MANAGING TARDINESS BOUNDS AND OVERLOAD IN SOFT REAL-TIME SYSTEMS

Jeremy P. Erickson

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2014

Approved by:

James H. Anderson

Sanjoy K. Baruah

Kevin Jeffay

F. Donelson Smith

Michael Reiter

Giuseppe Lipari

ABSTRACT

Jeremy P. Erickson: Managing Tardiness Bounds and Overload in Soft Real-Time Systems
(Under the direction of James H. Anderson)

In some systems, such as future generations of unmanned aerial vehicles (UAVs), different software running on the same machine will require different timing guarantees. For example, flight control software has hard real-time (HRT) requirements—if a job (i.e., invocation of a program) completes late, then safety may be compromised, so jobs must be guaranteed to complete within short deadlines. However, mission control software is likely to have soft real-time (SRT) requirements—if a job completes slightly late, the result is not likely to be catastrophic, but lateness should never be unbounded.

The global earliest-deadline-first (G-EDF) scheduler has been demonstrated to be useful for the multiprocessor scheduling of software with SRT requirements, and the multicore mixed-criticality (MC^2) framework using G-EDF for SRT scheduling has been proposed to safely mix HRT and SRT work on multicore UAV platforms. This dissertation addresses limitations of this prior work.

G-EDF is attractive for SRT systems because it allows the system to be fully utilized with reasonable overheads. Furthermore, previous analysis of G-EDF can provide "lateness bounds" on the amount of time between a job's deadline and its completion. However, smaller lateness bounds are preferable, and some programs may be more sensitive to lateness than others. In this dissertation, we explore the broader category of G-EDF-like (GEL) schedulers that have identical overhead characteristics to G-EDF. We show that by choosing GEL schedulers other than G-EDF, better lateness can be achieved, and that certain modifications can further improve lateness bounds while maintaining reasonable overheads. Specifically, successive jobs from the same program can be permitted to run in parallel with each other, or jobs can be split into smaller pieces by the operating system.

Previous analysis of MC^2 has always used less pessimistic execution time assumptions when analyzing SRT work than when analyzing HRT work. These assumptions can be violated, creating an overload that causes SRT guarantees to be violated. Furthermore, even in the expected case that such violations are

transient, the system is not guaranteed to return to its normal operation. In this dissertation, we also provide a mechanism that can be used to provide such recovery.

ACKNOWLEDGEMENTS

I am indebted to many people who helped me have a successful graduate school career. First and foremost is my advisor, Jim Anderson. Jim has always believed in me and encouraged me to work hard. He has also put in a great deal of work himself helping me write and revise, as well as running a research lab and bringing in the grant money needed to publish at international conferences. He has even stayed up late with students to help us finish our submissions before conference deadlines.

I also owe special thanks to Sanjoy Baruah, who advised me initially when I entered graduate school and mentored me for my first few publications. I have benefited greatly from his mentorship, as well as his skills doing and presenting theoretical research. I am also grateful to the remaining members of my committee: Kevin Jeffay, Don Smith, Mike Reiter, and Giuseppe Lipari. In addition to learning from their feedback on my research, I have also learned much in my classes with Kevin, Don, and Mike.

Before I started at UNC, I first learned research skills and solidified my decision to attend graduate school with my REU supervisors, Chris Christensen at Northern Kentucky University and Jintai Ding at the University of Cincinnati. They also coauthored my first publication. I am also indebted to the many people I coauthored papers with while at UNC: Uma Devi, Mac Mollison, Nan Guan, Greg Coombe, Bryan Ward, Glenn Elliott, Namhoon Kim, Cong Liu, and Ben Casses. Although I did not have the chance to coauthor papers with them, I have also appreciated the friendship of other members of the real-time group: Björn Brandenburg, Andrea Bastoni, Bipasa Chattopadhyay, Zhishan Guo, Jonathan Herman, Haohan Li, Chris Kenna, Alex Mills, Guru Aphale, Tahsin Kabir, Rui Liu, and Kecheng Yang.

Many of the staff members at UNC have also provided support, most notably Janet Jones, Jodie Turnbull, Dawn Andres, Tim Quigg, and Megan Erlacher. I am grateful to the Graduate School for providing support during my final year through the Dissertation Completion Fellowship.

