Multi-Criteria Evaluation of Partitioned EDF-VD for Mixed-Criticality Systems Upon Identical Processors

Paul Rodriguez
Université Libre de Bruxelles
ECE Paris
paurodri@ulb.ac.be

Laurent George
University of Paris-Est
LIGM / ECE Paris
lgeorge@ieee.org

Yasmina Abdeddaïm
University of Paris-Est
LIGM / ESIEE
yasmina.abdeddaim@esiee.fr

Joël Goossens
Université Libre de Bruxelles
joel.goossens@ulb.ac.be

***Abstract*—In this paper, we consider the partitioned EDF-VD scheduling problem of mixed critical systems with two criticality levels (LO and HI) on identical multiprocessors. Partitioned scheduling is an NP-hard problem that has been widely studied in the literature. The most common metaheuristic to solve partitioning problems consists in ordering tasks by a given criteria (such as task utilization) then assign tasks to processors in that order, choosing which processor using a heuristic rule such as First Fit or Best Fit. The current state of the art results show that First Fit Decreasing Density provides the best success ratio for single-criticality scheduling. In the context of mixed-criticality, we would like to investigate whether this is also true for assigning LO and HI critical tasks to processors. We consider two cases, one called "criticality aware" that first tries to assign HI tasks to processors and then LO tasks separately and the other one called "criticality unaware" that assigns tasks without taking their criticality into account. We test the performance of all combinations of sorting/partitioning heuristics in both cases, which leads to 1024 different heuristics in the aware case and 32 in the unaware case. We define two search algorithms to efficiently find which of these heuristics obtains the best success ratio. In addition, a new mixed-criticality multiprocessor random task set generation algorithm is proposed.**

I. Introduction

In this paper, we consider the problem of multiprocessor scheduling of mixed critical periodic tasks in the dual criticality case (LO and HI). In the multiprocessor case, two main paradigms have been considered for the scheduling of real-time tasks: the global scheduling and the partitioning approaches.

The global scheduling approach allows a job to migrate during its execution. At any time, a job is run only by one processor but the scheduler can decide to migrate it to another processor. The global scheduling approach provides better feasibility bounds but does not take into account job migrations costs.

In this paper, we consider the Partitioned scheduling problem. With the partitioning approach, the feasibility analysis on a multiprocessor requires to solve two problems:

- First, find a partitioning algorithm according to a partitioning heuristic.
- Second, use a uniprocessor feasibility condition on each processor to decide on the schedulability of the task set.

The partitioning approach therefore consists in statically assigning tasks to processors and then in solving the feasibility problem for a given partitioning on each processor. The problem of finding a feasible partitioning is a bin packing problem known to be NP-hard in the strong sense [1].

The partitioned approach received much more attention in the industry than the global one as it is a natural extension of uniprocessor scheduling. Partitioning has the advantage of utilizing all the feasibility results of uniprocessor scheduling [2] but also introduces some pessimism. In pathological cases, a partitioned system could be unfeasible with utilization just above 50% of the platform's capacity but simulation results [3] show that partitioned scheduling offers good success ratio (the ratio of successfully scheduled task sets over all task sets considered) even for high utilization. Furthermore, partitioned scheduling remains the industry standard in multiprocessor real-time systems and is for this reason still an important research subject, especially when all tasks do not have the same criticality. This means that the system may be subject to various certification processes, which are only concerned with the validation of a subset of the functionalities.

Furthermore, the certification processes are carried out using analysis methods whose rigor depends on the criticality of the tasks that need to be certified. Mixed-criticality systems are an attempt to model systems that need to be certified at various levels of assurance. In practice, a task that is subject to multiple certification processes will be characterized by multiple estimations of its Worst-Case Execution Time (WCET), some of which are more pessimistic than others. This reflects the differences in rigor adopted by the certification authorities.

