Semi-Clairvoyant Scheduling for Jobs with Multiple Criticalities

Kunal Agrawal
Washington University in St. Louis, MO, USA

Tung Duc Thai
Washington University in St. Louis, MO, USA

Jinhao Zhao
Washington University in St. Louis, MO, USA

Abstract

This paper considers scheduling jobs semi-clairvoyantly in mixed-criticality systems. In the semi-clairvoyant model, unlike the non-clairvoyant model, we know the WCET mode of the job at job arrival. Prior work has only considered this model for dual criticality jobs. We consider this problem for an arbitrary number of criticalities. We prove that there exist semi-clairvoyant schedulers that guarantee schedulability with speedup $\frac{2^m-1}{2^{m-1}}$ for systems with m criticality levels. In addition, we prove that this bound is tight by providing a job set construction that requires this speed for any semi-clairvoyant scheduler. Finally we provide a linear programming formulation that optimally schedules the semi-clairvoyant system of jobs. The number of variables and constraints is polynomial in the number of jobs, but exponential in the number of criticalities.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Software and its engineering → Real-time schedulability; Computer systems organization → Embedded and cyber-physical systems; Mathematics of computing → Linear programming; Theory of computation → Scheduling algorithms

Keywords and phrases mixed-criticality, semi-clairvoyance, schedulers, linear programming

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2026.21

Funding *Kunal Agrawal*: NSF CCF-2106699, CCF-2107280, PPoSS-2216971.

1 Introduction

The idea of mixed-criticality(MC) scheduling was introduced several years ago [23] to improve the performance and utilization of systems in the presence of timing unpredictability. While real-time schedulers rely on worst case execution time (WCET) estimates to achieve safety and predictability, these estimates are often extremely pessimistic. Mixed-criticality models allow the specification of multiple WCET parameters, one for each criticality level, which allows a finer granularity of information to be provided to the scheduler.

In this model, each job is assigned a ***static criticality*** level, which indicates the importance of the job. In particular, each job J_q has a static criticality level $\chi(J_q)$ between 1 and m as well as $\chi(J_q)$ WCETs $e^1(J_q), e^2(J_q), \ldots, e^{\chi(J_q)}(J_q)$ such that $e^{x+1}(J_q) \geq e^x(J_q)$. If any job J_q's actual work exceeds its $(x-1)$th WCET estimate $e^{x-1}(J_q)$, but does not exceed its xth criticality WCET estimate $e^x(J_q)$, then we say that the job's ***runtime mode*** is $\rho(J_q) = x$. If the highest runtime mode of any job is x, then the ***system criticality*** Φ is said to be x. In this case, jobs with static criticality smaller than x need not execute, but all deadlines for jobs with static criticality x or greater must meet their deadlines.

In the standard mixed-criticality model, the actual execution time of the job is revealed when the job finishes executing – the only way to know if a job will exceed x-th criticality WCET $e^x(J_q)$ is to execute it until it has exceeded this quantity or completed; in other words, the job's execution criticality is revealed as the job executes. More recently, researchers have

considered a slightly different model, called ***semi-clairvoyant*** mixed-criticality model [2] . In this model, the job's runtime mode $\rho(J_q)$ is revealed on arrival. This definition makes sense if the execution time depends on input parameters which may be known at arrival or if the job has several possible implementations one of which is chosen at arrival.

Agrawal et al [2] introduced this model and primarily considered it for 2 criticality levels. They showed a tight speedup bound of 3/2. They also provided a linear programming based algorithm to check for semi-clairvoyant schedulability and show that this algorithm is optimal – if any semi-clairvoyant scheduler guarantees schedulability to a system of jobs, then so can this scheduler. Semi-clairvoyance and its variants have since been considered extensively [10] in the literature. However, theoretical work on semi-clairvoyance seems restricted to 2 criticality levels. In this paper, we completely generalize these results to more than two criticality levels.

1. In Section 3, we show that given jobs with m distinct criticalities, no semi-clairvoyant scheduler can guarantee a speedup of less than $(2^m - 1)/2^{m-1}$ – this reduces to 3/2 for two criticality levels.
2. In Section 4, we show that there exist semi-clairvoyant schedulers which can achieve this speedup bound – in other words, the speedup bound of $(2^m - 1)/2^{m-1}$ is tight.
3. In Section 5, we provide linear programming formulation which allows us to check the schedulability of semi-clairvoyant job systems exactly. It is important to note that this LP-based scheduling algorithm is an optimal semi-clairvoyant algorithm (not just speedup optimal). In other words, given system of jobs, if any semi-clairvoyant scheduler can schedule this system, then this LP-based algorithm can also schedule it. If it does not find a solution, then we know that the system is not schedulable.

These results starkly indicate the advantage of semi-clairvoyance in mixed-criticality scheduling over the traditional model of non-clairvoyance. There are no tight speedup bounds known for non-clairvoyant systems; and all known bounds scale approximately linearly or slightly sublinearly with the number of criticality levels. On the other hand, semi-clairvoyant systems can be optimally scheduled and require speedup of less than 2 for arbitrary criticality levels.

2 Preliminaries

Mixed-Criticality Jobs. A problem instance Δ consists of n jobs where every $J_q \in \Delta$ has the following parameters: (1) $\chi(J_q) \in \{1, 2 \ldots, m\}$ denotes the *static criticality* or importance of the job. (2) $a(J_q)$ denotes the arrival time of the job. (3) $d(J_q)$ denotes the deadline of the job. (4) $[e^1(J_q), e^2(J_q), \ldots, e^{\chi(J_q)}(J_q)]$ denotes the worst case execution times (WCETs) of the job for the different modes.[1] Each job J_q in Δ is released at time $a(J_q)$, has deadline $d(J_q)$ and executes for some duration $\gamma(J_q)$. If the $e^{x-1}(J_q) < \gamma(J_q) \leq e^x(J_q)$, then the **runtime mode** of J_q is $\rho(J_q) = x$. We say the ***system criticality*** is the highest runtime mode of any job.

Correctness criteria. A scheduling algorithm is said to schedule Δ correctly iff the following condition is true. **All jobs with static criticality at or above the highest system criticality ever reached must meet their deadlines.** In particular, if the system criticality Φ is x at some time, but is never $x + 1$, then all jobs with static criticality $\chi_q \geq x$ must meet their deadlines. For instance, if no jobs exceed their $e^1(J_q)$, then all jobs must meet their deadlines. If some job J_q exceeds $e^{x-1}(J_q) < \gamma(J_q) \leq e^x(J_q)$ and no job has $\gamma(J_q) > e^x(J_q)$, then all jobs with (static) criticality x or above must meet their deadlines, but no guarantees are needed for jobs with static criticality smaller than x.

[1] We use $\chi(J_q)$ and χ_q interchangeably as appropriate – similarly for other parameters.

Clairvoyant Schedulability. Clairvoyant Scheduler is an idealization in MC system that it can know all the exact execution time γ before the arrival of any jobs. A clairvoyant scheduler can ignore all jobs with static criticality lower than j (where j is the highest execution criticality of any job) and run all other jobs using EDF. As a well-known result in EDF, a job set is schedulable iff for all intervals $[t_i, t_j]$, the total work of jobs with arrival time $\geq t_i$ and deadline $\leq t_j$ is at most $t_j - t_i$. For mixed criticality systems, this means that for all criticality levels x and all intervals $[t_i, t_j]$, the following holds: Say X is the set of jobs with criticality level at least x with arrival time $\geq t_i$ and deadline $\leq t_j$. The sum of $e^x(J_q)$ for all $J_q \in X$ is at most $t_j - t_i$.

Clairvoyant schedulers are generally used to characterize the performance of other schedulers. In particular, speedup factor is the typical metric that is used to compare different MC scheduling algorithms. Formally, an algorithm A has speedup factor f, if, given any instance that is schedulable on a unit-speed processor by optimal clairvoyant algorithm, A can schedule the same instance on a speed-f processor.

Non-clairvoyant Schedulability. Most work for scheduling MC systems is done under the non-clairvoyance assumption where γ values for jobs becomes known as the jobs execute. That is, we do not know the exact runtime mode for a job until the job finishes executing. Non-clairvoyant schedulability is known to be NP-complete for both jobs and tasks even for dual criticality systems [1].

Semi-clairvoyant schedulability. In this paper, we study semi-clairvoyant scheduling for MC systems. A semi-clairvoyant scheduler knows the job's static parameters (its release time, deadline, static criticality, and worst case execution time for all criticality modes) in advance. However, unlike the clairvoyant scheduler, it does not know the job's actual execution time and therefore, the job's or the system's runtime mode in advance. When a job J_q arrives at time a_q, its runtime mode is revealed to the scheduler, but the scheduler still does not know its exact execution time γ. Formally, it knows a mode x such that $e^{x-1}(J_q) < \gamma(J_q) \leq e^x(J_q)$.

Therefore, if a job J_q arrives with runtime mode $\rho(J_q) = x$ at time a_q and the system criticality Φ is smaller than x at this time , then a semi-clairvoyant scheduler can immediately increase the system criticality Φ to x. An instant at which an online MC system changes (increases) it system mode Φ is called a ***mode switch*** time. At this time, the MC scheduler immediately stop executing all jobs J_p with static criticality $\chi(J_p) < x$ since the correctness condition no longer requires that it finish these jobs. We denote the time instance for a mode switch to runtime mode α as k_α. We know that $0 = k_1 \leq k_2 \leq \ldots \leq k_\Phi$.

Difference between semi-clairvoyance and non-clairvoyance. For non-clairvoyant schedulers, a mode switch occurs and the system criticality Φ increases to x when some job J_q executes for more than $e^{x-1}(J_q)$ time since this is the time when the scheduler discovers that runtime mode of this job. Therefore, the mode switch can happen at any time during the schedule. In contrast, mode switches always occur at arrival times for semi-clairvoyant schedulers since the runtime mode of a job is known at arrival. This difference shows up in complexity results. For dual-criticality jobs, semi-clairvoyant schedulability can be checked in polynomial time by using a linear program while non-clairvoyant schedulability is NP-Hard. In addition, the speedup bounds are also different; for dual-criticality jobs, semi-clairvoyant jobs need a speedup of 3/2 relative to clairvoyant schedulability while non-clairvoyant systems require a speedup of about 1.62.

Table 1 The set of jobs that shows that any semi-clairvoyant scheduler needs a speedup of $2 - 1/2^{m-1}$ in order to schedules jobs with m different criticalities.

