SMILEY: A Mixed-Criticality Real-Time Task Scheduler for Multicore Systems

Alen Sabu, Biju Raveendran, Rituparna Ghosh
BITS Pilani K. K. Birla Goa Campus, India
{h2014023, biju, f2012109}@goa.bits-pilani.ac.in

***Abstract*—With the massive advancement of fabrication technology in recent past, the attempt to execute all kinds of tasks on a shared, multicore platform has grown into a wide research area. The growth of safety-critical systems has resulted in a major design issue of maintaining tight timing constraints for safety-critical and mission-critical workloads. A mix of tasks with different criticalities are deployed in a system, called mixed-criticality system, and there exist various scheduling algorithms to meet the real-time constraints of such systems. This paper discusses a novel algorithm for scheduling mixed-criticality sporadic real-time jobs in a hard affinity multicore environment. The dynamic slack generated at run-time while executing jobs is used to schedule low criticality tasks without missing high criticality task deadlines. This work attempts to reduce the unproductive time and to increase the number of completed low criticality jobs without compromising any high criticality job execution. Experimental evaluation with synthetic benchmark suites shows 73.9% and 85.2% improvement in time utilization when compared with EDF-VD and CBEDF respectively.**

***Index Terms*—Mixed-criticality systems; multicore scheduling algorithm; sporadic task systems.**

I. Introduction

A mixed-criticality task differs from a traditional real-time task because of the additional criticality constraint. There is a defined maximum criticality level for each task, and a job of that task can take criticality value less than or equal to the maximum criticality level of the task. At a higher criticality level, the applications need to be certified by Certification Authorities (CAs). Hence the worst-case execution time (WCET) used by CA for these tasks will be larger, as the verification time would be more with conservativeness. CAs are usually pessimistic with their WCET analysis on high criticality jobs while the system designer is more concerned about schedulability of all the jobs. The increasing demand for high processing power, and the size, weight and power (SWaP) constraints urge to deploy mixed-criticality tasks on multicore platforms, which makes mixed-criticality scheduling theory [1] more complex.

The task model used here is the same as that of a standard mixed-criticality task model which is used in most of the relevant literature [2]. There will be 'χ' execution times for a χ-level task, where $\chi \in \mathbb{N}$. So, the execution time (C) in a mixed-criticality system (MCS) [3] is a vector with L coordinates, where $L \in \mathbb{N}$ is the maximum criticality level of the system.

Now, the execution time vector of a task τ_i, $\overrightarrow{C_i} \in \mathbb{Q}_+^L$, can be represented as:

$$\overrightarrow{C_i} = [C_i(1), C_i(2), \ldots, C_i(\chi_i), \ldots, C_i(L)]$$

where χ_i is the criticality of the task τ_i. $\overrightarrow{C_i}$ always has L coordinates even though the criticality of the task τ_i is identified only till $\chi_i < L$. The remaining levels will have the execution times same as the execution time at level χ_i. That is,

$$C_i(\chi_i) = C_i(\chi_i + 1) = \ldots = C_i(L)$$

The execution time identified at each level is the WCET at that level.

Hence a typical task in MCS can be defined by a 4-tuple of parameters $\tau_i = (\chi_i, P_i, \overrightarrow{C_i}, D_i)$. Given below are the descriptions of the parameters.

- $\chi_i \in \mathbb{N}$ is the criticality of the task τ_i.
- $P_i \in \mathbb{Q}_+$ is the minimum time between the releases of two consecutive jobs of the same task τ_i, usually called as period of the task.
- $\overrightarrow{C_i} \in \mathbb{Q}_+^L$ is the vector representing the WCET estimates of the task at levels ranging from 1 to L.
- $D_i \in \mathbb{Q}_+$ is the maximum time after the release of the job upto which the job of a task τ_i is allowed in the ready queue, usually called as the deadline of the task.

The criticality of the jobs of task τ_i can be any natural number $\leq \chi_i$ depending on the time upto which the job has finished its execution. The system also has a criticality which starts in the lowest criticality mode. The criticality of the system changes when the behavior of the jobs changes, eventually reaching the highest criticality mode. This work assumes dual criticality levels with Level-1 as LO (no certification needed) and Level-2 as HI (certification needed). A task is said to be in LO criticality at any moment when its job has a criticality less than the system criticality at that moment.

The processor clock frequency has ceased to grow in the recent past, which resulted in the growth of multicore processors. The use of multicore processors in designing embedded systems is a growing trend. There would be an increase in utilization without much increase in energy consumption for multicore systems when compared with multiprocessor systems. High amount of parallelism can be achieved through multicore processors as the cores are tightly coupled on the same die and load-sharing is not complex. Also, multiprocessors

978-1-5386-5048-6/18/$31.00

have higher communication delay and operating frequency. Various power management techniques like Dynamic Voltage Frequency Scaling (DVFS) and dynamic procrastination can be more effective in multicore systems [4], [5].

When most of the recent works in real-time systems are aligned with multicore processors, it would be equitable that mixed-criticality systems as well advance in that direction. Multicore and manycore systems are the best solutions for a system which inhibits different kinds of complex, independent and less-predictable workloads.

