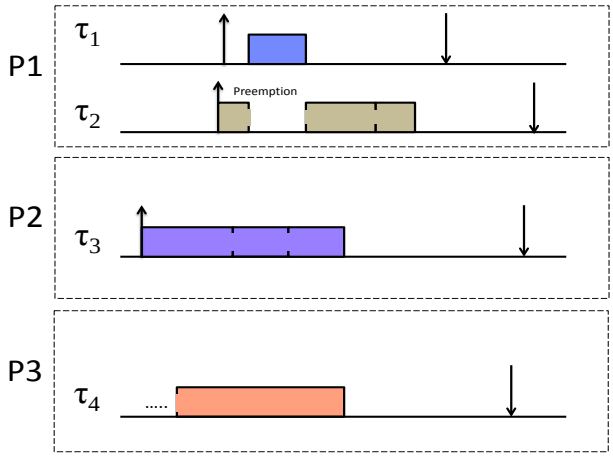

Fig. 9: Preemptive behavior of G-LP-EDF.

Example 4. In this example, consider three tasks τ_2, τ_3 and τ_4 that are currently executing on 3 processors and another task τ_1 that is released during their execution as illustrated in Figure 9. Assume that τ_4 has started executing its final NPR and the tasks have time periods $T_1 < T_2 < T_3 < T_4$. Under G-LP-EDF using lazy preemptions, when τ_1 is released, τ_2 is the task with the latest deadline and hence has the lowest priority; therefore τ_1 preempts τ_2. On the other hand under G-LP-FPS with lazy preemptions (as shown in Figure 10), assuming rate monotonic priority ordering, τ_4 has the lowest priority and hence the scheduler waits for the final NPR of τ_4 to complete instead of preempting τ_2 (and hence requiring no preemption). When considering EPA, under G-LP-EDF with EPA, τ_1 will preempt τ_3 since it is the first preemption point available. Now since τ_3 has an earlier absolute deadline than τ_2, τ_3 will preempt τ_2 at the next preemption point of τ_2. On the other hand, under G-LP-FPS with EPA, τ_1 preempts τ_3, but τ_3 does not preempt τ_2 due to its fixed (higher) priority when compared to τ_3.

Therefore, for applications in which run-time preemptions are directly or indirectly harmful, such as in the case of safety-critical system or energy constrained systems, it is best to use G-LP-FPS since it generates fewer number of preemptions than G-LP-EDF at runtime.

6 Conclusions and Future Work

In this paper, we empirically investigated the preemptive behavior of G-LP-EDF and G-LP-FPS under eager and lazy preemption approaches, along with G-P-FPS and G-P-EDF, varying a wide range of parameters. Our investigations reveal a number of interesting observations with respect to the observed number of preemptions under the different paradigms. In particular:

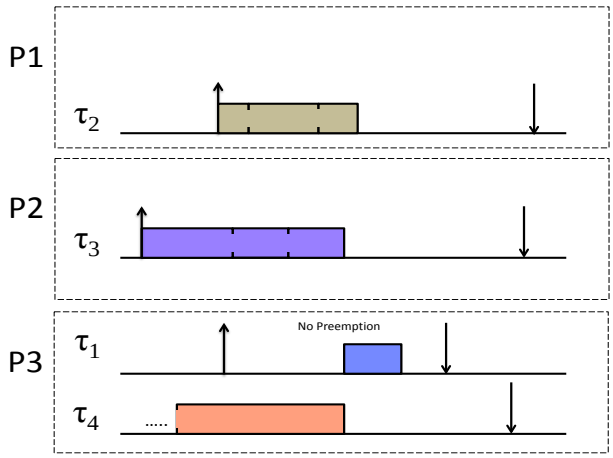

Fig. 10: Preemptive behavior of G-LP-FPS.

1. Limited preemptive scheduling on multiprocessors does not necessarily reduce the number of preemptions; in fact with an eager preemption approach, the number of preemptions could be larger than in the case of fully preemptive scheduling, as well as global LP scheduling with LPA.
2. We show that the well known observation regarding the preemptive behavior of EDF on uniprocessors generalizes to multiprocessors; G-P-EDF generates fewer preemptions than G-P-FPS.
3. We also show that the reduction in preemptions observed with EDF on uni- and multiprocessors, however, does not generalize to global limited preemptive scheduling; G-LP-EDF suffers from more preemptions than G-LP-FPS.
4. Our experiments show that G-LP-FPS with LPA suffers from the least number of preemptions.