The more a certification authority wants to ensure a task will never exceed its WCET, the more conservative its estimation will be. Nevertheless, when certifying that a task will meet its constraints, the assurance level that is used for other tasks is equal to the criticality of that particular task. This means that when running the system, if another task is run at a level of assurance that is higher (w.r.t. to its execution duration) than the criticality of the initial task, then this task will be suspended, since the conditions that guarantee its feasibility are no longer met.

A. This paper

In this paper we want to compare a set of criticality aware and unaware partitioned heuristics using experimental success ratio measurements based on task sets created through a new random system generator. As the large number of contesting heuristics introduces a considerable workload, special evaluation techniques based on a race approach are used to find the

best performing heuristic in the least amount of tests. These methods are also described in this paper through pseudocode.

B. Organisation

In Section II a brief review of other works in mixed-criticality uniprocessor and multiprocessor scheduling is given. In Section III the mixed-criticality task set model and notations are recalled. Section IV covers the description of the partitioning problem and a definition of the partitioning heuristics that are used in this paper and Section V covers our methods to evaluate the average success ratios of these heuristics. In Section VI our random task set generation algorithm is explained in detail as well as the results of our experiments.

II. Related Work

Mixed-Criticality scheduling is an emerging research domain which has gained a lot of interest in the past years. This approach was first introduced by Vestal [4]. In his work, he highlighted the difficulty in computing exact WCETs, and observed that in practice, the higher the degree of assurance required that a task will never exceed its WCET, the more conservative the approximation of the latter becomes. This degree of assurance is characterized by a level of criticality. He also suggested a fixed-task-priority strategy based on the Audsley priority assignment scheme [5]. Dorin et al. [6] proved that under the restricted case of independent task systems with constrained-deadlines, Vestal's modified Audsley's approach was optimal in the class of fixed-task priority algorithms. Nowadays, the Mixed-Criticality (MC)-Schedulability problem is commonly known to arise in two different contexts. The first one is concerned with applications that are subject to multiple certification requirements. In this context, different Certification Authorities (CA) need to validate the application functionalities. Nevertheless, the more critical a functionality is, the more pessimistic the CA will be in the estimation of the WCET. Baruah et al. [7] studied mixed-criticality systems in this context, but restricted their work to a set of mixed-criticality jobs. In particular, Baruah [8] pointed out the intractability of the MC-Schedulability problem, and quantified the fundamental limitations of MC-Scheduling for certification considerations. To tackle the intractability of MC-Scheduling, they suggest two sufficient schedulability conditions, referred to as the WCR-schedulability and OCBP-schedulability conditions. Later, Baruah and Li [9] extended their previous work and suggested a fixed-job-priority scheduling strategy based on their OCBP-schedulability condition. Baruah et al. also adapted the Earliest Deadline First algorithm to mixed-criticality systems, by modifying the deadlines of tasks. This approach is known as EDF-VD and is the one under study in this paper. More recently, Guan et al. [10] presented a new approach for scheduling mixed-criticality systems, which relies on an offline fixed-job-priority ordering computation, which is then used on-line by the scheduler. At the same time, Baruah et al. [11] formalized the response time analysis for mixed-criticality tasks. In [12] an overview of mixed-criticality scheduling on multiprocessors is proposed.

III. Model and notations

This paper is set in the context of constrained deadline periodic synchronous dual-criticality real-time task systems on a discrete timeline model. Each system is represented by a task set $\tau = \{\tau_1, \tau_2, \ \dots \ \tau_n\}$ where each task τ_i is a 4-uple $(C_i^{LO}, C_i^{HI}, D_i, T_i)$ where C_i^{LO} is the worst case execution time (WCET) in LO mode, C_i^{HI} is the WCET in HI mode, D_i is the relative deadline (the maximum allowed amount of time between an arrival and the corresponding end of this task) and T_i is the inter-arrival time (the period). All tasks satisfy the conditions $0 < C_i^{LO} \leqslant D_i \leqslant T_i$. If $C_i^{HI} > 0$, τ_i is said to be a HI criticality task and additionally $C_i^{LO} < C_i^{HI} \leqslant D_i$. A few additional notations are defined :