Most of the algorithms in-place for mixed-criticality scheduling do not consider the execution of LO criticality jobs. The ratio of number of failed LO criticality jobs over total number of LO criticality jobs is pretty high. Whenever there is space for LO criticality job(s), it needs to get executed even if the system is in HI criticality mode without compromising the deadline of any HI criticality job. A job is said have failed when it could not finish execution within its deadline. This work addresses increasing the performance of LO criticality jobs thereby reducing cascaded failures and unproductive time. Cascaded failures occur when a currently executing job triggers the failure of the jobs following it.

The scheduling algorithm which we present here is divided into three parts. The first part is Pre-scheduler, in which the HI criticality tasks are assigned to different cores based on first-fit decreasing bin-packing algorithm. The second part is SlackFinder which finds the available slack for a LO criticality job in a core at a particular time. The third part is Run-time scheduler which schedules HI criticality jobs in each core, executes SlackFinder, and tries to best-fit LO criticality jobs in the schedule of a core.

The remaining part of the paper is organized as follows. Section II describes the state of the art literature survey in multicore mixed-criticality scheduling. In Section III, we describe the work which is proposed with an example. Section IV describes the theoretical evaluation. The next section explains the experimental setup used in the evaluation of the results. The experimental results are shown in Section VI and we conclude the paper in Section VII.

II. Related Work

MCS has proven to be a dynamic and promising area of research and a significant amount of work has started since the last decade, and has increased manifolds ever since. Vestal's [6] work on Fixed Priority (FP) scheduling provided some important insights into the nature of mixed-criticality scheduling and presented a modified Preemptive Fixed Priority (PFP) schedulability analysis. He modelled task parameters such as WCET based on the criticality levels. The same code would thus have higher WCET if it is defined to be safety critical than it would otherwise. The first part of [6] presented a period transformation technique which modifies the workload in a way that HI criticality tasks get higher priorities and this approach augmented schedulability of multi-criticality systems if context-switching and code-slicing overheads are ignored. The underlying principle is that if a HI criticality task has a larger period than that of a LO criticality task then run-time slicing is used to schedule the HI criticality task such that it has a smaller period and execution time. The second part of [6] provided a scheduling technique using Audsley's priority assignment algorithm which can serve either as an alternative or used in combination with period transformation. The performance metric used to evaluate these techniques was the critical scaling factor. Period transformation combined with the modified analysis provided better results than Deadline Monotonic scheduling regardless of whether priorities were assigned using Deadline Monotonic ordering or Audsley's algorithm.

This was followed by the work of Baruah and Vestal [2] in 2008 which used a mixed-criticality sporadic task model, and conducted an in-depth study of schedulability and feasibility issues for mixed-criticality systems on uniprocessor platforms. They determined the relative abilities of various kinds of scheduling algorithms such as Earliest Deadline First (EDF), FP algorithms *etc*. There are multi-criticality sporadic task systems which are not schedulable by Fixed Task Priority (FTP) or Fixed Job Priority (FJP) algorithms but can be scheduled by Dynamic Priority (DP) algorithms. Furthermore, it gave an important result that EDF is no longer an optimal algorithm when multiple criticality levels were introduced. In fact, scheduling strategies of FTP and EDF cannot be compared with each other when it comes to scheduling of multi-criticality sporadic task systems as there are mixed-criticality sporadic task systems which are schedulable by one algorithm but not by the other. They proposed a hybrid-scheduling algorithm for scheduling multi-criticality sporadic task systems which dominated EDF and also Vestal's previously proposed algorithm.

Baruah *et al.* [7], [8] proposed EDF with Virtual Deadlines (EDF-VD) which could schedule mixed-criticality task systems for any number of criticality levels. EDF-VD is optimal with respect to metrics such as processor speedup factor and utilisation bounds. Furthermore, two extensions of EDF-VD were proposed in [8] to enhance its performance. They used a mixed-criticality sporadic task model for the analysis of implicit-deadline systems and arbitrary-deadline systems. The EDF-VD scheduler reduces deadline of HI criticality tasks when the system is in a LO criticality mode to ensure schedulability across a criticality change. EDF-VD may sometimes turn out to be excessively pessimistic and end up discarding LO criticality jobs altogether while executing in the HI criticality mode. Also once the system switches to HI criticality mode it never returns to the lower criticalities.

Park and Kim [9] proposed a dynamic slack based algorithm called Criticality Based Earliest Deadline First (CBEDF) for scheduling dual-criticality task systems on uniprocessor platforms. In CBEDF, the LO criticality tasks are executed in the slack generated by HI criticality tasks. It provided the schedulability test for the algorithm and showed that it outperforms those of previously proposed algorithms such as OCBP [10] and EDF. CBEDF suffers from the problem of a cascading effect of multiple LO criticality jobs failing to finish

before their deadlines and hogging the processor unnecessarily, while it could have been used more productively. Cascading failures of LO criticality jobs reduce the productivity of the processor as the time used to partially execute such jobs could have been otherwise used to do productive work.

The idea to use multiple computing modules in MCS is not new. Researchers tend to use multicore processors over multiprocessors because of their higher productivity with reduced energy consumption. Various multicore and multiprocessor algorithms were developed and proved to be optimal for MC tasks.