Jobs	χ_p	a_p	e_p^1	e_p^2	e_p^3	...	e_p^{m-2}	e_p^{m-1}	e_p^m	d
J_1	1	0	1	-	-	...	-	-	-	1
J_2	2	0	1	1	-	...	-	-	-	2
J_3	3	0	2	2	2	...	-	-	-	2^2
...	...	...	...	...	...	...	...	...	...	...
J_{m-1}	$m-1$	0	2^{m-3}	2^{m-3}	2^{m-3}	...	2^{m-3}	2^{m-3}	-	2^{m-2}
J_m	m	0	2^{m-2}	2^{m-2}	2^{m-2}	...	2^{m-2}	2^{m-2}	2^{m-2}	2^{m-1}
J_{m+1}	2	1	0	1	-	...	-	-	-	2
J_{m+2}	3	2	0	0	2	...	-	-	-	2^2
...	...	...	...	...	...	...	...	...	...	...
J_{2m-2}	$m-1$	2^{m-3}	0	0	0	...	0	2^{m-3}	-	2^{m-2}
J_{2m-1}	m	2^{m-2}	0	0	0	...	0	0	2^{m-2}	2^{m-1}

3 Lower Bound

In this section, we will show that given a feasible set of semi-clairvoyant jobs with m criticality levels, no online algorithm can guarantee schedulability with speedup smaller than $s = (2^m - 1)/2^{m-1}$. We will argue this by creating a set of jobs which are schedulable by a clairvoyant scheduler, but needs speed s to schedule online by any semi-clairvoyant scheduler.

The set of $2m - 1$ jobs are shown in Table 1. We can divide these jobs into two groups. The first m jobs $J_1, J_2, \ldots, J_m$ are called ***initial jobs*** – they all arrive at time 0. Here are the characteristics of these initial jobs: job J_q has deadline 2^{q-1}, static criticality q, and work (at all criticality levels) 2^{q-2} (Except the first job whose work at criticality level 1 is just 1). The second set of $m - 1$ jobs, namely $J_{m+1} \ldots, J_{2m-1}$, are the ***later jobs***. These are more interesting. Job J_{m+q} has arrival time 2^{q-1}, deadline 2^q and static criticality $q + 1$. Each of these jobs has work 0 at all criticality levels except at its highest criticality level – that is job J_{m+q} has $e_1, \ldots, e_q = 0$ and $e_{q+1} = 2^{q-1}$.

First, we argue that this set of jobs is schedulable by a clairvoyant scheduler.

▶ **Lemma 1.** *The set of jobs shown in Table 1 can be scheduled by a clairvoyant scheduler.*

Proof. The clairvoyant scheduler knows the criticality level of the system. Therefore, we just have to argue that the total work that the optimal scheduler has to do before every deadline is feasible. Note that the system criticality is determined by the execution time of the later jobs since we can assume that the initial jobs always arrive at criticality level 1. Intuitively, the system of jobs is designed so that at any system criticality , only one of the later jobs takes up any execution time and its work is equal to the work of the lower criticality jobs that need not execute. More concretely, the clairvoyant scheduler works as follows:

1. If the system remains at criticality level 1 – that is, $\gamma(J_{m+1}) = \gamma(J_{m+2}) = \ldots = \gamma(J_{2m-1}) = 0$ – then the scheduler has to execute only the initial jobs. Now consider any set of jobs $J_1, \ldots, J_q$ – the deadlines of these jobs are all before 2^{q-1} and the collective work of these jobs is also 2^{q-1}. Therefore, the clairvoyant scheduler can complete this work in time using EDF. Concretely, the system executes job J_q in the interval $[2^{q-1}, 2^q]$.
2. if the system remains at criticality level 2, it can drop the first job. In addition, $\gamma(J_{m+1}) = 1$, but $\gamma(J_{m+2}) = \ldots = \gamma(J_{2m-1}) = 0$. Therefore, it can execute J_2 and J_{m+1} by time 2 and then follow the same schedule outlined above, scheduling J_q in the interval $[2^{q-1}, 2^q]$.

3. More generally, if the system is at criticality level j, for $j > 1$, then it need never execute jobs $J_1, \ldots, J_{j-1}$ as well as jobs $J_{m+1}, \ldots, J_{m+j-2}$. In this case $\gamma(J_{m+j-1}) = 2^{j-2}$, but $\gamma(J_{m+j}) = \ldots = \gamma(J_{2m-1}) = 0$. Therefore, the clairvoyant scheduler can execute job J_j by time 2^{j-2}. At this time job J_{m+j-1} arrives and can be executed by time 2^{j-1}. The remaining schedule remains the same as above. ◀

▶ **Lemma 2.** *No semi-clairvoyant scheduler with speed less than $s < (2^m - 1)/2^{m-1}$ can schedule this system of jobs under all arrival conditions.*

Proof. Say this set of jobs is executed by a semi-clairvoyant scheduler on a processor of speed s. The initial criticality of the system is 1. Now we consider each interval $[0, 1)$ and $[2^q, 2^{q+1})$ for every q from 0 to $m - 1$:

- Over the interval $[0, 1)$, the first m jobs arrive. J_1 must receive 1 unit of execution (otherwise it will miss its deadline). The remaining $s - 1$ unit of execution could be used to execute any other initial jobs.
- Over the interval $[1, 2)$, J_{m+1} arrives. J_{m+1} with static criticality $\chi_{m+1} = 2$ (and work $\gamma(J_{m+1}) = 1$) increasing the criticality level of the system to 2. Both J_2 and J_{m+1} both have a deadline of 2; therefore, the semi-clairvoyant scheduler must complete 3 units of work (belonging to J_1, J_2, J_{m+1}) by time 2.
- At time 2, job J_{m+2} arrives with static criticality $\chi_{m+2} = 3$ and work $\gamma(J_{m+2}) = 2$. Again, the system criticality rises to 3 at this time, but both criticality 3 jobs must be completed by time 4 since that is their deadline. Therefore, the semi-clairvoyant scheduler must complete at least 7 units of work by time 4.
- More generally, at time 2^{j-1}, the system criticality increases to j due to the arrival of job J_{m+j-1} with its runtime mode at the level j. But by this time, all the jobs with lower criticality have already finished executing since they all have earlier deadlines.

Therefore, over the entire interval, up to time 2^{m-1}, the total work that is executed is the sum of the work of all initial jobs as well as the maximum work of the later jobs. The total work of the initial jobs is $1 + \sum_{q=0}^{m-2} 2^q = 2^{m-1}$. In addition, the later jobs always arrive at their highest criticality level; therefore, the total work due to the later jobs is $\sum_{q=0}^{m-2} 2^q = 2^{m-1} - 1$. Therefore, the semi-clairvoyant scheduler must do the total work of $2^m - 1$ in time 2^{m-1}. This means the speedup required is at least $(2^m - 1)/2^{m-1}$. ◀

4 Upper Bound

We will now show that there exist semi-clairvoyant schedulers which provide the speedup bound of $s = (2^m - 1)/2^{m-1}$. For the purposes of this proof, we assume that we have a job set Δ which is schedulable by a clairvoyant algorithm OPT on a processor of speed 1. We will define a theoretical semi-clairvoyant scheduler and prove that it can schedule any such job set on a processor of speed s. Since we already showed that there exist instances for which no semi-clairvoyant scheduler can provide a better speedup bound, we can conclude that this bound is tight.

While the scheduler we use in this section does not work when the speedup is $< s$, in the next section, we will provide a linear programming based semi-clairvoyant algorithm which is an optimal semi-clairvoyant scheduler for any instance and therefore also provides the stated speedup bound.

Proof Intuition

Let us first understand the high-level intuition and overview of how this proof will proceed. The details are in subsequent subsections.

1. We will first define a scheduling algorithm called *EDF-Zeno* – nominally, this algorithm pretends that a processor of speed $s = (2^m - 1)/2^{m-1}$ is split into m processors (called ***lanes***) with where the first lane has speed 1, the second 1/2, and so on; therefore, collectively, these lanes have speed s. Each of these processors run EDF, but (potentially) on different subsets of jobs. This scheduler also uses "flow control" in order to ensure that low criticality jobs do not get too much processing at the cost of higher criticality jobs.
2. We also apply a transformation called *zipping* on the job set. In particular, given a set of jobs Δ, we transform them into a different set of jobs, say Δ_{zip}, and prove that proving the speedup bound on this transformed set of jobs will prove it for the original set of jobs.
3. We now get into the nitty-gritty of the proof. We assume for contradiction that EDF-Zeno cannot correctly schedule some instance and identify the first deadline miss, say for job J_{miss} and assume that this happens when the system is in some system criticality Φ.
4. We identify the *busy period* preceding this deadline miss for all individual lanes. The definition of the busy period is somewhat complicated due to two reasons (1) each lane schedules different subsets of jobs; and (2) the scheduler on each lane is not quite pure-EDF due to flow control.
5. We then identify a subset of "bottleneck" jobs – these are the jobs that can potentially prevent our job J_{miss} from completing by its deadline. We then prove that if this set of bottleneck jobs is schedulable, then the original set of jobs is also schedulable. Therefore, we can restrict our attention to just this set of jobs.
6. This allows us to bound the interference that our job J_{miss} can experience during the busy period. However, this interference calculation is also complicated due to several factors: (1) some jobs may interfere in some lanes but not others; (2) the flow control mechanism allows us to bound some of the interference; and (3) lower criticality jobs sometimes stop interfering after mode switches.
7. After considering all these factors, we are able to bound the interference on the job and show that the interference is strictly smaller than the total execution capacity of the busy period. This gives us a contradiction due to the definition of the busy period.

Definition of EDF-Zeno

For this proof, we will use a relatively unnatural scheduling algorithm which is designed purely to make the proof tractable. We will assume that this scheduler always runs (semi-clairvoyantly) on a machine with speed $s = (2^m - 1)/2^{m-1}$. EDF-Zeno has two aspects: *division into lanes* and *flow control*. We will define both here.

▶ **Definition 3** (EDF-Zeno's Lane Policy). *For a machine with $s = (2^m - 1)/2^{m-1}$ speedup, EDF-Zeno divides it into m lanes, where lane L_i has speed $s_i = 1/2^{i-1}$ – lane L_1 has speed 1 and subsequent speeds decrease geometrically by a factor of 1/2.*[2]

A job with criticality β is only eligible to run on lanes L_1 through L_β. In other words, all jobs are eligible to run on L_1, but only jobs with criticality level 2 and above can run on L_2, and so on. Within each lane EDF-Zeno uses EDF to run jobs that are eligible to run on this lane.[3]

[2] In a sense, these lanes are time-sharing the processor and dividing up the speed of the processor among themselves.

[3] If a job is simultaneously running on multiple lanes, then it is equivalent to running the job at the speed that is the sum of those lanes' speed.

Here's the intuition for this policy: Recall that if we only had a 2-criticality system, then the speed of 1.5 is enough. Therefore, EDF-Zeno allows the jobs of the lowest two criticalities to essentially only use a machine with speed 1.5. In general, if we collectively consider the jobs of the lowest β criticalities, they are only using a machine of speed $(2^\beta - 1)/2^{\beta-1}$.

EDF-Zeno has another policy to restrict the execution of "current" criticality jobs. In particular, after the mode switch to criticality level β, jobs of criticality level β become the lowest criticality jobs. However, they are still allowed to use the first β lanes. Therefore, if a lot of criticality β work arrives after this mode switch, it can overuse the machine and prevent higher criticality jobs from running. Therefore, we restrict the execution provided to these jobs using the a ***flow control*** policy.