I am thankful to Northrop Grumman and Google for hosting my internships during graduate school, and for helping with related research. I am particularly grateful to my hosts: Prakash Sarathy, Daniel Barcay, Greg Coombe, and Philip Wells.

Christ Community Church and the graduate chapter of InterVarsity Christian Fellowship have played a large role in my life in Chapel Hill. Although there are too many great people to name, I am particularly grateful to the Senior family at CCC who has regularly hosted several of us for dinner and has always been there to provide practical things like rides to the airport, and to Teresa Leonarz and the West and Ellison families who have invested in me during my time here. Hank Tarlton has faithfully served the grad InterVarsity chapter as a staff member, and Fred and Nancy Brooks have been particularly generous as chapter sponsors, allowing us to use their home and their beach house on Oak Island countless times. I have enjoyed getting to know all these people during my time here.

My life has been greatly enriched by the friendships I've developed with those I've lived with. Kevin Ludwick and I went through most of graduate school together, and I've really appreciated his friendship and willingness to put up with my quirks. In my last year, I've also enjoyed getting to know Josh Welch, Josh Fuchs, and Alan Tubbs. It has been a pleasure.

My family has also been there to support me through the process of earning a Ph.D. It has been a pleasure to be closer to my uncle Gordon, aunt Carolyn, their five sons, and my cousins Laura and Amy, all of whom live in North Carolina. My uncle Gordon and aunt Carolyn in particular have been gracious in hosting me many times, and their place has served as my home away from home during many holidays. My parents Ken and Marianna and my brothers Matthew and Chris have also been a great support through thick and thin, and I've appreciated remaining close with them.

Finally, I owe the most gratitude to God, who has gifted me with the abilities and opportunity to complete this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

C-EDF	Clustered EDF
CVA	Compliant Vector Analysis
DAG	Directed Acyclic Graph
EDF	Earliest-Deadline-First
EDF-VD	EDF with Virtual Deadlines
EDZL	Earliest-Deadline until Zero-Laxity
EEVDF	Earliest Eligible Virtual Deadline First
E-MC	Elastic Mixed-Criticality
ER-EDF	Early-Release EDF
G-EDF	Global EDF
GEL	G-EDF-like
GEL-V	GEL with Virtual Time
G-FL	Global Fair Lateness
G-GUA	Greedy Global Utility Accrual
GVD	Global Value Density
HRT	Hard Real-Time
LP	Linear Program(ming)
MC	Mixed-Criticality
MC^2	Multi-Core Mixed-Criticality
MOCA	Multiprocessor Online Competitive Algorithm
NG-GUA	Non-Greedy Global Utility Accrual
OCBP	Own-Criticality Based Priority
P-EDF	Partitioned EDF
PP	Priority Point
PWCET	Provisioned WCET
RED	Robust Earliest Deadline
RTB	Robust Total Bandwidth
SRT	Soft Real-Time

SVO	Sporadic with Virtual time and Overload
UAV	Unmanned Aerial Vehicle
WCET	Worst-Case Execution Time

CHAPTER 1: INTRODUCTION

The goal of this dissertation is to improve the state of the art for *soft real-time* (*SRT*) multiprocessor scheduling by improving response times without significantly increasing runtime overheads, and by providing the ability to recover from overload situations when more work arrives than is expected. This work is motivated by next-generation *unmanned aerial vehicles* (*UAVs*), which will require SRT scheduling for advanced functionality such as mission planning. While prior work can guarantee bounds on response times, this dissertation provides methods to improve upon those bounds without incurring additional overheads, and provides methods that give more fine-grained control of such bounds. Furthermore, in order to reduce the size, weight, and power requirements for these UAVs, it will be necessary to run *hard real-time* (*HRT*) safety-critical software on the same multicore system as SRT mission-critical software. When provisioning the SRT portion of the system, assumptions about execution time could be made that are insufficiently pessimistic. As a result, the system may become *overloaded*. This dissertation also provides a method to manage such a situation.

We begin this chapter by providing a general introduction to real-time systems. We then provide a description of the basic scheduling framework used by the UAVs we consider. We then state the thesis of this dissertation, describe its contributions, and provide an outline for the remainder of the dissertation.

1.1 Real-Time Systems

Almost any computer system must produce valid outputs in order to be considered a “correct” system. In a real-time system, the result must also be produced at the right time. The precise definition of “at the right time” depends on the type of system.