- The set of HI tasks : $\tau_{HI} = \{\tau_i \in \tau \mid C_i^{HI} > 0\}$
- The set of LO tasks : $\tau_{LO} = \{\tau_i \in \tau \mid C_i^{HI} = 0\}$
- Task τ_i utilization in LO and HI modes : $U_{LO}(\tau_i) = \frac{C_i^{LO}}{T_i}$ and $U_{HI}(\tau_i) = \frac{C_i^{HI}}{T_i}$
- Task τ_i density in LO and HI modes : $\lambda_{LO}(\tau_i) = \frac{C_i^{LO}}{\min(D_i, T_i)}$ and $\lambda_{HI}(\tau_i) = \frac{C_i^{HI}}{\min(D_i, T_i)}$
- LO mode system utilization : $U_{LO}(\tau) = \sum_{\tau_i \in \tau} U_{LO}(\tau_i)$
- HI mode system utilization : $U_{HI}(\tau) = \sum_{\tau_i \in \tau} U_{HI}(\tau_i)$
- n_{LO} and n_{HI} are (respectively) the size of τ_{LO} (respectively) τ_{HI}
- $n = n_{HI} + n_{LO}$

IV. Partitioned EDF-VD

A. EDF-VD Feasibility Condition

EDF-VD is a mixed-criticality uniprocessor scheduling algorithm [13]. It has later been extended to multiprocessor platforms [14] by using the same concept as fpEDF [15], a multiprocessor single-criticality scheduling algorithm. EDF-VD performs by applying EDF on a set of tasks where the HI criticality tasks have smaller relative deadlines when the system is in LO mode. In [13] all such virtual deadlines (VD) are the product of a unique factor (the value is the same for all tasks in the system) with each of their original deadlines. A improvement over EDF-VD is made in [16] by defining an heuristic algorithm (tuneSystem) that reduces virtual deadlines on a per-task basis. The schedulability test corresponding to tuneSystem is also defined in [16]. This finer grained uniprocessor test is the one used by the partitioning heuristics in this paper, as it displays among the best average success ratio in the current uniprocessor mixed-criticality scheduling state of the art.

Ekberg and Yi [16] extend the common definition of the demand bound function (DBF) to mixed-criticality systems. For a dual-criticality task set τ, two demand bound functions are defined : $dbf_{LO}(t)$ and $dbf_{HI}(t)$. These two functions have the properties of a single-criticality DBF. This means that we have the EDF feasiblity condition :

$$\forall t > 0 : dbf_{LO}(t) \leqslant t \text{ and } dbf_{HI}(t) \leqslant t$$
$$\iff \text{the system is schedulable}$$

To verify schedulability using these conditions, the tuneSystem algorithm (see Algorithm 1) is run [16]. tuneSystem like EDF-VD will shift the load of the HI mode DBF towards

the LO mode DBF by reducing the virtual LO mode deadlines of HI tasks. However, tuneSystem makes accurate modifications to single tasks instead of multiplying all of them by a factor, which makes it more complex but also more powerful than EDF-VD [16]. Each time some t that does not satisfy the condition is found, a deadline is changed to fix the problem if possible. When the virtual deadlines of multiple tasks have been reduced, the algorithm can also backtrack some of these changes in order to maintain the condition on the LO mode DBF. This continues until either the condition is satisfied or no deadline changes can be made anymore, in which case the system is not schedulable using tuneSystem. This algorithm achieves much higher success ratio than EDF-VD for utilization above 80% as EDF-VD starts to fail the scheduling of some systems at around 70% utilization [16]. Before running the algorithm, a bound ($tBound$) on the latest time instant for which the dbf condition must be verified is computed using the technique in [17]. Doing this in HI and LO mode results in two different bounds, out of which the highest must be chosen.