▶ **Definition 4** (EDF-Zeno flow control). *Say that the mode switch to criticality level β happens at time k_β. Consider the set of all jobs X which have static criticality level β and they arrived after time k_β. At any time $t > k_\beta$, the total work done by EDF-Zeno on jobs in set X is at most $t - k_\beta$.*

We make a quick observation about the impact of flow control on various lanes:

▶ **Observation 5.** *Say, at time t, a set of jobs X of criticality level β are being flow-controlled. At time t, these jobs can run at speed at most 1. Therefore, if one of these jobs is the earliest deadline job, it can still run on lane 1. Therefore, lane 1 still follows EDF on jobs which are still in the system. However, flow-controlled jobs cannot run on any other lane.*

Zipping the Job Set

In order to simplify the proof, we will apply a transformation to the job set – this transformation guarantees that each job has at most two "interesting" criticality levels.

▶ **Definition 6** (Zipped problem set). *Given a set of jobs Δ, consider each job $J_q \in \Delta$ with static criticality $\chi(J_q) = \beta$ and work estimates $e^1(J_q), e^2(J_q), \ldots, e^\beta(J_q)$. In Δ_{zip}, $\beta - 1$ jobs, $J_q^2, J_q^3, \ldots, J_q^\beta$ (with the same release time and deadline as J_q), are created as follows: For $i = 2, 3, \ldots, \beta - 1$, if $e^i(J_q) - e^1(J_q) > 0$, job J_q^i has (static) criticality level i and $e^1(J_q^i) = e^2(J_q^i) = \ldots = e^{i-1}(J_q^i) = 0$, and $e^i(J_q^i) = e^i(J_q) - e^1(J_q)$; if $e^i(J_q) - e^1(J_q) = 0$, J_q^i is skipped. Finally, J_q^β has (static) criticality level β and $e^1(J_q^\beta) = e^2(J_q^\beta) = \ldots = e^{\beta-1}(J_q^\beta) = e^1(J_q)$, and $e^\beta(J_q^\beta) = e^\beta(J_q)$.*

Essentially, we create jobs such that their execution times for most of the criticality levels are 0. For example, if we are given a job J_q with static criticality $\beta = 4$ such that its execution times at levels $1, \ldots, 4$ was a, b, c, d, then we would have 3 jobs: (1) J_q^2 with static criticality 2 and work estimate at criticality levels 1 and 2 as 0 and $b - a$ respectively; (2) J_q^3 with static criticality 3 and estimate $0, 0, c - a$ respectively; and (3) J_q^4 static criticality 4 with estimates a, a, a, d respectively. This transformation simplifies the reasoning in the proof. We will now prove that this transformation is valid.

▶ **Lemma 7.** *If Δ is feasible, then Δ_{zip} is also feasible (on the processor of the same speed).*

Proof. As a stronger conclusion, we prove the equivalence in feasibility. That is, OPT can either schedule both job sets or neither. Consider any single criticality level β. The total work of J_q if it arrives in criticality level β is equal to the work of J_q^i's collectively in criticality level β by construction. Since they all share release times and deadlines, the total load within any interval remains the same for every criticality level. Since the feasibility of clairvoyant schedulers depends only on the workload within each time interval and at every criticality level, feasibility of the two systems is equivalent. ◀

We now consider semi-clairvoyant schedulability. This is less straightforward since semi-clairvoyant schedulability depends on the actual scheduler.

▶ **Lemma 8.** *If some semi-clairvoyant scheduler can schedule Δ_{zip}, then an* optimal *semi-clairvoyant scheduler can also schedule Δ (on the processor of the same speed). Therefore, if EDF-Zeno can schedule Δ_{zip}, then the optimal semi-clairvoyant scheduler can schedule Δ.*

Proof. Say that some semi-clairvoyant schedule, $A(\Delta)$, can correctly schedule Δ. We can now simulate the execution of Δ_{zip} – call this scheduler $B(\Delta_{\text{zip}})$. Say job J_q is released in runtime mode k when executing Δ and therefore, has execution requirement $e^k(J_q)$. Therefore, in Δ_{zip} all J_q^is will also be released on mode k which indicates the following: (1) if $k = 1$ or $k = \chi(J_q) = \beta$, then in Δ_{zip}, J_q^β has work $e^k(J_q)$ and other J_q^i don't have any work; (2) otherwise, J_q^β has work $e^1(J_q)$ and J_q^k has work $e^k(J_q) - e^1(J_q)$. Therefore, the total work is the same. Therefore, any time the scheduler $A(\Delta)$ runs J_q, $B(\Delta_{\text{zip}})$ runs one of the corresponding jobs. Therefore, both schedulers do exactly the same amount of work on J_q for all q. Therefore if $A(\Delta)$ meets all the relevant deadlines, then so does $B(\Delta_{\text{zip}})$. ◀

Note that the above lemma doesn't say anything about EDF-Zeno; in particular, we have not argued that if EDF-Zeno can schedule Δ_{zip} then it can also schedule Δ. As we will argue later, this is sufficient for us to prove the speedup bound we need.[4] The zipped problem set has the following property which will be helpful for our purposes.

▶ **Observation 9.** *In Δ_{zip}, a job with static criticality i is released in runtime mode either* 1 *or i.*

For the rest of this section, we will treat Δ_{zip} as Δ, our job set under consideration.

Busy Period

We are now ready to define some terms to prove the following theorem:

▶ **Theorem 10.** *A feasible zipped problem Δ_{zip} with m criticality levels is schedulable by EDF-Zeno with m lanes.*

Note that the behavior of a semi-clairvoyant scheduler depends on the timing of mode switches – that is, as jobs arrive in different criticality levels, the scheduler responds accordingly. Assume for contradiction that there is some runtime behavior R for which EDF-Zeno misses a deadline – and say J_{miss} is the first job in this runtime behavior which misses its deadline. Let the arrival time of J_{miss} be a, the deadline of J_{miss} be d, and the static criticality of J_{miss} is α. Also, assume the system is at system criticality Φ at time d. We know that $\Phi \leq \alpha$.

We can now ignore the behavior of the system after the time d.

Therefore, the system has gone through a series of mode switches at times $0 = k_1 \leq k_2 \leq \ldots \leq k_\Phi$. We can ignore all mode switches after Φ.

We now define the interesting period of execution, namely the *busy period* before d. Intuitively, the busy period contains jobs which interfered with the execution of job J_{miss} – this sort of busy period analysis is common in real-time systems. However, in this proof, this definition is somewhat involved since (1) the busy period can be different for each lane since different subsets of jobs are eligible to run in each lane; and (2) flow control causes scheduling anomalies which make the definition complicated.

[4] Alternatively, we can also transform arbitrary Δ into Δ_{zip} and just run EDF-Zenoon the corresponding Δ_{zip}. As we will prove that EDF-Zenocan schedule Δ_{zip}, this transformation means that Δ is schedulable.

▶ **Definition 11** (Busy period). *Say J_{early} is the set of jobs with deadline $\leq d$ and J_{late} are the remaining jobs. For each lane $\beta > 1$, we look backward from d, and find the latest time instance that is not running J_{early} (i.e., running J_{late} or idle). Let this time instance be b_β. The busy period for lane $\beta > 1$ is the interval $[b_\beta, d]$. For lane 1, the definition is a bit more complicated. We go backwards from d and find the latest instance when either lane 1 is not running J_{early}, or some J_{late} was scheduled on some other lane $\beta \geq 2$ while J_{early} were eligible for lane 1 (this only happens when job running on lane 1 is subject to flow control on other lanes) (see Observation 5).*

Note that $b_1, b_2, \ldots, b_m$ must exist, because time 0 always fits the conditions. We can now observe some structural properties of these busy periods.

▶ **Observation 12.** *Only jobs with deadline $\leq d$ are executed on a lane within its busy period.*

Moreover, the starting points of each busy period are nicely ordered:

▶ **Lemma 13.** $b_1 \leq b_2 \leq \ldots \leq b_m$.

Proof. We only need to prove for any β, $b_\beta \leq b_{\beta+1}$. Assume for contradiction that $b_\beta > b_{\beta+1}$. At any time in interval $[b_{\beta+1}, b_\beta]$, lane $\beta + 1$ is working on some job in J_{early} (Definition 11). But this job is also eligible to run on lane β (Definition 3), and it is not subject to flow control (since it is running on lane $\beta + 1$); therefore, lane β cannot be idle or working on J_{late} since it uses EDF. Hence a contradiction. ◀

Now we are ready to reduce the set of jobs that we must consider – nominally, these are jobs that execute within the busy period and therefore interfere with J_{miss}. However, again, the definition is somewhat complex due to the various lanes and due to flow control.

▶ **Definition 14** (Bottleneck Jobs). *We will define* bottleneck jobs B_β *for each criticality level $1 \leq \beta \leq m$. A job belongs to B_β if (1) its static criticality level $\geq \beta$; and (2) it has a deadline $\leq d$; and (3) it has not completed by time b_β. In addition, we will* truncate *both the interval and the work of some of these jobs. If a bottleneck job J_q arrived before b_1 (the start of the busy period of lane 1), then its release time is modified to b_1. In addition, if this job completed some execution say w before b_1, then its execution requirement is reduced by this quantity w. Call the (sub)set of bottleneck jobs that were truncated T. After truncation, we have the property that all bottleneck jobs have arrival time $\geq b_1$ and deadlines $\leq d$.*

A quick observation is that $B_m \subseteq \ldots \subseteq B_2 \subseteq B_1$. Precisely, B_β are the jobs that possibly interfere with J_{miss} on lane β. When we refer to bottleneck jobs without β, we mean B_1. We now show that the truncated jobs are a limited set of jobs.

▶ **Lemma 15.** *A bottleneck job J_q with criticality β is truncated if all three conditions hold: (1) $k_\beta \leq b_1 \leq k_{\beta+1}$ – i.e., the busy period for lane 1 starts during system's criticality β; (2) jobs of lane β were subject to flow control immediately before b_1; and (3) $b_1 = b_2 = \ldots = b_\beta$.*

Proof. Say t is the moment right before b_1. At time t, J_q has arrived, but not completed. Since lane 1 does pure EDF, it is neither idle nor executing a job in J_{late}. Therefore, b_1 was defined due to the second condition of Definition 11 – that some lane was subject to flow control. Say this lane was γ; which implies that $k_\gamma \leq b_1 \leq k_{\gamma+1}$. Since J_q is executing, $\beta \geq \gamma$. Therefore, J_q is eligible to execute on lane γ; but it doesn't. Therefore, J_q must be subject to flow control and $\gamma = \beta$. Finally, since the system is already in criticality level β at time t, lanes 1 through β have the same set of jobs that are eligible to run on them after this time. In addition, this is the last instant in time where b_1 is defined due to flow control. Therefore, due to Definition 11 and Lemma 13, we have $b_1 = b_2 = \ldots = b_\beta$. ◀

We are now ready to prove that we can safely restrict our attention to bottleneck jobs only for the purposes of analysis.