Mollison *et al.* [11] suggested an architecture for scheduling MC tasks on multiprocessor platforms which assumed five levels of criticality. They analyzed the schedulability of such a system to prove it to be optimal. Li *et al.* [12] modified EDF-VD [7] and combined it with fpEDF [13] to design a global scheduling algorithm for mixed-criticality task systems on multiprocessors. A job which gets preempted from one processor may resume its execution in another processor. This algorithm does not consider the processor migration time of a task which is not negligible. The performance of the algorithm was similar to that of EDF-VD. Once the system criticality reaches HI, the LO criticality jobs won't receive any further execution time.

In [14], Lee *et al.* extended the fluid scheduling model [15] to fit for mixed-criticality multiprocessor systems which they call MC-Fluid. The rate of execution of a task in fluid model is in proportion to its utilization. In MC-Fluid, the criticality of the task was also considered to determine the rate of execution. Thus a task in MC-Fluid has two execution rates - one when the system is in LO-mode and one when the system is in HI-mode. All LO criticality jobs that arrive after the mode switch are discarded.

Ren *et al.* [16] proposed an algorithm (TG-PEDF) for scheduling mixed-criticality tasks on multiprocessor systems. In TG-PEDF, all tasks had been grouped in such a way that no two HI criticality tasks were in the same group, *i.e.,* every HI criticality task was grouped with a subset of LO criticality tasks. The groups were scheduled under EDF scheme. Here, HI criticality tasks were isolated among each other and all LO criticality tasks were scheduled as best-effort. The results proved to outperform some for the existing multiprocessor MC scheduling algorithms for a better performance of LO criticality tasks.

In [17], Giannopoulou *et al.* considered the effects of resource sharing in multicore MCS. Jobs may stall waiting for a resource thereby increasing their execution times. They suggested a model for timing isolation on core level and global level, which allows only same criticality tasks to be executed simultaneously.

This work addresses the problem of unproductive processor utilization because of (i) discarding the LO criticality jobs in HI criticality mode, and (ii) cascaded failures.

III. Proposed Work

This work proposes SMILEY: Scheduler for Mixed-crIticality multicorE sYstems - a novel scheduling algorithm for mixed-criticality real-time tasks in multicore processors. Existing schedulers like EDF-VD [8] and CBEDF [9] concentrates on satisfying the schedulability of HI criticality jobs which results in LO criticality jobs missing their deadlines, even when there exists a feasible schedule with LO criticality jobs. This work focuses on designing a scheduler which maximizes the multicore processor utilization by executing maximum number of LO criticality jobs to completion without missing any HI criticality job's deadline.

The performance of a system is directly proportional to the productive processor utilization when the utilization is beyond 100%. The productive processor utilization is calculated as the total time used for executing completed jobs over total available execution time. Cascaded failures of LO criticality jobs result in low productive time. SMILEY devices a mechanism to check whether it is feasible to schedule a LO criticality job without missing deadlines of any HI criticality jobs, when the system is in HI criticality mode. Only the jobs which can finish their execution without deadline miss are allowed to schedule, which overcomes cascaded failures and guarantees better quality of service. This is achieved by finding maximum available slack between the current time and the deadline of the LO criticality job. The LO criticality job is admitted for execution only if the slack is larger than the WCET of that LO criticality job. All LO criticality jobs join global queue and are allocated to the same core where the previous instant of the same task executed, whenever possible. SMILEY reduces the number of decision points and the complexity of each decision.

A. The Algorithm

SMILEY is divided into three parts. The first part is a Pre-scheduler which statically allocates all HI criticality tasks to cores. SlackFinder is the next part which finds (at run-time) the slack available in a core for the execution of LO criticality jobs. The last part is the Run-time scheduler which schedules all the HI criticality jobs and the admitted LO criticality jobs in the system.

1) Pre-scheduler: This part is done offline and the results are used only when the system shifts to HI criticality mode. Pre-scheduler follows first-fit decreasing (FFD) [18] bin packing with sorting order as period, and then as HI criticality utilization to assign HI criticality tasks to cores statically. HI criticality utilization of a task is the utilization while considering the task's HI criticality execution time.

Pre-scheduler outputs the minimum number of cores (N_π) required for executing all the HI criticality jobs successfully when the system is in HI criticality mode, and the task set for each core. It returns error if number of cores in need for HI criticality tasks are more than the total number of cores available (Π_{max}) and we assume that Π_{max} cores can successfully schedule all the tasks in LO criticality mode with EDF scheduling algorithm. The remaining cores $(\Pi_{max} - N_\pi)$

can either be shutdown or be used to execute the rejected LO criticality jobs (in Run-time Scheduler).

The other variables used in the algorithm are defined below.