Algorithm 1: tuneSystem

Require: $tBound$ as defined in text
$candidates \leftarrow \tau_{HI}$
$changed \leftarrow$ **false**
$modTask \leftarrow \epsilon$
while $changed$ **do**
  $changed \leftarrow$ **false**
  for $t = 0$ **to** $tBound$ **do**
    if $dbf_{LO}(t) > t$ **then**
      if $modTask = \epsilon$ **then**
        return false
      end if
      increment the virtual deadline of $modTask$
      if $modTask \in candidates$ **then**
        $candidates \leftarrow candidates \setminus modTask$
      end if
      $modTask \leftarrow \epsilon$
      $changed \leftarrow$ **true**
      break
    end if
    if $dbf_{HI}(t) > t$ **then**
      if $candidates = \phi$ **then**
        return false
      end if
      $modTask \leftarrow$ task in $candidates$ which HI dbf increases the most between t and $t-1$
      decrement the virtual deadline of $modTask$
      if the WCET in LO mode of $modTask$ is equal to its virtual deadline **then**
        $candidates \leftarrow candidates \setminus modTask$
      end if
      $changed \leftarrow$ **true**
      break
    end if
  end for
end while

B. Partitioning heuristics

The problem of finding an assignment of tasks to processors that fits a given platform is similar to the Bin-Packing problem (BPP) which is known to be NP-hard. Variants of BPP frequently occur in computer science problems and this resulted in various heuristics being developped in the literature [18]. In this paper the focus will be put on heuristics following a strict framework (which can be thought as a metaheuristic) : items (tasks) are sorted given a specific *criteria* then they are assigned to bins in that order. The choice of which bin a given item will be assigned to is specified by an *assignment rule*. Together, the sorting criteria and assignment rule unequivocally describe the heuristic [19]. A total of four possible task ordering criteria are considered : utilization, period, deadline and density (U, P, L and D for short) and two possible orders for each of them (increasing or decreasing, respectively I and D). Additionally, the most frequent assignement rules are First Fit, Next Fit, Best Fit and Worst Fit (F, N, B and W). In the single-criticality partitioning literature, Best Fit and First Fit with Decreasing Utilization or Decreasing Density are found to display the best success ratio [19].

All variants considered, there is a total of 32 possible heuristics following this "criticality unaware" framework. In this paper a new type of heuristic ("criticality aware") is defined specifically for mixed-criticality systems. In this new kind of heuristic, the task set τ is split into HI and LO task sets, τ_{HI} and τ_{LO}. First, tasks in τ_{HI} are partitioned on all processors following one of the heuristics described above. Then, tasks in τ_{LO} are partitioned on the remaining space on the platform following another (possibly the same) heuristic. There are 1024 different heuristics following this new structure. In criticality aware heuristics, Worst Fit and Best Fit behave a little differently, as it makes more sense to use the utilization in HI mode during the assignment of tasks in τ_{HI} and the utilization in LO mode during the assignment of tasks in τ_{LO}. Following the same principle, in all heuristics LO criticality tasks of are ordered using their C_i^{LO} value as WCET (which is needed to calculate utilization and density) and HI criticality tasks with their C_i^{HI} value as WCET. Heuristics are noted using their assignement rule followed by the two letters code of their ordering criteria. For example, Best Fit Increasing Period will be noted B_{IP}. Criticality aware heuristics are noted by giving the heuristic for the LO tasks followed by a slash then the heuristic for the HI tasks, such as B_{IP}/N_{IL}.