A system is typically defined to be “HRT” if each job (i.e., invocation of a program, or “task”), has a deadline by which it must complete in order for the system to be correct. This definition of correctness is needed if drastic consequences could result from a missed deadline. For example, a task that adjusts flight surfaces on an aircraft has such a requirement, as a missed deadline could result in a crash. In order to guarantee the correctness of such a system, it is typically necessary to make highly pessimistic assumptions about system behavior, in order to ensure that a deadline cannot be missed under any possible circumstance. This usually requires over-provisioning the system.

A system is defined to be "SRT" if it has less stringent requirements. In such a system, each job typically still has a deadline, but the system may be deemed correct even if some jobs miss their deadlines. For example, one type of SRT constraint would be the requirement that some fraction of all deadlines in the system be met. This type of correctness is often sufficient. For example, a video decoding system that operates at 50 frames per second must decode each frame within a 20 ms period, or the video may visibly skip. Such a skip is not catastrophic, and the reduced pessimism can allow a system to be more fully provisioned. After describing a task model next, we will describe the particular SRT criterion that we use.

1.1.1 Task Model and SRT Criterion

In this dissertation, we consider scheduling under the *sporadic task model*. In order to describe this model, we depict in Figure 1.2 an example task running by itself. (The key for all figures in this chapter is given in Figure 1.1.) A task represents one process that is composed of a (potentially infinite) series of discrete jobs. When a new job is available for execution, we say that job is *released* by the task. However, because the task is a single process, if the job's predecessor has not yet completed, then the new job must wait to actually begin execution. (This requirement is relaxed in Chapter 4 of this dissertation.)

The system is composed of a set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n tasks.

The *worst-case execution time* (*WCET*) of each task τ_i, denoted C_i, is an upper bound on the execution time for any of its jobs. In Figure 1.2, $C_1 = 2$ ms, so no job runs for over 2 ms. However, some jobs run for only 1 ms, as allowed by the model.

The *minimum separation time* of each task τ_i, denoted T_i, is the minimum amount of time between two job releases. In Figure 1.2, $T_1 = 3$ ms, so job releases occur at least three units apart. However, after the job at time 9 is released, no new job is released until time 14, as allowed by the model.

The *absolute deadline* of a *job* is the point in time by which that job should finish. In Figure 1.2, the absolute deadline of the first job is at time 2. As discussed above, the precise interpretation of "should finish" depends on whether the system is HRT or SRT. The *relative deadline* of each *task* τ_i, denoted D_i, is the time between the release time and absolute deadline of each job of that task. In Figure 1.2, $D_1 = 2$ ms, so for example, the job released at time 9 has its absolute deadline at time 11.

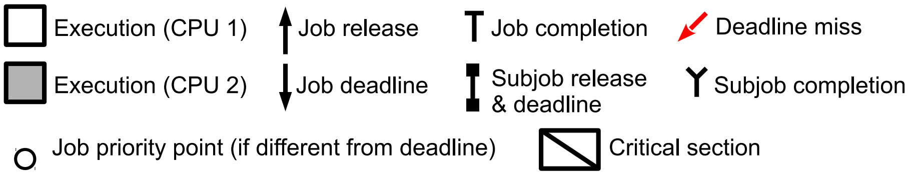


Figure 1.1: Key for all figures in Chapter 1.

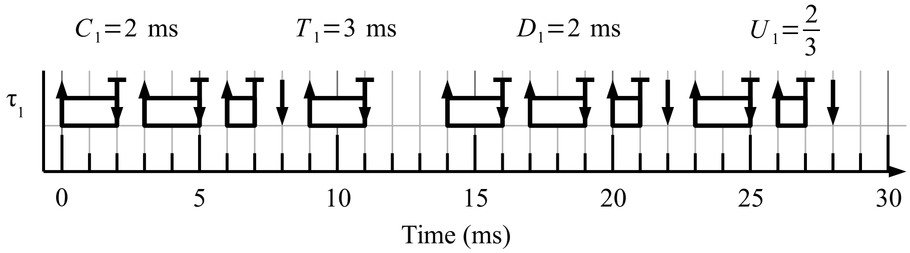


Figure 1.2: Example of a sporadic task.