- $TasksHI$: Set of HI criticality tasks sorted by non-increasing order of their period and then by HI criticality utilization
  (a) $TasksHI_i.\mu$: HI criticality utilization of i^{th} task
  (b) $TasksHI_i.\pi$: Core allocated for i^{th} task
- N_{HI} : Total number of HI criticality tasks
- π : Set of all available cores in the system
  (a) $\pi_j.\mu$: Utilization of j^{th} core
  (b) $\pi_j.k$: Number of tasks allocated to j^{th} core
- U_{max} : Maximum schedulable utilization of a core

Algorithm: Pre-scheduler

Input: $TasksHI$, N_{HI}, U_{max}, Π_{max}
Output: N_π, $TasksHI$

for all Cores π_j $(0 \leq j < \Pi_{max})$ **do**
  $\pi_j.\mu \leftarrow 0$
  $\pi_j.k \leftarrow 0$
end for
$q \leftarrow 0$
for $i \leftarrow 0$ to $N_{HI} - 1$ **do**
  for $j \leftarrow 0$ to q **do**
    if $\pi_j.\mu + TasksHI_i.\mu \leq U_{max}$ **then**
      $\pi_j.\mu \leftarrow \pi_j.\mu + TasksHI_i.\mu$
      $\pi_j.k \leftarrow \pi_j.k + 1$
      $TasksHI_i.\pi \leftarrow j$
      break
    end if
  end for
  if $j > q$ **then**
    if $j = \Pi_{max}$ **then**
      return -1
    end if
    $q \leftarrow q + 1$
    $\pi_q.\mu \leftarrow TasksHI_i.\mu$
    $\pi_q.k \leftarrow 1$
    $TasksHI_i.\pi \leftarrow q$
  end if
end for
return $N_\pi \leftarrow q + 1$

2) SlackFinder: SlackFinder is invoked in Run-time Scheduler. The algorithm determines the slack available for a LO criticality job in a core at a particular time. For a LO criticality job J_{LO} with deadline D_{LO} at time $currTime$ $(\leq D_{LO})$, we need to consider the parameters as described below.

- S_1 : Set of jobs that are already present in local ready queue ($ReadyQ$)
- S_2 : Set of jobs that will arrive between $currTime$ and D_{LO}
- D_{max} : Maximum of the deadlines of all jobs present in S_1 and S_2, with hyperperiod as upper limit
- S_3 : Set of jobs that will arrive after D_{LO} with deadlines less than or equal to D_{max}
- S_4 : Set of jobs that will arrive after D_{LO} with deadlines greater than D_{max}

Since jobs in S_1, S_2 and S_3 have deadlines before D_{max}, these jobs have to execute completely by D_{max}. Only a portion of the jobs in S_4 need to be executed between D_{LO} and D_{max} which is calculated as

$$\frac{WCET(J_{S4}) \times (D_{max} - Arrival(J_{S4})) \times U_{HI}}{Deadline(J_{S4}) - Arrival(J_{S4})}$$

where job $J_{S4} \in S_4$ and U_{HI} $(= \pi_j.\mu)$ is the total utilization of HI criticality tasks allocated to that core (π_j).

Let $S = S_1 \cup S_2 \cup S_3 \cup S_4$. Hence, S is the set of jobs with complete or partial executions within the interval $currTime$ to D_{max}. SlackFinder sorts all the jobs in S in decreasing order of their deadlines. At each decision point, the jobs whose deadlines are greater than or equal to the present time (t) are considered. The idle times between $currTime$ and D_{LO} are considered as *slack* of J_{LO} in that core.

Algorithm: SlackFinder

Input: $ReadyQ$, $currTime$, D_{LO}, U_{HI}
Output: $Slack$

D_{max} = Maximum deadline of all jobs available and are arriving in $ReadyQ$ till D_{LO}
Let S be the set of jobs available between $currTime$ and D_{max}, sorted according to non-increasing order of their deadlines
$Slack \leftarrow 0$
$t \leftarrow D_{max}$
while $S \neq \emptyset$ **do**
  $J \leftarrow$ dequeue S
  if $J.deadline > D_{max}$ **then**
    $t \leftarrow t - \frac{J.wcet \times (D_{max} - J.arrival) \times U_{HI}}{J.deadline - J.arrival}$
  else if $J.deadline \geq t$ **then**
    $t \leftarrow t - J.wcet$
  else
    if $t < D_{LO}$ **then**
      $Slack \leftarrow Slack + t - J.deadline$
    end if
    $t \leftarrow J.deadline - J.wcet$
  end if
end while
if $t > D_{LO}$ **then**
  $t \leftarrow D_{LO}$
end if
$Slack \leftarrow Slack + t - currTime$
return $Slack$

3) Run-time scheduler: Run-time scheduler (Fig. 1) is responsible for scheduling HI criticality and LO criticality jobs admitted in the local ready queue ($ReadyQ_j$) of each core (π_j). All HI criticality jobs allocated to a specific core are added directly to its ready queue whereas all LO criticality jobs join the global queue, $JobQueue_{LO}$. The number of decision points in each core is the distinct arrival and completion times of the allocated HI criticality or LO criticality jobs. At each decision point of a core, an acceptance test is performed for all

jobs in the $JobQueue_{LO}$. It finds the available slack between current time and the deadline of the LO criticality job in each core. If the slack is sufficient enough to accommodate the job, it is transferred to that core's (Π_{LO}) local queue. If more than one core has sufficient slack, then the one with least slack ($Slack_{LO}$) greater than LO criticality job's WCET is chosen. All the jobs in local ready queue are then scheduled according to EDF scheduling algorithm.