▶ **Lemma 16.** *If the original set of jobs was feasible, then (constructed) bottleneck jobs B_1 are also feasible. That is, if OPT could schedule the original set, it can schedule the bottleneck set B_1. In addition, EDF-Zeno still misses the deadline J_{miss} if only the jobs in B_1 are released.*

Proof. Say O is OPT's schedule on the original set of jobs and O' is OPT's schedule on the bottleneck jobs. Removal of jobs obviously doesn't impact feasibility. Now consider the truncated jobs. From Lemma 15, we know that jobs are only truncated when subject to flow control and all truncated jobs belong to the same criticality, say γ. Therefore, these jobs have done (collectively) $b_1 - k_\gamma$ work at time b_1 (that's why they were flow controlled). Since they were all released after time k_γ, O cannot do more than $b_1 - k_\gamma$ work on these jobs by time b_1. O has at least as much work to do on these jobs as O'; therefore, if O can schedule them, then so can O'.

We now show that if EDF-Zeno misses the deadline for J_{miss} in the original set of jobs, then it also misses it for the bottleneck jobs. Say S is the original schedule and S' is the schedule for bottleneck jobs. First we note that we can safely ignore jobs in J_{late} since they do not interfere with jobs in B_1 on any of the lanes.

Now, if we just look at J_{early}, all jobs that are not completed before time b_1 are included in the bottleneck jobs B_1 (either fully or in a truncated form). Therefore, within interval $[b_1, d]$ S and S' are responsible for the same set of jobs. Also note that jobs are truncated according to their execution in S; therefore, truncated jobs have exactly the same remaining work and the same deadline in S and S'. Therefore, the schedules for S and S' are identical. ◀

From now, we will implicitly assume that we are just running EDF-Zeno on bottleneck jobs. We first prove a structural property that relates the mode switches with the busy period.

▶ **Lemma 17.** *For any $2 \leq \beta \leq \Phi$, if $b_1 \leq k_\beta$, we have $b_\beta \leq k_\beta$.*

Proof. Proof by contradiction. If $b_\beta > k_\beta$, consider the time instance just before b_β – we know that no job in J_{early} with criticality level at least β is available; otherwise lane β would run it. However, since $b_1 \leq k_\beta$, there must be a job in J_{early} running on lane 1 at time t and this job has criticality at least β (because the system is in criticality β, the lower jobs are dropped). It is either eligible for lane β (an immediate contradiction), or it triggers the flow control so that it cannot run on lane β. In the second case, since lane β is not running J_{early}, b_β also matches the condition of Definition 14, which means $b_1 = b_\beta > k_\beta$, which also leads to a contradiction. ◀

We now observe another structural property which relates the mode switch times and the busy period. In particular, we need only focus on the worst case for the purposes of our bound and the following lemma implies that $b_1 \leq k_2$ is the worst case. That is, the busy period for lane 1 starts before the mode switch to criticality level 2. Intuitively, this is easy to see – if $b_1 > k_2$, we functionally only have $m - 1$ criticalities since no (static) criticality level 1 job is part of B_1. Therefore, the problem becomes easier.

▶ **Lemma 18.** *We can assume that $b_1 \leq k_2$ without loss of generality.*

Proof. Say, for contradiction, that $b_1 > k_2$. Let $k_\beta < b_1 \leq k_{\beta+1}$. Now B_1 does not have a job with criticality $< \beta$ since all those jobs were already dropped at time k_β. Create a new problem instance NB, which is basically B_1, but transforms everyone's criticality levels

so that criticality $\beta - 1 + i$ in B_1 acts as criticality i in NB. OPT can still schedule NB; however for EDF-Zeno, each job in NB can use fewer lanes than it could when this job was in B_1. Hence, no job NB finishes before the corresponding job in B_1. Therefore, if NB is at least as hard as B_1; more precisely, if NB is schedulable, then so is B_1. NB has the property that $b_1 < k_2$. ◀

Defining Interference

We now are able to specifically measure the interference experienced by J_{miss} on each processing lane in terms of these job sets $B_1, \ldots, B_m$. We will define several such parameters. In our proof, these are called utilities since they measure partial work of jobs actually completed by EDF-Zeno on specific lanes during the busy period instead of the entire work of the job.

▶ **Definition 19** (Utilities U_x^y)**.** *Consider a schedule of EDF-Zeno on a set of bottleneck jobs B_1. Consider any γ for $1 \le \gamma \le m$. We first define the partial utility $U_x^y(\gamma)$ as the total work done by EDF-Zeno within the busy period on some job $J \in B_\gamma \setminus B_{\gamma+1}$ (formally, $B_{m+1} = \varnothing$) with static criticality x and runtime criticality (mode) y. The utility U_x^y is defined as $U_x^1 = \sum_{\beta=x}^{\alpha} U_\beta^1(x)$, and $U_x^x = \sum_{\beta=x}^{\alpha} U_x^x(\beta)$.*

Note that for all $1 < y < x$, we have $U_x^y = 0$ since we are only considering Δ_{zip} (Observation 9). Consider a job in $B_x - B_{x+1}$ but with static criticality $\beta > x$. This job must have arrived and completed within the interval $[b_x, b_{x+1}]$ (Definition 14). Therefore, we treat it (functionally) as a criticality x job for the purposes of utilities. We can make the following observation:

▶ **Observation 20.** *When collecting partial utilities $U_\beta^1(x)$, each partial utility is counted exactly once to $U^1_{\min\{\beta,x\}}$. Meanwhile for $\beta \ge 2$, all $U_\beta^\beta(x)$ trivially count to U_β^β (and $\beta < x$ results in $U_\beta^\beta(x) = 0$ because before k_β there are no jobs with runtime mode β). Therefore, all the partial utilities are counted exactly once.*

Another important note: One might think that the sum of all the utilities is equal to the sum of the work of all the bottleneck jobs during our execution R. However, it turns out that the sum of utilities might be smaller due to three factors:

- Some jobs may remain incomplete due to a mode switches – therefore, the total work done by EDF-Zeno is smaller than the work of the job.
- More importantly, some jobs may be in $B_1 \setminus B_2$ (or more generally, some jobs are in $B_1, .., B_i$, but not in subsequent lanes) even though its static criticality is greater than 1. This means that this job arrived after time b_1 but completed before b_2, and lane 2 subsequently experienced an idle instant (or worked on J_{late}). Therefore, some of this job's work is included in the busy period of lane 1 and therefore appears in the utilities U_1^1, but any work done by lane 2 doesn't is not included in any of the utilities.
- Finally, recall that J_{miss} belongs to J_{early}, so the completed part of J_{miss} shows up in the utilities. But, since we assume (for contradiction) that J_{miss} misses its deadline, therefore, it has some remaining work when we get to time d. Lets call this uncompleted work δ.

Bounding the utilities

We will now provide bounds on these utilities which will allow us to reach a contradiction. The following two lemma generates $2\alpha - 1$ inequalities due to the different values of β.

▶ **Lemma 21.** *For* $1 \le \beta \le \alpha$,

$$\underbrace{\sum_{\gamma=\beta}^{m} U_\gamma^1}_{Part\ 1} + \underbrace{\mathbb{I}(\beta \neq 1) U_\beta^\beta + \mathbb{I}(\beta = \alpha \vee \rho(J_{miss}) = 1)\delta}_{Part\ 2} \le d - b_\beta.$$

Proof. Suppose the system is in criticality β at time d. We know that the clairvoyant algorithm which runs in system criticality β can finish all relevant jobs in B_β. Lets call this algorithm OPT^β. Since OPT^β never runs any jobs with criticality level $< \beta$, these jobs have arrival time $\ge b_\beta$ and deadline $\le d$ (Definition 14 and Lemma 13). Therefore, the total work of these jobs must be at most $d - b_\beta$ to make OPT^β feasible – this is the RHS of the statement.

Now, lets count up the total work of these jobs. First, consider a job with static criticality $\gamma > \beta$. This job never switched to $\beta + 1$ mode. Since it is a zipped job, it arrived in mode 1. Therefore, the total work of these jobs collectively is at least U_γ^1 since EDF-Zeno did this much work on these jobs. This is Part 1 on the LHS, excluding the first term where $\gamma = \beta$. Now, consider jobs with criticality level β. Some of these jobs may arrived in mode 1 (if they arrived before k_β) and some in mode β. Therefore, the work of these jobs is at least $U_\beta^1 + U_\beta^\beta$. In addition, if $\beta = \alpha$ or J_{miss} releases in runtime mode 1, then the total work of these all these jobs also includes δ since J_{miss} is part of this set and it has δ work undone at the deadline. This is Part 2 of the LHS (the second term has an If, since we over count when $\beta = 1$). Summing them all up in the LHS gives us a lower bound on the total work of jobs that must be scheduled by OPT in criticality level β. Since RHS is an upper bound, we get the inequality. ◀

▶ **Lemma 22.** *For* $1 \le \beta < \alpha$,

$$\underbrace{\sum_{\gamma=1}^{\beta} U_\gamma^1}_{Part\ 1} + \underbrace{\sum_{\gamma=2}^{\beta} U_\gamma^\gamma + \frac{2^\beta - 1}{2^{\beta-1}} \sum_{\gamma=\beta+1}^{\alpha} U_\gamma^\gamma}_{Part\ 2} \le \frac{2^\beta - 1}{2^{\beta-1}} d - \sum_{\gamma=1}^{\beta} \frac{1}{2^{\gamma-1}} b_\gamma.$$

Proof. Consider EDF-Zeno for jobs with criticality $\gamma \le \beta$ in the various sets $B_1, B_2, \ldots$. Each of these jobs arrives either in runtime criticality 1 or γ. Therefore, the total utility (execution given to these jobs collectively by EDF-Zeno) is $\sum_{\gamma=1}^{\beta} U_\gamma^1 + \sum_{\gamma=2}^{\beta} U_\gamma^\gamma$. All these jobs will be dropped at $k_{\beta+1}$. Therefore, the total work EDF-Zeno does on these jobs is no more than the possible execution time on the various lanes. The total available execution time is: $(k_{\beta+1} - b_1)$ on lane 1;$0.5 \times (k_{\beta+1} - b_2)$ on lane 2, and so on. (Note that due to Lemmas 17, 18, we can assume that $b_\beta \le k_{\beta+1}$, so these intervals always exist). This means $\sum_{\gamma=1}^{\beta} U_\gamma^1 + \sum_{\gamma=2}^{\beta} U_\gamma^\gamma \le \sum_{\gamma=1}^{\beta} \frac{1}{2^{\gamma-1}}(k_{\beta+1} - b_\gamma)$.

On the other hand, recall that the system is in criticality Φ at time d. For $\beta < \gamma < \Phi$, consider U_γ^γ. This is the work done by EDF-Zeno on jobs released with static criticality γ and runtime mode γ. Therefore, these jobs must be in set B_γ and must be released after k_γ.