For simplicity, within this chapter we will consider *implicit-deadline* task systems in which for each task τ_i, $D_i = T_i$. However, all original work in the remainder of this dissertation is also applicable to *arbitrary-deadline* task sets that may violate this assumption. For the purpose of examples, we will often use the notation $\tau_i = (C_i, T_i)$ for tasks within implicit-deadline task systems.

A final parameter of each task τ_i is its *utilization*, denoted U_i. A task's utilization is simply the ratio of its WCET to its minimum separation time: $U_i = \frac{C_i}{T_i}$. The utilization of a task is significant because it indicates the long-term processor share needed by the task, in the worst case.

Suppose a job is released at time r, has an absolute deadline at time d, and completes at time t. Then, its *response time* is $t - r$, its *lateness* is $t - d$, and its *tardiness* is $\max\{0, t - d\}$, as depicted in Figure 1.3. Observe that, if a job completes no earlier than its deadline, then its lateness and tardiness are identical and nonnegative. Otherwise, its lateness is negative and its tardiness is zero.

With these definitions in place, we now specify the particular SRT criterion we use: *bounded lateness*. If a task has an upper bound on the lateness of any of its jobs, then such a bound is called a *lateness bound*. If all tasks have lateness bounds, then the system has bounded lateness. *Bounded tardiness* (with *tardiness bounds*) and *bounded response time* (with *response-time bounds*) are equivalent to bounded lateness in the sense that a system has bounded lateness if and only if it has bounded tardiness and if and only if it has bounded response time. Much past work, e.g., (Devi and Anderson, 2008; Leontyev and Anderson, 2010), has used the bounded tardiness criterion for SRT. All three of these criteria are useful because each guarantees that each

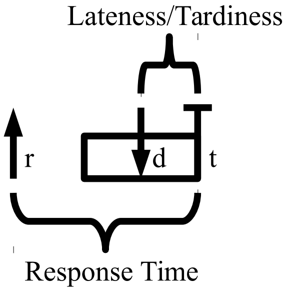


Figure 1.3: Response time, lateness, and tardiness. If t were before d, then lateness would be negative, while tardiness would be zero.

task receives sufficient processor share in the long term. Furthermore, lateness bounds can indicate that jobs must finish *before* their deadlines, whereas tardiness bounds cannot. For the remainder of this dissertation, when "SRT" is used without qualification, the bounded lateness criterion is in use.

The practicality of the bounded lateness model does depend on having reasonably small lateness bounds, and smaller lateness bounds generally provide a practical improvement. Therefore, one area of focus of this dissertation will be choosing appropriate scheduling algorithms to minimize lateness bounds.

1.1.2 Scheduling Algorithms

We will now discuss several common classes of scheduling algorithms. We first define relevant terms. A task system is *feasible* if some scheduling algorithm can schedule it correctly. When considering HRT, a system is said to be scheduled "correctly" if no job misses its deadline. When considering SRT, a system is said to be scheduled "correctly" if it has bounded lateness. A scheduler is said to be *optimal* if it correctly schedules any feasible task system.

A common uniprocessor scheduling algorithm is the *earliest-deadline-first* (*EDF*) scheduling algorithm, in which jobs are prioritized by absolute deadline, with ties broken arbitrarily but consistently. EDF is an optimal uniprocessor algorithm for both HRT and SRT systems. In particular, EDF can schedule any implicit-deadline task system with $\sum_{\tau_i \in \tau} U_i \leq 1$. An example EDF schedule is depicted in Figure 1.4.

When considering multiprocessor systems, we denote the number of processors as m. There are multiple ways to extend EDF scheduling to a multiprocessor setting. One method is *partitioned EDF* (*P-EDF*). Under P-EDF, each task is statically assigned to a processor, and each processor schedules its tasks using EDF. For implicit-deadline task systems, assigning tasks to processors is equivalent to solving a bin-packing-like

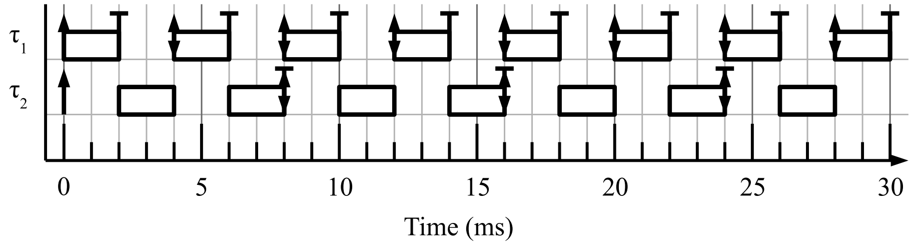


Figure 1.4: EDF schedule of $\tau_1 = (2,4)$ and $\tau_2 = (4,8)$.