For each core π_k ($0 \leq k < N_\pi$), arrival or completion of its allocated HI or LO criticality jobs to / from its local queue is a decision point ($nextDecisionPoint$). For each decision point of a core π_k, if $JobQueue_{LO}$ is not locked, then π_k acquires lock on $JobQueue_{LO}$ and checks each job in $JobQueue_{LO}$. It allocates the jobs to corresponding π_l ($0 \leq l < N_\pi$) or rejects according to the algorithm given in Fig. 1.

Example 1: Consider the following task set τ with parameters as mentioned in Table I.

TABLE I: Task set for SMILEY

$\tau_\mathbf{i}$	$\chi_\mathbf{i}$	$\mathbf{P_i}$	$\mathbf{c_i(1)}$	$\mathbf{c_i(2)}$	$\mathbf{D_i}$
τ_0	2	10	1	3	10
τ_1	2	10	2	5	10
τ_2	2	15	2	3	15
τ_3	2	15	4	6	15
τ_4	2	30	5	10	30
τ_5	1	10	3	3	10
τ_6	1	10	2	2	10
τ_7	1	15	4	4	15

We consider tasks with a maximum of two levels of criticality. Tasks τ_0, τ_1, τ_2, τ_3, and τ_4 are HI criticality tasks whereas τ_5, τ_6, and τ_7 are LO criticality tasks. The execution time for each task at LO criticality is given by $c_i(1)$ and that at HI criticality is given by $c_i(2)$. All the HI criticality tasks are initially sorted in non-increasing order of their periods. Hence, the HI criticality tasks τ_4 and τ_3 will join with core 0, and τ_2, τ_1 and τ_0 with core 1. All LO criticality jobs will join with a global queue and these jobs are checked for its acceptance at each decision point.

Fig. 2a and Fig. 2b shows the resultant schedule produced by SMILEY using 2 cores. The first subscript in the notation $J_{x,y}$ means the task index and the second one means the job instance of the task.

The accepted list of LO criticality jobs are $J_{5,0}, J_{6,0}$ and $J_{5,1}$. The list of rejected jobs are $J_{7,0}, J_{6,1}, J_{7,1}, J_{5,2}$ and $J_{6,2}$. At time 0, the $JobQueue_{LO}$ has $J_{5,0}, J_{6,0}$ and $J_{7,0}$ available ready for execution. The slack available in core 0 and core 1 at time 0 is 8 units and 0 units respectively. Hence, job $J_{5,0}$ with WCET of 3 units is accepted in the ready queue of core 0. For $J_{6,0}$ with WCET of 2 units, the slacks available are 5 units and 0 units respectively. $J_{6,0}$ is accepted in the ready queue of core 0. For $J_{7,0}$ with WCET of 4 units, the slack available are 3 units and 0 units respectively. Hence $J_{7,0}$ is rejected by both the cores.

In the above example, SMILEY offers a schedule with 100% productive time whereas CBEDF and EDF-VD can offer 90% and 93.4% productive time respectively.

IV. Theoretical Analysis

A. Schedulability and Correctness

This work assumes hard affinity tasks with no migration and all the basic scheduling requirements of a traditional MCS held valid. That is, in a dual criticality system, all jobs need to meet its deadline when the system is in LO criticality mode. We consider that the system follows EDF schedule when it is in LO criticality mode. When the system is in HI criticality mode, all the jobs fired by HI criticality tasks need to meet its deadlines and the status of LO criticality jobs are not taken into account.

All HI criticality jobs are allocated to its corresponding cores and they must meet their deadlines. Since allocation of HI criticality tasks to cores happens statically using FFD algorithm by considering the WCET (utilization) of the task, HI criticality jobs are always guaranteed to complete before their deadlines in an EDF schedule. The HI criticality utilization at HI-level,

$$U_{HI}^{HI} = \sum_{\text{i}|\chi_i = \text{HI}} \frac{C_i(HI)}{P_i} \leq N_\pi * U_{max}$$

where N_π is the minimum number of cores required and U_{max} is the maximum schedulable utilization (= 1 here). For each core π_j,

$$\sum_{\text{i }|\tau_i \text{ is alloted to } \pi_j} \frac{C_i(HI)}{P_i} \leq U_{max}$$

Each LO criticality job is assigned to a core only if there is enough slack to execute the job completely within the core. Any LO criticality job $J_{i,j}$ of τ_i which crosses $C_i(LO)$ is killed. If the number of available cores is less than the required number calculated by Pre-scheduler, then the task-set will be rejected.

B. Complexity

Pre-scheduler implements FFD bin-packing algorithm. It involves (i) sorting of HI criticality tasks, and (ii) allocating tasks to cores based on their utilization using First Fit Algorithm. The time complexity involved here is $\mathcal{O}(N_{HI}.\log N_{HI})$, when N_{HI} is the number of HI criticality tasks. Pre-scheduler is a one-time operation.