V. Partitioning heuristics selection

One of the goals of this paper is to assess which heuristic has the best success ratio by experimentation or to what extent the kind of tested system can influence which heuristic is best. The results of such experiments heavily rely on which system generation algorithm was used (described in Section VI-A) but also on its parameters. Both heuristic evaluation methods use the same system generation parameters, which can be described as follows :

- System tailored for 4 processors
- 20 tasks, out which 8 are HI criticality tasks
- Both $U_{LO}(\tau)$ and $U_{HI}(\tau)$ randomly (and independently) distributed between 0 and the number of processors
- The minimum period $tMin = 5$ and the maximum period $tMax = 50$

A. Racing between heuristics

The large number of possible heuristics and long computation time required to test systems on many heuristics leads to a need for efficiency. In Algorithm 2, the search of the best heuristic is concentrated towards those that showed a good success ratio in past experiments. Such selection algorithms are usually called racing algorithms and are used in machine learning for model selection and parameter tuning [20]. In our context this approach enables us to quickly eliminate non significant heuristics. The search algorithm itself uses three parameters :

- $nRounds$ is the number of times the outer loop (called round) of the algorithm is executed, corresponding to the number of times the working set of heuristics is shrunk.
- $nTests$ is the number of systems generated and tested during the first round.
- $exploration$ is the factor (between 0 and 1) of reduction of the size of the working set of heuristics. Additionally the number of tested systems is multiplied by $\frac{1}{exploration}$ each round.

Algorithm 2: Racing for heuristics

Require: $nRounds$, $nTests$ and $exploration$ as defined previously.
$heuristics \leftarrow$ all possible heuristics
$curTests \leftarrow nTests$
for $r = 1$ **to** $nRounds$ **do**
  for $t = 1$ **to** $curTests$ **do**
    $system \leftarrow$ randomly generated system
    for all $h \in curHeuristics$ **do**
      partition $system$ with h
      update success ratio of h with the result
    end for
  end for
  $curHeuristics \leftarrow$ the $exploration^r$ portion of $heuristics$ with highest success ratio
  $curTests \leftarrow curTests/exploration$
end for
return $heuristics$ sorted by success ratio

The time complexity of Algorithm 2 in terms of number of partitionings is in $O(nRounds \cdot nTests \cdot |heuristics|)$. Note that the number of partitionings per round does not change. In later rounds, fewer heuristics are tested on a larger number of systems.

B. Direct elimination

Direct elimination is a slightly different form of racing. In Algorithm 3 heuristics are selected for further testing based on their ability to dominate other heuristics rather than simply success ratio. Each run begins with all heuristics being tested on one system. If at least one heuristic could schedule the system, all the heuristics that were unable to schedule it are discarded. This is repeated until only one heuristic remains in the set of heuristics (or if we have reasonable evidence that stability has been reached, see $stability$), then the whole set of heuristics is reset and the operation is repeated.

- $nRuns$ determines the number of times the complete operation (starting with all heuristics and reducing until stability) is done.
- $stability$ is the maximum amount of systems that will be tested without eliminating heuristics.

Algorithm 3: Direct heuristic elimination

Require: $nRuns$ and $stability$ as defined previously.
for $r = 1$ **to** $nRuns$ **do**
  $heuristics \leftarrow$ all possible heuristics
  while $t < stability$ and $|heuristics| > 1$ **do**
    $system \leftarrow$ randomly generated system
    $schedules \leftarrow$ mapping of all heuristics to **false**
    for all $h \in heuristics$ **do**
      partition $system$ with h
      if h can partition $system$ **then**
        $schedules[h] \leftarrow$ **true**
      end if
      update success ratio of h with the result
    end for
    if $\exists i \mid schedules[i]$ and $\exists j \mid \neg schedules[j]$ **then**
      $t = 0$
      for all $h \in heuristics$ **do**
        if $\neg schedules[h]$ **then**
          remove h from $heuristics$
        end if
      end for
    else
      $t = t + 1$
    end if
  end while
end for
return $heuristics$ sorted by success ratio

The worst case number of tests of Algorithm 3 is in $O(nRuns \cdot stability \cdot |heuristics|)$ as it will take a maximum of $nTest - 1$ tests to eliminate each heuristic individually. In practice Algorithm 3 is faster than Algorithm 2 because the actual number of tests depends on the generated systems for Algorithm 3 but not for Algorithm 2. One system is very often enough to eliminate a large portion of the set of heuristics, which means $stability \cdot |heuristics|$ is a very pessimistic estimate of the time required for one round.