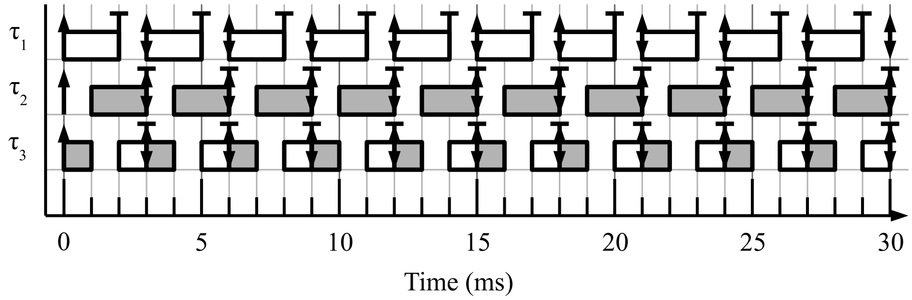


Figure 1.5: Correct (both HRT and SRT) schedule of a system with three tasks where each $\tau_i = (2,3)$.

problem. The items are the *n* tasks, with weights equal to utilizations, and the bins are the *m* processors, each with capacity one.

The primary limitation of P-EDF is related to the bin-packing problem: there are task systems that are feasible on *m* processors with techniques other than partitioning, but that cannot be partitioned onto the same set of processors. As an example, consider the task system with three identical tasks $(2,3)$. Each task has a utilization of $\frac{2}{3}$, so no two tasks can be allocated on the same processor and three processors are required. However, this task system is actually feasible using only two processors. As an example, Figure 1.5 depicts a correct schedule for this task system on only two processors when all jobs are released as early as possible. Notice that in this schedule, jobs of τ_3 *migrate* between processors during execution.

An alternative to P-EDF is *global EDF* (*G-EDF*), in which all processors share a global run queue and the *m* jobs with the soonest deadlines execute. A G-EDF schedule of our running example (as in Figure 1.5) is depicted in Figure 1.6. Unfortunately, as can be seen in the figure, all jobs of τ_3 miss their deadlines. This demonstrates that G-EDF is not optimal in a HRT sense. However, notice that no job of τ_3 misses its deadline by more than 1 ms. In fact, Devi and Anderson (2008) demonstrated that G-EDF is in fact optimal in a SRT sense.

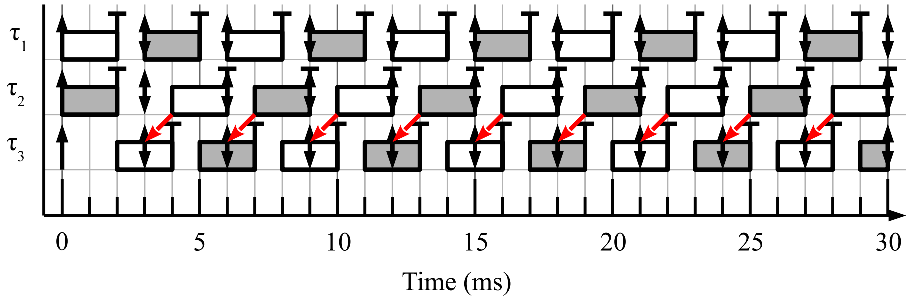


Figure 1.6: G-EDF schedule of the same system as Figure 1.5. This schedule is SRT-correct, but not HRT-correct.

Schedulers that are optimal in a HRT sense for implicit-deadline sporadic task systems do exist, e.g., (Anderson and Srinivasan, 2004; Baruah et al., 1996; Megel et al., 2010; Regnier et al., 2011). However, all such schedulers either are difficult to implement in practice or cause jobs to frequently be preempted by other jobs or migrated between CPUs. Even the schedule in Figure 1.5, which is for a very simple task system, requires each of τ_3's jobs to incur a migration. Furthermore, in order to achieve optimality, it is necessary to change the relative priorities of jobs while those jobs are running. In Figure 1.5, each of τ_3's jobs initially has a higher priority than the corresponding job of τ_2, but only for 1 ms. This type of priority change, which can cause problems for locking protocols Brandenburg (2011), does not occur under G-EDF. Therefore, G-EDF remains a good choice for SRT systems, and we use G-EDF as the basis for the work in this dissertation.