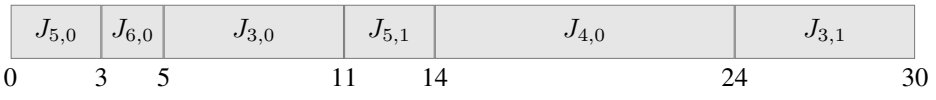
SlackFinder finds the slack available for a core at a particular time instant. Sorting is required to populate S. The time complexity of this algorithm is dependent on the number of jobs available between $currTime$ and D_{max}. Hence this phase is also polynomial time bounded with complexity $\mathcal{O}(\nu_s \log \nu_s)$, where ν_s is the number of jobs available in S between $currTime$ and D_{max}. SMILEY (Fig. 1) runs on all cores. The scheduler gets invoked when there is (i) an arrival of a HI criticality job, (ii) an arrival of a LO criticality job, (iii) a departure of a HI criticality job, and (iv) a departure of a LO criticality job.

Steps 2 to 4 has a complexity $\mathcal{O}(N_\pi)$. Another loop with loop count = number of decision-points begins at step 6 and ends at step 30. Steps 7 to 9 has a linear time complexity

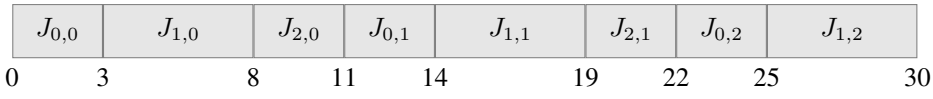
Input: $TasksHI$, $TasksLO$, N_π
Output: SMILEY schedule
Global data structures: $ReadyQ_0, ReadyQ_1, \ldots, ReadyQ_{N_\pi-1}$, $JobQueue_{LO}$, $Slack_{LO}$, Π_{LO}

1: $currTime \leftarrow 0$
2: **for all** Cores π_k $(0 \leq k < N_\pi)$ **do**
3: $\quad ReadyQ_k \leftarrow \emptyset$
4: **end for**
5: Do steps 6 to 30 for each core π_k $(0 \leq k < N_\pi)$
6: **while** $currTime < hyperperiod_{global}$ **do**
7: $\quad$ **for all** HI criticality jobs $J_{i,j}$ (where $TasksHI_i$ is allocated to π_k) arriving at $currTime$ **do**
8: $\quad\quad ReadyQ_k \leftarrow ReadyQ_k \cup J_{i,j}$
9: $\quad$ **end for**
10: $\quad$ **while** at $currTime$, $JobQueue_{LO}$ is not empty **do**
11: $\quad\quad$ **if** $JobQueue_{LO}$ is locked **then**
12: $\quad\quad\quad$ break
13: $\quad\quad$ **end if**
14: $\quad\quad$ Acquire lock on $JobQueue_{LO}$
15: $\quad\quad Slack_{LO} \leftarrow hyperperiod_{global}$
16: $\quad\quad \Pi_{LO} \leftarrow -1$
17: $\quad\quad$ **for all** Cores π_l $(0 \leq l < N_\pi)$ **do**
18: $\quad\quad\quad$ CheckAllocation($currTime$, l, $ReadyQ_l$, $\pi_l.\mu$)
19: $\quad\quad$ **end for**
20: $\quad\quad$ **if** $\Pi_{LO} \neq -1$ **then**
21: $\quad\quad\quad$ Acquire lock on $ReadyQ_{\Pi_{LO}}$
22: $\quad\quad\quad ReadyQ_{\Pi_{LO}} \leftarrow ReadyQ_{\Pi_{LO}} \cup JobQueue_{LO}.front()$
23: $\quad\quad\quad$ Release lock on $ReadyQ_{\Pi_{LO}}$
24: $\quad\quad$ **end if**
25: $\quad\quad$ dequeue $JobQueue_{LO}.front()$
26: $\quad\quad$ Release lock on $JobQueue_{LO}$
27: $\quad$ **end while**
28: $\quad$ Schedule job $ReadyQ_k.front()$ till $nextDecisionPoint$
29: $\quad currTime \leftarrow nextDecisionPoint$
30: **end while**
31: **function** CHECKALLOCATION($currTime$, i, $ReadyQ$, U_i)
32: $\quad Slack_i \leftarrow$SlackFinder ($ReadyQ$, $currTime$, $JobQueue_{LO}.front().deadline$, U_i)
33: $\quad$ **if** $Slack_i \geq JobQueue_{LO}.front().wcet$ and $Slack_i < Slack_{LO}$ **then**
34: $\quad\quad Slack_{LO} \leftarrow Slack_i$
35: $\quad\quad \Pi_{LO} \leftarrow i$
36: $\quad$ **end if**
37: **end function**

Fig. 1: SMILEY Run-time Scheduler



(a) Core 0



(b) Core 1

Fig. 2: SMILEY schedule

proportional to the number of jobs arriving at $currTime$ to the core. Steps 17 to 19 has $\mathcal{O}(N_\pi)$ calls to the function CheckAllocation, which has the same time complexity as that of SlackFinder. Hence the time complexity for this loop becomes $\mathcal{O}(N_\pi) * \mathcal{O}(\nu_s \log \nu_s)$. This shows that the run-time scheduling phase is polynomial time bounded which proves SMILEY is polynomial time bounded.