VI. Experiments

A. Task set generation

Widely used standard random task set generation algorithms exist for uniprocessor single-criticality scheduling, such as UUniFast [21] for generating uniform task utilizations. But this standard way of generating task sets does not extend to more specific systems, where an ecosystem of techniques still exists. UUniFast has been extended to multiprocessor systems in [22], which is the basis of the multiprocessor mixed-criticality task set generation algorithm found in this paper. Other works in the literature have proposed techniques for generating mixed-criticality task sets, such as [14], [23] using an explicit scaling factor between the utilization in HI and LO mode.

In this paper task utilizations in LO and HI mode are generated taking the constraints specific to mixed-criticality

systems into account and in a way that aims to keep the uniformity of the generated systems intact. The generator receives guidelines on $U_{HI}(\tau)$, $U_{LO}(\tau)$, the ratio of HI tasks ($ratioHI = \frac{n}{n_{HI}}$) and the maximum and minimum period ($tMax$ and $tMin$). The generator will try to create a system meeting those guidelines as precisely as possible, although making no guarantees.

1) LO and HI utilizations: The algorithm generates n utilization values from $U_{LO}(\tau)$ for LO mode and n_{HI} utilization values from $U_{HI}(\tau)$ for HI mode using a variant of the multiprocessor UUniFast algorithm [22]. Those utilization values are then associated to one another in HI-LO couples making sure that for each HI task the HI utilization is higher than the LO utilization. If this cannot be achieved, HI utilizations are re-generated for one less HI task until one such association can be found. When a valid association is found, it is randomly shuffled with the limitation that is has to stay correct.

2) Periods, WCETs and deadlines: Periods are randomly chosen between $tMin$ and $tMax$ with a log uniform random variable biased towards lower values, as done in [22]. WCETs are then directly calculated from those periods and the utilizations. HI WCETs are checked to be at least one plus the corresponding LO WCET. Periods are then adjusted to make sure no task has utilization above or equal to one then ensuring that the total system utilization is below the given guidelines for LO and HI utilization. This is done by repeatedly choosing a random task in τ and incrementing its period until both conditions become satisfied. Finally, deadlines for each task are randomly chosen between the highest WCET (C_i^{LO} for LO tasks, C_i^{HI} for HI tasks) and the period with a logarithmic uniform random variable biased towards higher values.

B. Results

The methods explained in Section V-A and Section V-B agree on the general domination of the single-criticality heuristics F_{DU} and F_{DD} both in criticality unaware (confirming previous results in single-criticality systems [19]) and in criticality aware mixed-criticality partitioning heuristics. However the conducted experiments aggregated success ratios based on systems with utilizations in HI and LO mode ranging from 1 to the number of processors (4), giving the same weight to each system. In a realistic environment, it is expected that systems with higher utilization will be more interesting as we want to use the platform to maximum capacity (or use lighter hardware to run the same set of tasks). Three of the best heuristics have been run on new systems generated with a range of fixed HI and LO utilizations (all other parameters remaining the same) to show how their success ratio evolves with HI and LO utilization. The results for F_{DD}, F_{DD}/F_{DD} and F_{DD}/W_{DD} are respectively shown in Figure 3, 1 and 2.