On systems with a large number of processing cores, the overheads incurred by locking and maintaining a global run queue may result in large overheads (Bastoni et al., 2010). Therefore, a compromise between P-EDF and G-EDF called *clustered EDF* (*C-EDF*), where tasks are partitioned onto clusters of CPUs and G-EDF is used within each cluster, is preferable in such cases. Because G-EDF is used within each cluster, the work in this dissertation is directly applicable.

1.2 Mixed Criticality and MC^2

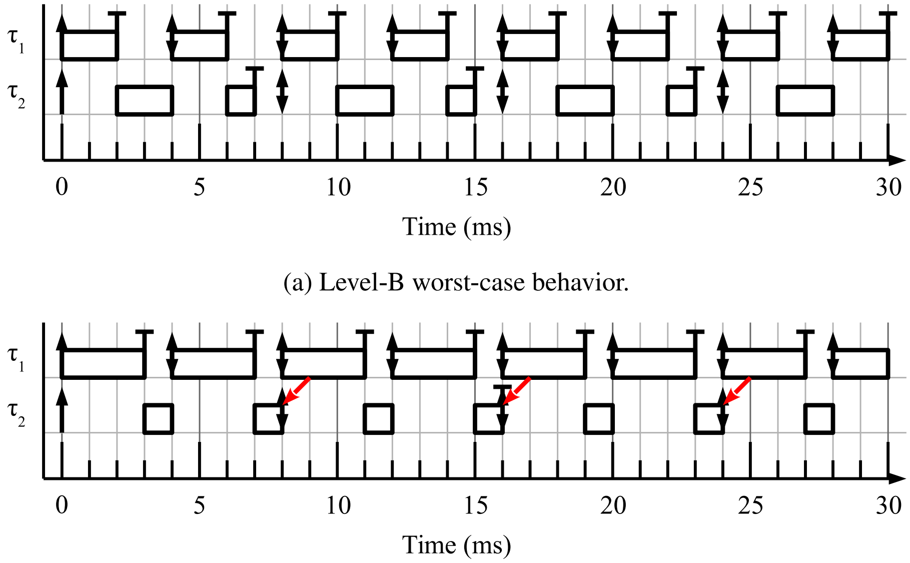
Sometimes different applications that will be run on the same physical machine have different requirements for timing correctness. For example, as discussed in the beginning of Section 1.1, some applications have HRT constraints (requiring all deadlines to be met), while others have SRT constraints (where bounded lateness is acceptable). This sort of mixture of requirements will become increasingly relevant for future generations of UAVs, as more tasks that have traditionally been performed by humans are instead performed by software. For example, safety-critical software performing functions such as flight control continues to have stringent HRT constraints, whereas mission-critical software performing planning functions has only SRT constraints. Running both sets of software on the same machine could significantly reduce the size, weight, and power required for the aircraft.

Furthermore, there may be further distinctions in requirements than simply the difference between HRT and SRT constraints. For example, some tasks may be so critical that it is necessary to use WCET estimates determined by a tool that provides a provable upper bound on execution time, in order to provide the strongest possible guarantee that no WCET is exceeded. Such a level of certainty may be necessary in order for the system to be acceptable to a relevant certification authority. However, for other tasks, it may be sufficient to use less pessimistic WCET estimates, such as those determined by measuring the largest execution on a real system and multiplying by a safety factor.

Under most real-time scheduling analysis, the system can only be deemed correct if it can be proven to be correct *even using the most pessimistic assumptions for all tasks*. For example, in order to prove that the flight-control software will behave correctly, it is necessary to use highly pessimistic WCET estimates for the mission-control software as well. On a multicore system, this could be orders of magnitude more pessimistic than actual behavior (Mollison et al., 2010), resulting in a system that is unnecessarily underutilized.