V. Experimental Setup

The experiments were conducted on SimMCSched, a multicore simulation environment created for this project using C++. SimMCSched takes task set corresponding to HI criticality tasks ($TasksHI$) and LO criticality tasks ($TasksLO$) as input. It also takes the maximum number of cores (Π_{max}) and allowed maximum utilization per core (U_{max}) as input. The maximum allowed utilization depends on the maximum number of HI criticality tasks per core and the scheduling algorithm in use. SimMCSched assumes all tasks are sporadic in nature with HI and LO criticality levels. The actual execution time of jobs is assumed to be same as WCET, and the scheduling and preemption overheads are negligibly small. The system is assumed to be in HI criticality mode.

The task sets are generated with random execution times and periods, with varying number of tasks and total utilizations. Total utilization of a task set consisting of N tasks is calculated as $\sum_{i=0}^{N-1} \frac{C_i(\chi_i)}{P_i}$, for a task τ_i with P_i as period, χ_i as its criticality level and $C_i(k)$ as the execution time at level k. For each HI criticality utilization, a set of 100 different task sets are generated and the average value of the parameter to be evaluated for all the 100 task sets is taken for analysis. Within 100 task sets, the hyperperiod varies from 900 units to 32000 units. LO criticality task sets are created for varying utilizations (40% to 65%), while the HI criticality utilization is set to 70%. The number of tasks in both HI and LO criticality task sets are also varied for evaluation.

The theoretical analysis of scheduling algorithm includes worst case complexity, schedulability and decision points. The experimental parameters used for analysis are total productive time and number of decision points.The total productive time is defined as the sum of actual execution times of all the finished HI criticality jobs and LO criticality jobs in a hyperperiod. For evaluation purposes, it is normalized with hyperperiod. The number of decision points is the total number of decisions a core takes within the hyperperiod. For each core, the number of decision points include all arrivals and departures of HI criticality jobs and admitted LO criticality jobs. For evaluation purposes, it is normalized with total number of jobs and total productive time. SimMCSched implements SMILEY, EDF-VD and CBEDF algorithms.

VI. Experimental Results

In this section, we compare the experimental results of SMILEY with two existing algorithms. The following graphs exemplify the results obtained.

A. Unproductive Time

In Fig. 3, Unproductive Time (in percentage) is plotted against the utilization of LO criticality jobs (LO Utilization). We define Unproductive Time as

$$\textit{Unproductive Time} = \frac{\textit{Hyperperiod} - \textit{Total Productive Time}}{\textit{Time available for LO execution}}$$

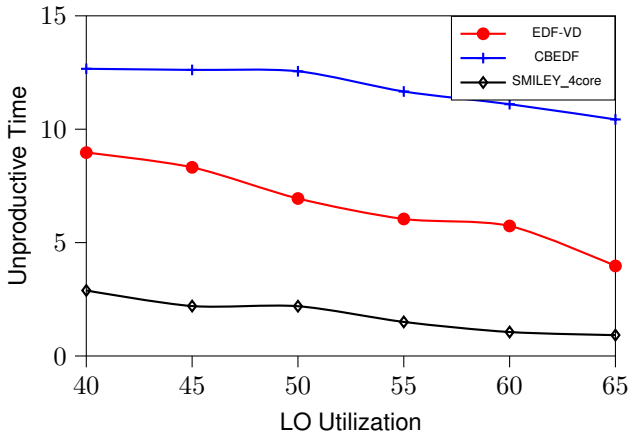


Fig. 3: LO Utilization vs Unproductive Time

The performance of algorithm is better when the value of Unproductive Time is low. Irrespective of task-set, SMILEY offers the least Unproductive Time compared to other two algorithms. It is observed that 73.9% and 85.2% of saving in Unproductive Time is obtained in comparison with EDF-VD and CBEDF respectively. This is because of the increase in number of finished LO criticality jobs in SMILEY.

All the algorithms show negative slope with increase in LO Utilization, which means that the Unproductive Time gets reduced. For lower LO Utilization values, the number of jobs available for filling the Unproductive Time is lesser compared to that of increased utilization.

B. Decisions Per Job

Fig. 4 shows the plot of the number of (in percentage) decision points per job (Decisions per Job) against LO Utilization.

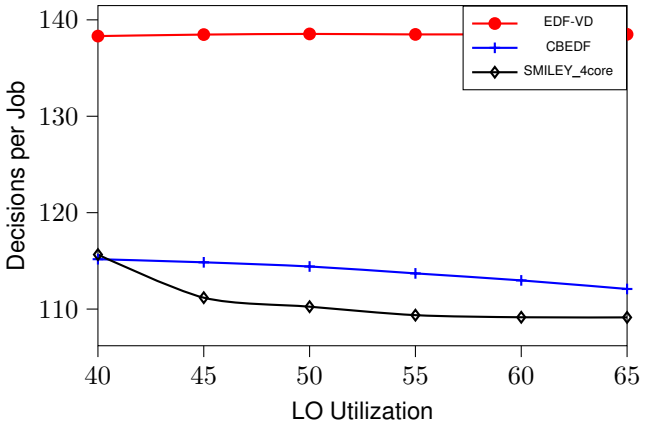


Fig. 4: LO Utilization vs Decisions per Job

Irrespective of algorithm, Decisions per Job shows slight reduction with increase in LO Utilization. This is because, many LO criticality jobs gets rejected with increase in LO Utilization. SMILEY is observed to have 20% and 2.7% lesser values for Decisions per Job when compared with EDF-VD and CBEDF respectively.