The intuitive expectation is that using W_{DD} as a HI mode heuristic will give better results when $U_{LO}(\tau)$ is high and $U_{HI}(\tau)$ is low. This is motivated by the reasoning that if HI utilization is as balanced between processors as possible, it is more likely that more LO tasks will be schedulable over the whole system. However if we forget about LO tasks, W_{DD} performs worse than F_{DD} to find a good partitioning of HI tasks (the same way it performs worse in single-criticality scheduling), which means F_{DD}/F_{DD} has an advantage over F_{DD}/W_{DD} when partitioning task sets with high $U_{HI}(\tau)$ and low $U_{LO}(\tau)$. This is verified in our experiments as when

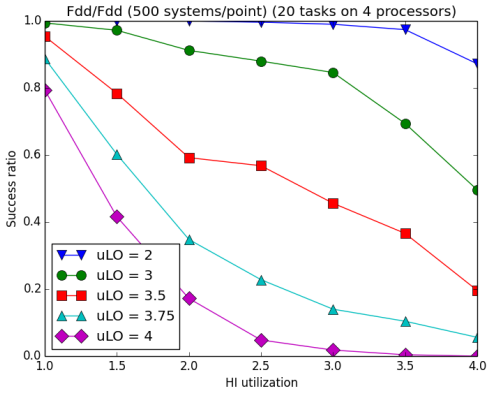

Fig. 1. Success ratio of F_{DD}/F_{DD} for various HI and LO utilizations

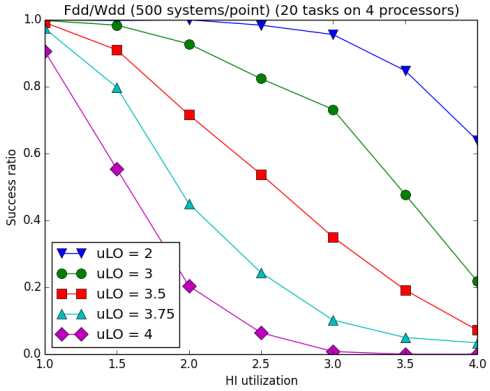

Fig. 2. Success ratio of F_{DD}/W_{DD} for various HI and LO utilizations

compared to F_{DD}/F_{DD}, F_{DD}/W_{DD} does globally better when HI utilization is lower than 2.5 if LO utilization is at least 3.5.

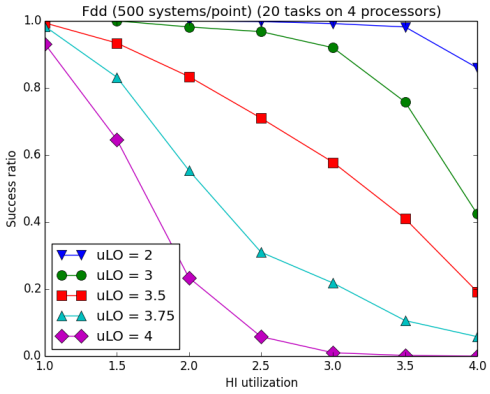

Fig. 3. Success ratio of F_{DD} for various HI and LO utilizations

F_{DD} obtains slightly better success ratio in most situations. However if we try to directly compare F_{DD} with F_{DD}/W_{DD}, we obtain surprising results.

In Figure 4 the amount of systems that are schedulable by F_{DD}/W_{DD} but not by F_{DD} is compared with the amount

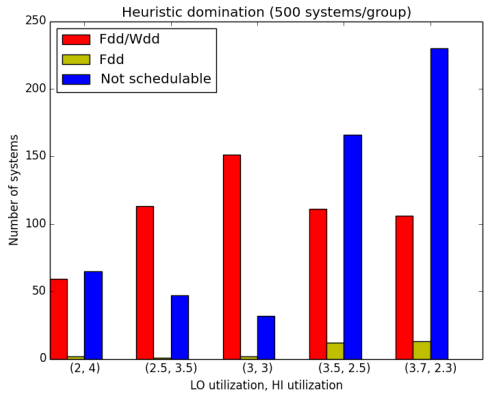

Fig. 4. Number of systems where F_{DD}/W_{DD} dominates F_{DD} and vice versa

of systems schedulable by F_{DD} but not by F_{DD}/W_{DD} for various values of $U_{LO}(\tau)$ and $U_{HI}(\tau)$. For all the chosen utilization combinations, there are very few systems that belong to the second category. This number increases as HI utilization increases and LO utilization decreases, which hints that not allowing LO tasks to be partitioned before any HI tasks (the definition of criticality aware heuristics) might not be a good choice in systems that have heavy LO tasks.