Mixed-criticality scheduling algorithms and analysis address this problem. Vestal (2007) proposed that a single scheduling algorithm could be analyzed under multiple sets of assumptions about WCET estimates. The system has a finite number of criticality levels, and each task is assigned a *criticality level* and, for each criticality level in the system including its own, a *provisioned worst-case execution time* (*PWCET*). For arbitrary level ℓ, the system is considered to be *correct* at level-ℓ if all tasks with a criticality level at or above level ℓ are scheduled correctly, assuming that no job of any task exceeds its level-ℓ PWCET. An example is depicted in Figure 1.7 with two criticality levels, A (high) and B (low). Figure 1.7(a) depicts the worst-case behavior assuming that no job of any task exceeds its level-B PWCET, and Figure 1.7(b) depicts the worst-case behavior assuming that no job of any task exceeds its level-A PWCET. Observe that deadlines are only missed in Figure 1.7(b), that only τ_2 (which is a level-B task) has jobs that miss their deadlines, and that this schedule involves jobs exceeding their level-B PWCET. The provided scheduling algorithm correctly schedules this task system.

Motivated by the same UAV system considered in this dissertation, Herman et al. (2012) proposed a specific scheduler, the *multi-core mixed-criticality* (*MC*^{2}) scheduler, that supports four criticality levels,



(a) Level-B worst-case behavior.

(b) Level-A worst-case behavior.

Figure 1.7: Possible schedules for a uniprocessor mixed-criticality system with two criticality levels, A (high) and B (low), both with HRT requirements. Level-A τ_1 has a minimum separation time of 4 ms, a level-A PWCET of 3 ms, and a level-B PWCET of 2 ms. Level-B τ_2 has a minimum separation time of 8 ms, a level-A PWCET of 4 ms, and a level-B PWCET of 3 ms. τ_1 is statically prioritized over τ_2.

A through D. (An earlier version of MC^2 that supports five criticality levels was proposed by Mollison et al. (2010).) The architecture of MC^2 is depicted in Figure 1.8. Each criticality level is scheduled independently, and higher criticality levels are statically prioritized over lower criticality levels. Level A has HRT requirements. Tasks are partitioned onto CPUs and scheduled using a per-CPU table with a precomputed schedule. Level B also has HRT requirements and requires tasks to be partitioned onto CPUs, but uses P-EDF for scheduling. Level C has SRT requirements, and tasks are scheduled using G-EDF. Finally, level D is *best effort*, which means that it has no real-time guarantees. Level D can be scheduled using the general-purpose scheduler provided by the underlying operating system (OS).

In this dissertation, we focus on scheduling at level C.

1.3 Past SRT Work

We now briefly review the past work on SRT scheduling that is directly relevant to this dissertation. Further review and other work on SRT scheduling is discussed in Chapter 2.

As mentioned above, past work on bounded lateness has actually been stated in the form of the equivalent condition of bounded tardiness. The seminal work on bounded tardiness was that by Devi and Anderson

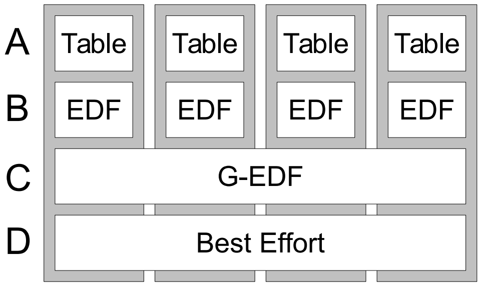


Figure 1.8: Architecture of MC^2.

(2008), who considered G-EDF scheduling. Devi and Anderson showed that the tardiness of any job of τ_i is at most $x + C_i$, where