Due to the flow control policy of EDF-Zeno (Definition 4), we know that $U_\gamma^\gamma \le k_{\gamma+1} - k_\gamma$ for $\beta < \gamma < \Phi$, and $U_\Phi^\Phi \le d - k_\Phi$. Also, for $\Phi < \gamma \le \alpha$, $U_\gamma^\gamma = 0$.

Multiplying the second inequality by $\frac{2^\beta - 1}{2^{\beta-1}}$, and adding them to the first inequality gives us the lemma. ◀

Finally, using Lemmas 21 and 22, we can complete the proof for Theorem 10.

Table 2 The coefficients of each inequality when adding them up.

Weight	LHS(Part 1)	LHS(Part 2)	RHS
A_1	$U_1^1 + U_2^1 + \ldots + U_m^1$		$d - b_1$
A_2	$U_2^1 + \ldots + U_m^1$	U_2^2	$d - b_2$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$A_{\alpha-1}$	$U_{\alpha-1}^1 + U_\alpha^1 + \ldots + U_m^1$	$U_{\alpha-1}^{\alpha-1}$	$d - b_{\alpha-1}$
A_α	$U_\alpha^1 + \ldots + U_m^1$	U_α^α	$d - b_\alpha$
A_2	U_1^1	$U_2^2 + U_3^3 + U_4^4 \ldots + U_\alpha^\alpha$	$d - b_1$
A_3	$U_1^1 + U_2^1$	$U_2^2 + \frac{3}{2}(U_3^3 + U_4^4 + \ldots + U_\alpha^\alpha)$	$d - b_1 - \frac{1}{2}b_2$
A_4	$U_1^1 + U_2^1 + U_3^1$	$U_2^2 + U_3^3 + \frac{7}{4}(U_4^4 + \ldots + U_\alpha^\alpha)$	$d - \sum_{\beta=1}^{3} \frac{1}{2^{\beta-1}} b_\beta$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
A_α	$U_1^1 + U_2^1 + \ldots + U_{\alpha-1}^1$	$U_2^2 + \ldots + U_{\alpha-1}^{\alpha-1} + \frac{2^{\alpha-1}-1}{2^{\alpha-2}} U_\alpha^\alpha$	$d - \sum_{\beta=1}^{\alpha} \frac{1}{2^{\beta-1}} b_\beta$

Proof. We first define some *weights* that we will multiply to these inequalities in order to get appropriate coefficients. In particular, we define $A_\alpha = \frac{1}{2^{\alpha-1}}$ and for all $1 \le \beta < \alpha$, $A_\beta = \frac{2^\beta - 1}{2^{\beta-1}} A_{\beta+1}$. As an illustration, if J_{miss}'s static criticality is 4, then $A_4 = 1/8$, $A_3 = 7/32$, $A_2 = 21/64$, and $A_1 = 21/64$.

Considering Lemmas 21 and 22, we have a total of $\alpha + (\alpha - 1) = 2\alpha - 1$ inequalities. For clarity, we put these in Table 2 where the first α rows are for Lemma 21 and the subsequent $\alpha - 1$ rows are for Lemma 22. We also divide the LHS into two parts as marked in the corresponding lemmas: Part 1 corresponds to the work included in U_x^1 and Part 2 corresponds to the work included in U_x^x. We multiply each row by weights as shown in the table and add them together.

These coefficients have the following properties (which can be proved through induction):

$$\sum_{\gamma=1}^{\beta} A_\gamma = 2^{\beta-1} A_\beta, \quad A_\beta + \frac{1}{2^{\beta-1}} A_{\beta+1} = \frac{1}{2^{\beta-1}}$$

Using these properties, we first show that the coefficients of each term that appears on the LHS is 1 once add up all the inequalities. For Part 1, check U_β^1 for $1 \le \beta \le \alpha$: it appears in the first β rows and last $\alpha - \beta$ rows. (For instance, U_1^1 appears in the first row and the last $\alpha - 1$ rows. Therefore, the coefficients of each U_β^1 is $\sum_{\beta=1}^{\alpha} A_\beta = 2^{\alpha-1} A_\alpha = 1$. For $\beta > \alpha$, these U_β^1s have the same coefficient as U_α^1, so the total is still 1.

Part 2 is a bit more involved. We use induction to show that after adding all the rows, the coefficient of each term is 1. The base case is $\beta = 2$ (the coefficient of U_2^2 is $A_2 + \sum_{\beta=2}^{\alpha} A_\beta = 1$). We now show that the difference between the coefficients of U_β^β and $U_{\beta+1}^{\beta+1}$ is 0.

1. Row β only has U_β^β, with weight A_β;
2. Row $\beta + 1$ only has $U_{\beta+1}^{\beta+1}$, with weight $A_{\beta+1}$;
3. Row $\alpha + \beta$ has $U_\beta^\beta + \frac{2^\beta - 1}{2^{\beta-1}} U_{\beta+1}^{\beta+1}$, with weight $A_{\beta+1}$.
4. In other rows, the coefficient of U_β^β and $U_{\beta+1}^{\beta+1}$ are the same.

Therefore, the difference between the coefficient of U_β^β and $U_{\beta+1}^{\beta+1}$ is:

$$A_\beta - A_{\beta+1} + A_{\beta+1}(1 - \frac{2^\beta - 1}{2^{\beta-1}}) = A_\beta - \frac{2^\beta - 1}{2^{\beta-1}} A_{\beta+1} = 0.$$

Now consider δ (which was not included in the table). If $\alpha > \Phi$, then job J_{miss} appears in runtime mode 1; therefore, its coefficient is 1; otherwise, it is released in runtime mode Φ and its coefficient is $1/2^{\alpha-1}$. Therefore, in the worst case, it has a coefficient of $1/2^{\alpha-1}$.

For RHS, check b_β for $1 \leq \beta \leq \alpha$. b_β appears at the β-th row and the last $\beta - 1$ rows (with coefficient $\frac{1}{2^{\beta-1}}$). The total coefficient is:

$$-A_\beta - \frac{1}{2^{\beta-1}} \sum_{\gamma=\beta+1}^{\alpha} A_\gamma = -A_\beta - \frac{1}{2^{\beta-1}}(1 - 2^{\beta-1}A_\beta) = -\frac{1}{2^{\beta-1}}.$$

Finally, check d, the coefficient is

$$\sum_{\beta=1}^{\alpha} A_\beta + \sum_{\beta=2}^{\alpha} A_\beta = 2 - A_1 = \frac{2^\alpha - 1}{2^{\alpha-1}}$$

Collecting them altogether, the sum of these inequalities results in:

$$\sum_{\beta=1}^{\alpha} U_\beta^1 + \sum_{\beta=2}^{\alpha} U_\beta^\beta + \frac{1}{2^{\alpha-1}}\delta \leq \frac{2^\alpha - 1}{2^{\alpha-1}} d - \sum_{\beta=1}^{\alpha} \frac{1}{2^{\beta-1}} b_\beta.$$

Since J_{miss} misses its deadline, $\delta > 0$, so we have

$$\sum_{\beta=1}^{\alpha} U_\beta^1 + \sum_{\beta=2}^{\alpha} U_\beta^\beta < \frac{2^\alpha - 1}{2^{\alpha-1}} d - \sum_{\beta=1}^{\alpha} \frac{1}{2^{\beta-1}} b_\beta.$$

The LHS is all the work of B_1 that was done by EDF-Zeno within the busy period and the RHS is the total execution capacity of the busy period. Therefore, by definition of the busy period, the work done cannot be smaller than the capacity of the processor. Hence a contradiction. ◀

We have now shown that EDF-Zeno provides the optimal speedup bound. However, it is important to note that it is not an optimal scheduler for individual instances. For instance, even if a particular set of jobs is semi-clairvoyantly schedulable on speed 1.1 processors, EDF-Zeno will still (potentially) need speed $s = (2^m - 1)/2^{m-1}$ speed processor to schedule it. Alternatively, one can say that, on a speed 1 processor, EDF-Zeno only guarantees schedulability to job sets which are feasible on speed $1/s$ processor. In the next section, we provide a scheduler which is instance optimal.

5 Scheduling Multi-Criticality Semi-Clairvoyant Jobs Optimally

This section provides a linear programming formulation for checking the schedulability for a system of n semi-clairvoyant jobs with m criticality levels. Solving the linear program also provides a scheduling algorithm that can then be implemented at runtime as the jobs arrive. This scheduler provides a quantitatively different guarantee relative to EDF-Zeno. Given a particular set of jobs, if the linear program fails to provide a schedule, then no semi-clairvoyant scheduler can schedule this set of jobs under all possible runtime conditions (sequence of mode switches).

5.1 Semi-Clairvoyant Scheduling using LP

Basic Ideas and Notation

We will use a linear program to check if the system of such jobs is schedulable. Since we have n jobs, we have $2n$ ***key instances*** on the timeline which represent the arrival times and deadlines of these jobs[5], dividing the timeline into $2n - 1$ intervals. We will sort these key

[5] For simplicity in exposition and without loss of generality, we assume that all these instances are unique.

instances and represent them as t_i. For each of these intervals, we will compute how much execution should be done by jobs with each static criticality given the history of criticality switches so far. In particular, the linear program uses the following notation:

- The variables for the linear program are $c^x_{k_\alpha,j}(k_1, \ldots k_\alpha)$: Amount of x criticality work done from k_α to t_j. k_y is the time instant system criticality increased to y (Note that $k_1 = 0$). The parameters $k_1, k_2, \ldots, k_\alpha$ can be a set of any size between 1 and m and must all take values smaller than t_j since a mode switch after time t_j cannot impact the execution time given to any criticality level up to time t_j.[6] These are the quantities the linear program will compute.
- Key instances t_i: obtained by sorting the release time and deadline of all the jobs. Assume that $t_0 = 0$.
- $S^x_{i,j}$: the set of jobs with static criticality x with arrival time $\geq t_i$ and deadline $\leq t_j$. These are the jobs with static criticality x that must be executed between time t_i and t_j
- $\sum_{J \in S^x_{i,j}} e^y(J)$: sum of y criticality work over $S^x_{i,j}$.

Note that the variables $c^x_{k_\alpha,j}(k_1, \ldots k_\alpha)$ compute the entire schedule for semi-clairvoyant scheduler under all possible mode-switch conditions and prefixes. There are approximately mn^{m+2} of these variables since i, j, k_d can take n values each and x takes m values. One can store these in a lookup table to be used for runtime scheduling.

LP Constraints

The basic idea behind the linear program is simple. We will consider all pairs i, j such that $0 \leq i < j$ and consider the interval between t_i and t_j. For all possible mode switch conditions, we want to ensure that we can allocate enough time to work for jobs of each static criticality level. In other words, we want to ensure that for all jobs that arrive at or after time t_i and have a deadline at or before t_j can get "sufficient execution time" to satisfy the correctness condition. Note that not all of these jobs may meet their deadlines since the scheduler may abandon some of these jobs due to mode switches. These variables c represent the reserved time for each static criticality level within each interval.