VII. Conclusion

In this paper, we have considered the problem of partitioned EDF-VD scheduling for mixed critical (HI and LO) periodic tasks on identical multiprocessors. In a mixed critical system, it is mandatory to grant HI critical tasks. We have studied different meta-heuristic that maximize the success ratio of LO critical tasks while granting HI critical tasks. We considered two partitioned scheduling approaches, one called criticality aware that first tries to assign HI tasks to processors and then LO tasks separately and one called "criticality unaware" that does not take into account the criticality of the tasks. We have adopted a race metaheuristic to select the best partitioning heuristics according to several placement and sorting criteria. We show that taking into account criticality levels by first assigning HI critical tasks leads to better success ratio for HI tasks. Furthermore, when Worst Fit with Decreasing Density succeeds to partition HI tasks, assigning LO tasks with Fist Fit Decreasing Density maximizes the success ratio of LO tasks.

References

[1] D. Johnson, "Fast algorithms for bin packing," *Journal of Computer and Systems Science*, vol. 8(3):272314, 1974.

[2] A. Burns and R. Davis, "Mixed criticality systems-a review," , Department of Computer Science, University of York, Tech. Rep., 2013.

[3] L. George, P. Courbin, and Y. Sorel, "Job vs. portioned partitioning for the earliest deadline first semi-partitioned scheduling," *Elsevier Journal of systems architecture, Special issue on multiprocessor real-time scheduling Ed. U.Devi and J.H. Anderson*, vol. 57, no. 5, pp. 518–535, 2011.

[4] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 239–243.

[5] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Tech. Rep. YCS-164, 1991.

[6] F. Dorin, P. Richard, M. Richard, and J. Goossens, "Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities," *Real-Time Syst.*, vol. 46, pp. 305–331, December 2010.

[7] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 13–22.

[8] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *Mathematical Foundations of Computer Science 2010*, pp. 90–101, 2010.

[9] H. Li and S. Baruah, "An algorithm for scheduling certifiable mixed-criticality sporadic task systems," in *31st IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 183–192.

[10] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," in *32nd IEEE Real-Time Systems Symposium*. IEEE Computer Society, 2011, pp. 13–23.

[11] S. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *32nd IEEE Real-Time Systems Symposium*. IEEE Computer Society, 2011, pp. 34–43.

[12] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-criticality scheduling on multiprocessors," *Real-Time Systems*, pp. 1–36, 2013.

[13] S. K. Baruah, V. Bonifaci, G. DAngelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *Algorithms–ESA 2011*. Springer, 2011, pp. 555–566.

[14] H. Li and S. Baruah, "Global mixed-criticality scheduling on multiprocessors," in *24th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 2012, pp. 166–175.

[15] S. K. Baruah, "Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 781–784, 2004.

[16] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Systems*, pp. 1–39, 2013.

[17] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 1990, pp. 182–190.

[18] R. Yesodha and T. Amudha, "A comparative study on heuristic procedures to solve bin packing problems," *International Journal*, 2012.

[19] I. Lupu, P. Courbin, L. George, and J. Goossens, "Multi-criteria evaluation of partitioning schemes for real-time systems," in *2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2010, pp. 1–8.

[20] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, "A racing algorithm for configuring metaheuristics." in *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 2. GECCO, 2002, pp. 11–18.

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

[22] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2010, pp. 6–11.

[23] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Improving the scheduling of certifiable mixed-criticality sporadic task systems," Tech Rep 2013-008, Department of Information Technology, Uppsala University, Tech. Rep., 2013.