Future work include studies on a real hardware and trade-offs involving preemption point placement, schedulability and approach to preemption *viz.* EPA and LPA.

References

1. S Baruah, M Bertogna, and G Buttazzo. *Multiprocessor Scheduling for Real-Time Systems*. Springer International Publishing, 2015.
2. A Bastoni, B.B Brandenburg, and J.H Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *The International Workshop OSPERT*, 2010.
3. A. Block, H. Leontyev, B.B. Brandenburg, and J.H. Anderson. A flexible real-time locking protocol for multiprocessors. In *The 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2007.

4. C. Borchert, H. Schirmeier, and O. Spinczyk. Generative software-based memory error detection and correction for operating system data structures. In *The 43rd International Conference on Dependable Systems and Networks*, June 2013.
5. Björn B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, University of North Carolina at Chapel Hill, 2011.
6. R.J. Bril, J.J. Lukkien, and W.F.J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Systems*, 2009.
7. A Burns and R Davis. Mixed criticality systems - a review. In *http://www-users.cs.york.ac.uk/ burns/review.pdf (accessed: 31 Jul 2015)*.
8. G. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag, 2004.
9. G. Buttazzo. Rate monotonic vs. EDF: Judgment day. In *Real-time Systems*, 2005.
10. G.C. Buttazzo, M. Bertogna, and G. Yao. Limited preemptive scheduling for real-time systems: A survey. *The IEEE Transactions on Industrial Informatics*, 2012.
11. B Chattopadhyay and S Baruah. Limited-preemption scheduling on multiprocessors. In *The 22nd International Conference on Real-Time Networks and Systems*. ACM, 2014.
12. R Davis, A Burns, J Marinho, V Nelis, S Petters, and M Bertogna. Global fixed priority scheduling with deferred pre-emption. In *The International Conference on Embedded and Real-Time Computing Systems and Applications*, 2013.
13. R I. Davis and A Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 2011.
14. R I. Davis, A Burns, J Marinho, V Nelis, S M. Petters, and M Bertogna. Global and partitioned multiprocessor fixed priority scheduling with deferred preemption. *ACM Transactions on Embedded Computing Systems*, 2015.
15. Jos Manuel Silva dos Santos Marinho. Real-time limited preemptive scheduling. In *PhD thesis FEUP Porto*, 2015.
16. J Marinho, V Nelis, S Petters, M Bertogna, and R Davis. Limited pre-emptive global fixed task priority. In *The Real-time Systems Symposium*, 2013.
17. Y Nie. Limited-preemptive fixed priority scheduling of real-time tasks on multiprocessors. In *Master thesis, Malardalen University*, 2015.
18. Bo Peng, Nathan Fisher, and Marko Bertogna. Explicit preemption placement for real-timeconditional code. In *The Euromicro Conference on Real-Time Systems*, July 2014.
19. M Short. The case for non-preemptive, deadline-driven scheduling in real-time embedded systems. In *Lecture Notes in Engineering and Computer Science: Proceedings of the World Congress on Engineering*, 2010.
20. A Thekkilakattil, S Baruah, R Dobrin, and S Punnekkat. The global limited preemptive earliest deadline first feasibility of sporadic real-time tasks. In *The 26th Euromicro Conference on Real-Time Systems*, July 2014.
21. A Thekkilakattil, R Davis, R Dobrin, S Punnekkat, and M Bertogna. Multiprocessor fixed priority scheduling with limited preemptions. In *The 23rd International Conference on Real-Time Networks and Systems*, 2015.
22. Abhilash Thekkilakattil, Kaiqian Zhu, Yonggao Nie, Radu Dobrin, and Sasikumar Punnekkat. An empirical investigation of eager and lazy preemption approaches in global limited preemptive scheduling. Technical report, September 2015 http://www.es.mdh.se/publications/4057-.
23. K Zhu. Limited-preemptive edf scheduling of real-time tasks on multiprocessors. In *Master thesis, Malardalen University*, 2015.