Consider an instant k_α when some job with static criticality α might arrive. We will consider two cases.

Case 1. We first consider the simple case where $t_i \geq k_\alpha$. In this case, Inequality 1 represents the fact that all jobs must be able to meet their deadlines – it simply checks that we reserve enough time for each criticality level to complete the work that must be done within that interval. Inequality 2 represents the fact that we cannot execute more work than has arrived so far for any static criticality level. Finally, Inequality 3 represents the fact that the total execution within any interval cannot exceed the total time in that interval.

for $x \in \{\alpha, \ldots, m\}$:

$$c^x_{k_\alpha,j}(k_1, \ldots k_\alpha) - c^x_{k_\alpha,i}(k_1, \ldots k_\alpha) \geq \sum_{J \in S^x_{i,j}} e^\alpha(J) \tag{1}$$

[6] For example, $c^3_{1,j}(0)$ denotes the execution time allocated to the jobs with static criticality 3 up to time t_j. As another example, $c^9_{4,10}(0, t_4, t_4)$ denotes the amount of time spent on jobs with static criticality 9 between time t_4 and t_{10} assuming the system criticality increased to 3 (and implicitly to 2) at time t_4.

$$c^x_{k_\alpha,j}(k_1,\dots k_\alpha)+\sum_{y=1}^{\alpha-1} c^x_{k_y,k_{y+1}}(k_1,\dots,k_y) \le \sum_{J\in S^x_{k_\alpha,2n-1}} e^\alpha(J) - \sum_{J\in S^x_{j,2n-1}} e^\alpha(J)$$
$$+\sum_{y=1}^{\alpha-1}\Big(\sum_{J\in S^x_{k_y,2n-1}} e^y(J) - \sum_{J\in S^x_{k_{y+1},2n-1}} e^y(J)\Big) \tag{2}$$

$$\sum_{y=\alpha}^{m}(c^y_{k_\alpha,j}(k_1,\dots k_\alpha) - c^y_{k_\alpha,i}(k_1,\dots k_\alpha)) \le t_j - t_i \tag{3}$$

Case 2. Now consider the case where $t_i < k_\alpha$. This is the crucial inequality since this is the only one that is worried about changes in the system criticalities. Say t_i is between 2 transition time $k_{\beta-1}$ and k_β. Now, we must ensure that all deadlines between k_α and t_j are met. Inequality 4 checks this. The first term on the LHS represents the amount of time allocated to the jobs with static criticality x between time k_α and t_j given the history of criticality switches up to k_α. The term $c^x_{k_{\beta-1},k_\beta}(k_1,\dots,k_{\beta-1}) - c^x_{k_{\beta-1},i}(k_1,\dots,k_{\beta-1})$ represents the amount of time allocated to the jobs with static criticality x between t_i and k_β and the last term $\sum_{y=\beta+1}^{\alpha} c^x_{k_{y-1},k_y}(k_1,\dots,k_{y-1})$ represents the amount of time allocated to the jobs with static criticality x between k_β and k_α. Therefore, collectively, the LHS represents the amount of time allocated to the jobs with static criticality x between t_i and t_j. The RHS represents the total amount of work that must be done for these jobs. The first term, $\sum_{J\in S^x_{k_\alpha,j}} e^\alpha(J)$ represents the jobs that arrive between time k_α and t_j – these jobs arrive with $\rho=\alpha$; so we consider $e^\alpha(J)$ for these jobs. The second term, $(\sum_{J\in S^x_{i,j}} e^{\beta-1}(J) - \sum_{J\in S^x_{k_\beta,j}} e^{\beta-1}(J))$ represents the jobs that arrived between time t_i and k_β – these jobs arrive at $\rho=\beta-1$. The last term $\sum_{y=\beta+1}^{\alpha}(\sum_{J\in S^x_{k_{y-1},j}} e^{y-1}(J) - \sum_{J\in S^x_{k_y,j}} e^{y-1}(J))$ represents the jobs that arrive between time k_β and k_α.

$$c^x_{k_\alpha,j}(k_1,\dots k_\alpha) + c^x_{k_{\beta-1},k_\beta}(k_1,\dots,k_{\beta-1}) - c^x_{k_{\beta-1},i}(k_1,\dots,k_{\beta-1})$$
$$+\sum_{y=\beta+1}^{\alpha} c^x_{k_{y-1},k_y}(k_1,\dots,k_{y-1}) \ge \sum_{J\in S^x_{k_\alpha,j}} e^\alpha(J) + \Big(\sum_{J\in S^x_{i,j}} e^{\beta-1}(J) - \sum_{J\in S^x_{k_\beta,j}} e^{\beta-1}(J)\Big)$$
$$+\sum_{y=\beta+1}^{\alpha}\Big(\sum_{J\in S^x_{k_{y-1},j}} e^{y-1}(J) - \sum_{J\in S^x_{k_y,j}} e^{y-1}(J)\Big) \text{ for } x \text{ in } \alpha,\dots,m \tag{4}$$

The final inequality just checks that these variable values are well-formed since we cannot do more work until time t_j than we do until time t_{j+1}.

$$c^y_{k_\alpha,j}(k_1,\dots k_\alpha) \le c^y_{k_\alpha,j+1}(k_1,\dots k_\alpha). \tag{5}$$

Without worrying about the actual asymptotic, it should be clear that the number of constraints is polynomial in n and exponential in m.

Runtime scheduling

Let $\widehat{c}^{x_1}_{k_1,i}(k_1), \widehat{c}^{x_2}_{k_2,i}(k_1,k_2),\dots,\widehat{c}^{x_d}_{k_d,i}(k_1,\dots,k_d)$ be the solution of our LP. In particular, the set of solution looks like this: $\{\widehat{c}^1_{k_1,i}(k_1), \widehat{c}^2_{k_1,i}(k_1),\dots,\widehat{c}^d_{k_1,i}(k_1), \widehat{c}^2_{k_2,i}(k_1,k_2),\dots \widehat{c}^d_{k_2,i}(k_1,k_2),\dots,$ $\widehat{c}^{d-1}_{k_{d-1},i}(k_1,\dots,k_{d-1}), \widehat{c}^d_{k_{d-1,i}}(k_1,\dots,k_{d-1}), \widehat{c}^d_{k_d,i}(k_1,\dots,k_d)\}$.

Consider an arbitrary criticality x. During runtime, let say that the system at time t_i is currently in α criticality mode. Then the mode switch time $k_1, k_2, \ldots, k_\alpha$ is known. We ***reserve*** $\widehat{c}^x_{k_\alpha,i+1}(k_1,\ldots,k_\alpha) - \widehat{c}^x_{k_\alpha,i}(k_1,\ldots,k_\alpha)$ time for x criticality work in the interval $[i, i+1)$(Jobs with the same criticality level are scheduled by EDF).

5.2 Proof of Correctness

We now prove that this linear program is correct. That is, a system of jobs is semi-clairvoyantly schedulable iff the the linear program is feasible.

Direction 1: LP solution can be used to schedule jobs

We first consider the case when the linear program has a solution and computes the c values. We must argue that the computed values of c are enough to schedule all jobs correctly under any sequence of mode switches.

We first make an observation – without loss of generality, we will assume that the actual execution time γ of the job is always maximum for a given criticality level since that is the worst case for any semi-clairvoyant scheduler. We can formally state this as follows:

▶ **Observation 23.** *When a job J arrives with runtime ρ, then it always have requires $\gamma(J)$ execution time. We assume this since this is the worst case for semi-clairvoyant schedulers.*

We now show that if linear program has a solution, then every reserved interval is busy.

▶ **Lemma 24.** *Say the linear program computes a solution and that some mode switches have occurred at times $k_1, k_2, \ldots, k_\alpha$. Now consider any time t_j. $c^x_{k_\alpha,j+1}(k_1,\ldots k_\alpha) - c^x_{k_\alpha,j}(k_1,\ldots k_\alpha)$ is exactly the amount of x criticality work done in $[t_j, t_{j+1})$ assuming Observation 23 holds.*

Proof. Consider an arbitrary criticality level x and any combination of mode switch k_1, k_2, $\ldots, k_\alpha$ that occur at or before time t_j.

We want to use induction to show that $c^x_{k_\alpha,j+1}(k_1,\ldots k_\alpha) - c^x_{k_\alpha,j}(k_1,\ldots,k_\alpha)$ is exactly the amount of x criticality work done in $[j, j+1)$.

Base case. $j = 0$, this is true since both reserved time and work available are 0.

Induction step. Consider any $j' < j$ and say k_β is the time of the last mode switch at or before time $t_{j'}$. We assume that, $c^x_{k_\beta,j'+1}(k_1,\ldots k_\beta)$ - $c^x_{k_\alpha,j'}(k_1,\ldots,k_\alpha)$is the amount of x work executed in $[t_{j'}, t_{j'+1})$. We want to show that $c^x_{k_\alpha,j+1}(k_1,\ldots,k_\alpha) - c^x_{k_\alpha,j}(k_1,\ldots,k_\alpha)$ is exactly the amount of x criticality work done in $[t_j, t_{j+1})$.

Since the maximum amount of work arrives in every interval (Observation 23), the right hand side of (2) represents the total work x criticality work that must be done before time t_{j+1} under these mode switches. Subtracting $c^x_{k_\alpha,j}(k_1,\ldots,k_\alpha) - \sum_{y=1}^{\alpha-1} c^x_{k_y,k_{y+1}}(k_1,\ldots,k_y)$ on both sides of (2), we get:

$$c^x_{k_\alpha,j+1}(k_1,\ldots,k_\alpha) - c^x_{k_\alpha,j}(k_1,\ldots,k_\alpha) \leq \sum_{J\in S^x_{k_\alpha,2n-1}} e^\alpha(J) - \sum_{J\in S^x_{j+1,2n-1}} e^\alpha(J) +$$

$$\sum_{y=1}^{\alpha-1}\Big(\sum_{J\in S^x_{k_y,2n-1}} e^y(J) - \sum_{J\in S^x_{k_{y+1},2n-1}} e^y(J)\Big) - c^x_{k_\alpha,j}(k_1,\ldots,k_\alpha) - \sum_{y=1}^{\alpha-1} c^x_{k_y,k_{y+1}}(k_1,\ldots,k_y) \qquad (6)$$

By induction hypothesis on every small interval between $[0, k_\alpha)$, we know that: $\sum_{y=1}^{\alpha-1} c^x_{k_y,k_{y+1}}(k_1, \ldots, k_y)$ is exactly the amount of x work completed in $[0, k_\alpha)$. Similarly, by induction hypothesis, if we consider any interval between $[k_\alpha, t_j)$, $c^x_{k_\alpha,j}(k_1, \ldots, k_\alpha)$ is the amount of work with static criticality x completed in $[k_\alpha, j)$.