C. Decisions Per Productive Time

The plot between the number (in percentage) of decision points per productive time (Decisions per Productive Time) and LO Utilization is shown in Fig. 5.

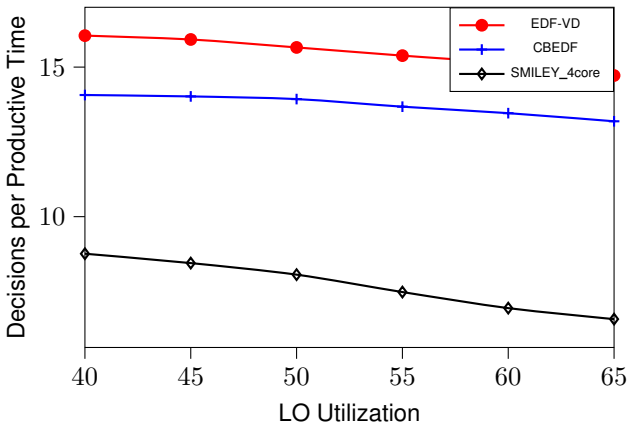


Fig. 5: LO Utilization vs Decisions per Productive Time

Irrespective of algorithm, Decisions per Productive Time decreases with increase in LO Utilization. When the utilization increases, many jobs get rejected at first instance itself. SMILEY is observed to have 50.3% and 43.9% lesser values when compared to EDF-VD and CBEDF respectively. SMILEY has lesser number of decisions as it inserts a LO criticality job in the ready queue only if the job can be completed.

VII. Conclusions

This work proposed SMILEY, a mixed-criticality scheduling algorithm for multicore systems. The results show that SMILEY outperform widely used mixed-criticality scheduling algorithms like EDF-VD and CBEDF. SMILEY along with the other algorithms were simulated for uniprocessor and multicore platforms.

It is observed that there is an increase in productive time for SMILEY. The unproductive time in SMILEY has reduced by 73.9% and 85.2% when compared with EDF-VD and CBEDF respectively. SMILEY has 20% and 2.7% lesser values for the number of decision points taken per job correspondingly. Likewise, the number of decision points taken per productive time has also reduced by 50.3% and 43.9%. Essentially, SMILEY tries to include maximum number of LO criticality jobs and thus maximizing the productive time without compromising the execution of HI criticality jobs.

References

[1] S. Baruah, "Mixed-criticality scheduling theory: Scope, promise, and limitations," *IEEE Design Test*, vol. 35, no. 2, pp. 31–37, April 2018.

[2] S. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," in *2008 Euromicro Conference on Real-Time Systems*, July 2008, pp. 147–155.

[3] A. Burns and R. I. Davis, "Mixed criticality systems-a review ," 2015.

[4] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, Dec 2006, pp. 347–358.

[5] N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram, "A case for guarded power gating for multi-core processors," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, Feb 2011, pp. 291–300.

[6] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, Dec 2007, pp. 239–243.

[7] S. K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *Proceedings of the 19th European Conference on Algorithms*, ser. ESA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 555–566. [Online]. Available: http://dl.acm.org/citation.cfm?id=2040572.2040633

[8] S. Baruah, V. Bonifaci, G. DAngelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *2012 24th Euromicro Conference on Real-Time Systems*, July 2012, pp. 145–154.

[9] T. Park and S. Kim, "Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems," in *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, Oct 2011, pp. 253–262.

[10] H. Li and S. Baruah, "An algorithm for scheduling certifiable mixed-criticality sporadic task systems," in *2010 31st IEEE Real-Time Systems Symposium*, Nov 2010, pp. 183–192.

[11] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in *2010 10th IEEE International Conference on Computer and Information Technology*, June 2010, pp. 1864–1871.

[12] H. Li and S. Baruah, "Outstanding paper award: Global mixed-criticality scheduling on multiprocessors," in *2012 24th Euromicro Conference on Real-Time Systems*, July 2012, pp. 166–175.

[13] 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, June 2004.

[14] J. Lee, K. Phan, X. Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee, "Mc-fluid: Fluid model-based mixed-criticality scheduling on multiprocessors," in *2014 IEEE Real-Time Systems Symposium*, Dec 2014, pp. 41–52.

[15] P. Holman and J. H. Anderson, "Adapting pfair scheduling for symmetric multiprocessors," *J. Embedded Comput.*, vol. 1, no. 4, pp. 543–564, Dec. 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1233791.1233800

[16] J. Ren and L. T. X. Phan, "Mixed-criticality scheduling on multiprocessors using task grouping," in *2015 27th Euromicro Conference on Real-Time Systems*, July 2015, pp. 25–34.

[17] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, "Scheduling of mixed-criticality applications on resource-sharing multicore systems," in *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)*, Sept 2013, pp. 1–15.

[18] D. Johnson, A. Demers, J. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," vol. 3, pp. 299–325, 12 1974.