Therefore, the right hand side of (6) represents the total leftover work with static criticality x that is must be done within the interval $[t_j, t_{j+1})$ under these mode switch conditions. So $c^x_{k_\alpha,j+1}(k_1, \ldots, k_\alpha) - c^x_{k_\alpha,j}(k_1, \ldots, k_\alpha)$ must be the exact amount of x criticality work done within this interval. ◀

▶ **Corollary 25.** *The system is never idle – some work is executed during any reserved time.*

We can now show that the LP schedule guarantees that all deadlines are met.

▶ **Lemma 26.** *If the LP has a solution, then the semi-clairvoyant scheduler correctly schedules jobs under all mode switch conditions.*

Proof. As in Section 4, we assume, for contradiction, that job J_{miss} with static criticality level $\chi(J_{\text{miss}}) = x$ arrived at t_a and misses its deadline at t_d. In other word, we did not reserve enough time for x criticality work. Now, consider only jobs with criticality x. Let $[t_b, t_d)$ be the busy period for criticality x, i.e., t_b is the latest time instant before $t_a (t_b \leq t_a)$ such that in the interval $[t_b, t_d)$, only jobs that arrive after t_b and have deadline before t_d are executed within the time reserved for x-criticality work.

From (4) or (1) (if there is at least 1 mode switch between t_b and t_d then use (4), else use (1)), substitute $i = b$ and $j = d$.

In both cases, the LHS denotes the x criticality work that was executed in the interval $[t_b, t_d)$. However, the RHS is exactly the jobs with criticality x, arrival time $\geq t_b$ and deadline $\leq t_d$. In addition, note that if the maximum amount of work arrives in every interval, from Corollary 25, none of the reserved time is wasted on idleness. Therefore, this entire interval was used. However we still missed J_{miss} deadline at t_d, which is a contradiction. ◀

Direction 2: If LP fails, then the system of jobs is not semi-clairvoyantly schedulable

Now we argue that the linear program is optimal. In other words, if the linear program does not find a feasible solution, then no online semi-clairvoyant scheduler can schedule this system of jobs under all mode change conditions.

Now consider an optimal online scheduler OPT^α for scheduling a set of jobs. We will now construct $S = \{\widehat{c}^x_{k_\alpha,i}(k_1, k_2, \ldots, k_\alpha)\}$ for all i and possible mode switch times $k_1, k_2, \ldots, k_\alpha$ before time t_i such that $\widehat{c}^x_{k_\alpha,i}(k_1, k_2, \ldots, k_\alpha)$ represents the amount of time OPT^α spends on x-criticality work OPT^α schedules between time k_α and t_i.

▷ Claim 27. With OPT^α, we can construct this set S.

Proof. Since the optimal scheduler OPT^α exists, we can construct S by simulating it under all mode switch conditions. Consider an arbitrary criticality x. If no mode switch happens (system always stay at 1 criticality), we can get $\widehat{c}^x_{k_1,i}(k_1)$ for any i by calculating the amount of x criticality work executed in the interval $[0, i)$. If instead, the system switch to 2 criticality at some time $k_2 = i'$, we can still get $\widehat{c}^x_{k_1,i}(k_1)$ is still the same for any $i \leq i'$ since OPT^α is an online scheduler and cannot look into the future to know that the mode switch will occur at time $t_{i'}$. Also, we can get $\widehat{c}^x_{k_2,i}(k_1, k_2)$ for any $t_i \geq k_2$ by calculating the amount of x criticality work executed in the interval $[k_2, i)$. In this manner, we can run the scheduler for all mode switch possibilities and calculate a consistent S. ◁

It is straightforward to see that the set S could be used to in our runtime schedule. Therefore, we get the following claim.

▷ Claim 28. If there is no solution to the linear program, then set S from OPT^{α} must also not satisfy the LP.

We can now argue that this is impossible if OPT^{α} is a valid scheduler.

▶ **Lemma 29.** *If set S does not satisfy the linear program, then OPT^{α} is not a valid semi-clairvoyant scheduler.*

Proof. We can look at each constraint and check what happens if S does not satisfy the constraint. If S does not satisfies an inequality in (4), then there exists j', l', α', x' such that:

$$\widehat{c}^{x'}_{k_{\alpha'},j'}(k_1,\ldots,k_{\alpha'}) + \widehat{c}^{x'}_{k_{\beta-1},k_\beta}(k_1,\ldots,k_{\beta-1}) - \widehat{c}^{x'}_{k_{\beta-1},l'}(k_1,\ldots,k_{\beta-1}) + \sum_{y=\beta+1}^{\alpha'} \widehat{c}^{x'}_{k_{y-1},k_y}(k_1,\ldots,k_{y-1}) < \sum_{J\in S^{x'}_{k_{\alpha'},j'}} e^{\alpha'}(J) + \sum_{y=\beta+1}^{\alpha'} \Big(\sum_{J\in S^{x'}_{k_{y-1},j'}} e^{y-1}(J) - \sum_{J\in S^{x'}_{k_y,j'}} e^{y-1}(J) \Big) + \sum_{J\in S^{x'}_{l',j'}} e^{\beta-1}(J) - \sum_{J\in S^{x'}_{k_\beta,j'}} e^{\beta-1}(J)$$

On the LHS, $\widehat{c}^{x'}_{k_{\beta-1},k_\beta}(k_1,\ldots,k_{\beta-1}) - \widehat{c}^{x'}_{k_{\beta-1},l'}(k_1,\ldots,k_{\beta-1})$ is the amount of x' work done by OPT^{α} in $[l', k_\beta)$, $\sum_{y=\beta+1}^{\alpha'} \widehat{c}^{x'}_{k_{y-1},k_y}(k_1,\ldots,k_{y-1})$ is the amount of x' work done by OPT^{α} in $[k_\beta, k_{\alpha'})$ and $\widehat{c}^{x'}_{k_{\alpha'},t_{j'}}(k_1,\ldots,k_{\alpha'})$ is the amount of x' work done by OPT^{α} in $[k_{\alpha'}, j')$. So, the **LHS is the amount of x' work done by OPT^{α} in $[t_{l'}, t_{j'})$**. On the RHS, we have: $\sum_{J\in S^{x'}_{l',j'}} e^{\beta-1}(J) - \sum_{J\in S^{x'}_{k_\beta,j'}} e^{\beta-1}(J)$ is the amount of x' work arrives in $[t_{l'}, k_\beta)$ that has deadline $t_{j'}$, $\sum_{y=\beta+1}^{\alpha'}(\sum_{J\in S^{x'}_{y-1,j'}} e^{y-1}(J) - \sum_{J\in S^{x'}_{y,j'}} e^{y-1}(J))$ is the amount of x' work arrives in $[k_\beta, k_{\alpha'})$ that has deadline $t_{j'}$, and $\sum_{J\in S^{x'}_{k_{\alpha'},j'}} e^{\alpha'}(J)$ is the amount of x' work arrives in $[k_{\alpha'}, t_{j'})$ that has deadline $t_{j'}$. So the **RHS is the amount of x' work arrives after $t_{l'}$ and has deadline $t_{j'}$**. If $LHS < RHS$, then OPT^{α} must miss the deadline of some x' criticality job that has deadline $t_{j'}$. Since OPT^{α} can schedule the system, OPT^{α} must satisfies (4).

If S does not satisfy an inequality in (1), then there exists j', l', α', x' such that:

$$\widehat{c}^{x'}_{k_{\alpha'},j'}(k_1,\ldots,k_{\alpha'}) - \widehat{c}^{x'}_{k_{\alpha'},l'}(k_1,\ldots,k_{\alpha'}) < \sum_{J\in S^{x'}_{l',j'}} e^{\alpha'}(J)$$

$\widehat{c}^{x'}_{k_{\alpha'},j'}(k_1,\ldots,k_{\alpha'}) - \widehat{c}^{x'}_{k_{\alpha'},l'}(k_1,\ldots,k_{\alpha'})$ is the amount of x' work done by OPT^{α} in $[l', j')$ while $\sum_{J\in S^{x'}_{l',j'}} e^{\alpha'}(J)$ is the amount of x' work arrives after $t_{l'}$ and has deadline $t_{j'}$. If $LHS < RHS$, then OPT^{α} must misses the deadline of some x' criticality job that has deadline $t_{j'}$. Since OPT^{α} can schedule the system, OPT^{α} must satisfies (1).

If S does not satisfies an inequality in (2), then there exists j', α', x' such that:

$$\widehat{c}^{x'}_{k_{\alpha'},j'}(k_1,\ldots,k_{\alpha'}) + \sum_{y=1}^{\alpha'-1} \widehat{c}^{x'}_{k_y,k_{y+1}}(k_1,\ldots,k_y) > \sum_{J\in S^{x'}_{k_{\alpha'},2n-1}} e^{\alpha'}(J) - \sum_{J\in S^{x'}_{j',2n-1}} e^{\alpha'}(J) + \sum_{y=1}^{\alpha'-1} \Big(\sum_{J\in S^{x'}_{k_y,2n-1}} e^{y}(J) - \sum_{J\in S^{x'}_{k_{y+1},2n-1}} e^{y}(J) \Big)$$

On the LHS, $\sum_{y=1}^{\alpha'-1} \widehat{c}^{x'}_{k_y,k_{y+1}}(k_1,\ldots,k_y)$ is the amount of x' work done by OPT^α in $[0,k_{\alpha'})$ and $\widehat{c}^{x'}_{k_{\alpha'},j'}(k_1,\ldots,k_{\alpha'})$ is the amount of x' work done by OPT^α in $[k_{\alpha'},j')$. So, the **LHS is the amount of x' work done by OPT^α in $[0,j')$**. On the RHS, $\sum_{y=1}^{\alpha'-1}(\sum_{J\in S^{x'}_{k_y,2n-1}} e^y(J) - \sum_{J\in S^{x'}_{k_{y+1},2n-1}} e^y(J))$ is the amount of x' work arrives in $[0,k_{\alpha'})$ and $\sum_{J\in S^{x'}_{k_{\alpha'},2n-1}} e^{\alpha'}(J) - \sum_{J\in S^{x'}_{j',2n-1}} e^{\alpha'}(J)$ is the amount of x' work arrives in $[k_{\alpha'},j')$. So, the **RHS is the amount of x' work arrives before $t_{j'}$**. By definition, LHS cannot be larger than RHS. Thus, OPT^α must satisfies (2).

If S does not satisfies an inequality in (3), then there exists i',j',α' such that:

$$\sum_{y=\alpha'}^{d} (\widehat{c}^{y}_{k_{\alpha'},j'}(k_1,\ldots,k_{\alpha'}) - \widehat{c}^{y}_{k_{\alpha'},i'}(k_1,\ldots,k_{\alpha'})) > t_{j'} - t_{i'}$$

LHS is the total amount of work done by OPT^α in $[t_{i'},t_{j'})$ while $t_{j'}-t_{i'}$ is the amount of time available between $t_{i'}$ and $t_{j'}$. Since OPT^α can schedule this system, LHS cannot greater than RHS. Thus, OPT^α must satisfies (3).

Therefore, OPT^α must satisfies all constrains of LP, which is a contradiction. Thus, if there is no solution to the LP, the system is unschedulable. ◀

6 Related Work

Since it was first considered, mixed criticality model has been studied extensively [12, 20, 21, 13, 24, 19, 18, 9, 14, 15, 17] (see [10] for a survey). While most of this work is on recurrent tasks, there has also been significant work on jobs.

Most prior work considers the non-clairvoyant model where the system discovers the runtime mode of a job as it executes. Baruah et al. [6] developed Own Criticality Based Priority (OCBP) which provides the optimal speedup bound for jobs in the non-clairvoyant setting. Several runtime mechanisms have also been explored empirically. Baruah and Burns [3] demonstrated that it is practical to implement a MCS with run-time monitoring, which led to the formal introduction of a theoretical framework [4]. Static Mixed Criticality (SMC) and Adaptive Mixed Criticality (AMC) [4] are the continuation of OCBP by allowing run-time monitoring. Most work on mixed criticality systems considers the case where the system mode monotonically increases – as jobs arrive with higher runtime mode, the system criticality increases and lower criticality jobs are discarded, but some systems [22, 7, 8] allow systems to switch back to lower criticality mode.

In 2019, Agrawal et al. [2] introduced the notion of semi-clairvoyance that is used in this paper. Burns and Davis [11] provided an analysis of AMC algorithm in dual-criticality system with semi-clairvoyant behavior. Zhao et al [25] examine the integration of AMC, Preemption Threshold Scheduling(PTS) and semi-clairvoyance. Baruah and Ekberg [5] study graceful degradation (ensure less criticality jobs can be dropped safely during mode switch) for semi-clairvoyant systems. Jiang et al. [16] proposed the notion of quarter-clairvoyance, where execution time is revealed not when a job is released but when it is executed for a certain amount of time.

7 Conclusions

In this paper, we provide tight upper and lower bounds of $(2^m-1)/2^{m-1}$ on the speedup required to schedule feasible mixed criticality jobs in the semi-clairvoyant model. In addition, we provide a linear program which can (instance)-optimally schedule a system of semi-clairvoyant jobs with arbitrary criticality levels. While the running time of the LP is

exponential in the number of criticalities, this seems unavoidable and is also potentially not too onerous given that the number of criticalities is typically small. We also provide EDF-Zeno, which (while not instance optimal) provides the optimal speedup bound for semi-clairvoyant jobs. While we use it as an analysis tool, it can also be used as a scheduling algorithm if one does not want to solve the linear program.

This work demonstrates the significant advantage of semi-clairvoyance over non-clairvoyance since there are functionally no nontrivial speedup bounds for non-clairvoyant schedulers for more than 2 criticality levels and solving the scheduling problem is NP-complete even for two criticality levels. This indicates that getting the information about mode switches early and at a predictable time is a significant benefit and any systems which can implement semi-clairvoyance or variants thereof should attempt to do so since it can lead to significant advantages in schedulability. In particular, there is a small marginal cost for adding additional criticality levels, which would allow system designers more flexibility in their design.

There are several open questions. First, could one either design an exact algorithm for semi-clairvoyance which runs in polynomial time in the number of jobs and criticalities or prove that it is impossible. In addition, speedup bounds for semi-clairvoyant tasks (especially non-implicit deadline tasks) has not been studied at all, to the best of our knowledge. Finally, how much can we relax semi-clairvoyance and still get optimal algorithms and good speedup bounds relative to full non-clairvoyance?

References

1 Kunal Agrawal and Sanjoy K. Baruah. Intractability issues in mixed-criticality scheduling. In Sebastian Altmeyer, editor, *30th Euromicro Conference on Real-Time Systems, ECRTS 2018, Barcelona, Spain, July 3-6, 2018*, volume 106 of *LIPIcs*, pages 11:1–11:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ECRTS.2018.11.

2 Kunal Agrawal, Sanjoy K. Baruah, and Alan Burns. Semi-clairvoyance in mixed-criticality scheduling. In *IEEE Real-Time Systems Symposium, RTSS 2019, Hong Kong, SAR, China, December 3-6, 2019*, pages 458–468. IEEE, IEEE, 2019. doi:10.1109/RTSS46320.2019.00047.

3 Sanjoy K. Baruah and Alan Burns. Implementing mixed criticality systems in ada. In Alexander B. Romanovsky and Tullio Vardanega, editors, *Reliable Software Technologies - Ada-Europe 2011 - 16th Ada-Europe International Conference on Reliable Software Technologies, Edinburgh, UK, June 20-24, 2011. Proceedings*, volume 6652 of *Lecture Notes in Computer Science*, pages 174–188. Springer, Springer, 2011. doi:10.1007/978-3-642-21338-0_13.

4 Sanjoy K. Baruah, Alan Burns, and Robert I. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*, pages 34–43. IEEE, IEEE Computer Society, 2011. doi:10.1109/RTSS.2011.12.

5 Sanjoy K. Baruah and Pontus Ekberg. Graceful degradation in semi-clairvoyant scheduling. In Björn B. Brandenburg, editor, *33rd Euromicro Conference on Real-Time Systems, ECRTS 2021, Virtual Conference, July 5-9, 2021*, volume 196 of *LIPIcs*, pages 9:1–9:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ECRTS.2021.9.

6 Sanjoy K. Baruah, Haohan Li, and Leen Stougie. Mixed-criticality scheduling: Improved resource-augmentation results. In Thomas Philips, editor, *Proceedings of the ISCA 25th International Conference on Computers and Their Applications, CATA 2010, March 24-26, 2010, Sheraton Waikiki Hotel, Honolulu, Hawaii, USA*, pages 217–223. ISCA, 2010.

7 Iain Bate, Alan Burns, and Robert I. Davis. A bailout protocol for mixed criticality systems. In *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*, pages 259–268. IEEE, IEEE Computer Society, 2015. doi:10.1109/ECRTS.2015.30.

8 Iain Bate, Alan Burns, and Robert I. Davis. An enhanced bailout protocol for mixed criticality embedded software. *IEEE Trans. Software Eng.*, 43(4):298–320, 2017. doi:10.1109/TSE.2016.2592907.

9 Alan Burns and Robert Davis. *Mixed Criticality Systems - A Review : (13th Edition, February 2022)*. White Rose Research Online, February 2022.

10 Alan Burns and Robert I. Davis. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6):82:1–82:37, 2018. doi:10.1145/3131347.

11 Alan Burns and Robert I. Davis. Schedulability analysis for adaptive mixed criticality systems with arbitrary deadlines and semi-clairvoyance. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*, pages 12–24. IEEE, IEEE, 2020. doi:10.1109/RTSS49844.2020.00013.

12 Lia Cagnizi, Federico Reghenzani, and William Fornaciari. Poster abstract: Run-time dynamic WCET estimation. In *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation, IoTDI 2023, San Antonio, TX, USA, May 9-12, 2023*, IoTDI '23, pages 458–460, New York, NY, USA, 2023. ACM. doi:10.1145/3576842.3589168.

13 Akanksha Chaudhari and Sanjoy K. Baruah. Efficient schedulability analysis of semi-clairvoyant sporadic task systems with graceful degradation. In Yasmina Abdeddaïm, Liliana Cucu-Grosjean, Geoffrey Nelissen, and Laurent Pautet, editors, *RTNS 2022: The 30th International Conference on Real-Time Networks and Systems, Paris, France, June 7 - 8, 2022*, RTNS '22, pages 116–126, New York, NY, USA, 2022. ACM. doi:10.1145/3534879.3534881.

14 Pontus Ekberg and Wang Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real Time Syst.*, 50(1):48–86, January 2014. doi:10.1007/s11241-013-9187-z.

15 Zhishan Guo, Luca Santinelli, and Kecheng Yang. EDF schedulability analysis on mixed-criticality systems with permitted failure probability. In *21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2015, Hong Kong, China, August 19-21, 2015*, pages 187–196. IEEE Computer Society, 2015. doi:10.1109/RTCSA.2015.8.

16 Zhe Jiang, Kecheng Yang, Nathan Fisher, Neil C. Audsley, and Zheng Dong. Pythia-mcs: Enabling quarter-clairvoyance in i/o-driven mixed-criticality systems. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*, pages 38–50. IEEE, IEEE, 2020. doi:10.1109/RTSS49844.2020.00015.

17 Jaewoo Lee, Kieu-My Phan, Xiaozhe Gu, Jiyeon Lee, Arvind Easwaran, Insik Shin, and Insup Lee. Mc-fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*, pages 41–52. IEEE Computer Society, 2014. doi:10.1109/RTSS.2014.32.

18 Ruixiao Li, Fahao Chen, and Peng Li. Semi-clairvoyant scheduling of speculative decoding requests to minimize LLM inference latency. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, Montreal, Canada, August 16-22, 2025*, IJCAI '25, pages 8554–8562. ijcai.org, 2025. doi:10.24963/ijcai.2025/951.

19 Behnaz Ranjbar, Tuan D. A. Nguyen, Alireza Ejlali, and Akash Kumar. Power-aware runtime scheduler for mixed-criticality systems on multicore platform. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 40(10):2009–2023, 2021. doi:10.1109/TCAD.2020.3033374.

20 Federico Reghenzani, Zhishan Guo, Luca Santinelli, and William Fornaciari. A mixed-criticality approach to fault tolerance: Integrating schedulability and failure requirements. In *28th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2022, Milano, Italy, May 4-6, 2022*, pages 27–39. IEEE, 2022. doi:10.1109/RTAS54340.2022.00011.

21 Amir Hassan Safizadeh, Sepideh Safari, Shayan Shokri, and Shaahin Hessabi. Energy-aware fault-tolerant mapping of mixed-criticality tasks on heterogeneous multicores. *IEEE Trans. Sustain. Comput.*, 10(4):756–767, 2025. doi:10.1109/TSUSC.2025.3532766.

22 François Santy, Gurulingesh Raravi, Geoffrey Nelissen, Vincent Nélis, Pratyush Kumar, Joël Goossens, and Eduardo Tovar. Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems. In Michel Auguin, Robert de Simone, Robert I. Davis, and Emmanuel Grolleau, editors, *21st International Conference on Real-Time Networks and Systems, RTNS 2013, Sophia Antipolis, France, October 17-18, 2013*, pages 183–192. ACM, 2013. doi:10.1145/2516821.2516834.

23 Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*, pages 239–243. IEEE, IEEE Computer Society, 2007. doi:10.1109/RTSS.2007.47.

24 Yiwen Zhang and Hui Zheng. Energy-aware reliability guarantee scheduling with semi-clairvoyant in mixed-criticality systems. *J. Syst. Archit.*, 156(C):103269, November 2024. doi:10.1016/j.sysarc.2024.103269.

25 Qingling Zhao, Mengfei Qu, Bo Huang, Zhe Jiang, and Haibo Zeng. Schedulability analysis and stack size minimization for adaptive mixed criticality scheduling with semi-clairvoyance and preemption thresholds. *J. Syst. Archit.*, 124:102383, 2022. doi:10.1016/j.sysarc.2021.102383.