$$x \triangleq \frac{C_{\text{sum}} - C_{\text{min}}}{m - U_{\text{sum}}},$$

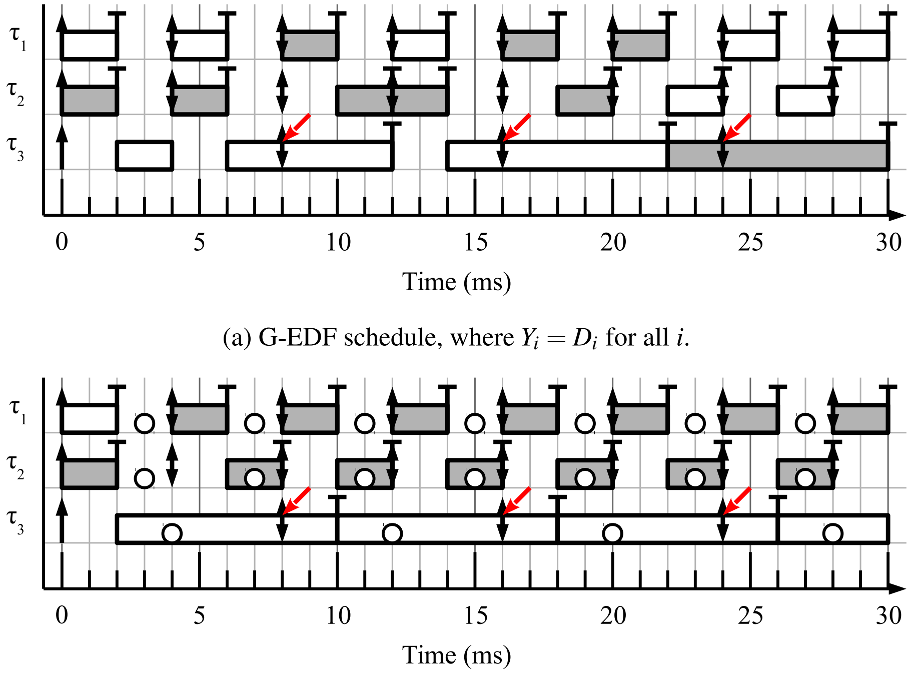
C_{sum} is the sum of the $m-1$ largest values of C_i, C_{min} is the smallest value of C_i, and U_{sum} is the sum of the $m-2$ largest values of U_i.

Leontyev and Anderson (2010) performed significant extensions to Devi and Anderson's initial work. Rather than limiting their analysis to G-EDF, they considered a broader class of *window-constrained* schedulers. Window-constrained schedulers have a specific property that guarantees that each job will eventually become and remain the highest priority job in the system. Leontyev and Anderson also considered *restricted supply*, in which some processors are not fully available to the task system being scheduled. The scheduling of level C in MC^2 can be analyzed by using restricted supply to model execution at levels A and B.

The tardiness bounds provided by Leontyev and Anderson (2010) are significantly more complex than those provided by Devi and Anderson (2008), due to both of the generalizations provided. Therefore, we do not provide the specific expressions here.

Leontyev et al. (2011) considered a task model that is more general than the sporadic task model, using a framework called *real-time calculus*. They considered *delay bounds*, which correspond to response-time bounds under the sporadic task model. As discussed above, bounded response time is equivalent to bounded lateness and bounded tardiness. Leontyev et al. provided a method to determine whether a given set of response-time bounds could be met.

Leontyev et al. also provided a method to determine lateness bounds for a family of *G-EDF-like* (*GEL*) schedulers. Recall that, under G-EDF, jobs are prioritized based on their absolute deadlines, and the absolute



(a) G-EDF schedule, where $Y_i = D_i$ for all i.

(b) G-FL schedule, with $Y_1 = Y_2 = 3$ and $Y_3 = 4$.

Figure 1.9: Comparison of two GEL schedules of the same task system, with $\tau_1 = \tau_2 = (2,4)$ and $\tau_3 = (8,8)$.

deadline of each job of τ_i is D_i units of time after its release. Under a GEL scheduler, jobs are prioritized based on *priority points* (*PPs*) that may differ from absolute deadlines. In an analogous manner to G-EDF and absolute deadlines, a job under a GEL scheduler has a higher priority than another if it has an earlier PP. A per-task constant Y_i (priorit$Y$) takes the place of D_i: the PP of each job is Y_i units of time after its release. The implementation of any GEL scheduler is identical to that of G-EDF, except that Y_i is used for prioritization in place of D_i.

An example comparing two GEL schedulers is depicted in Figure 1.9. Figure 1.9(a) depicts G-EDF itself, where $Y_i = D_i$ for all i, and Figure 1.9(b) depicts a different GEL scheduler, the *global fair lateness* (*G-FL*) proposed in Chapter 3 of this dissertation.

1.4 Thesis Statement

The original bounds provided by Devi and Anderson (2008) were tighter for G-EDF than those provided by Leontyev and Anderson (2010), due to the increased generality considered by Leontyev and Anderson.

However, further improvements are possible, and the model considered by Devi and Anderson requires implicit deadlines (each $D_i = T_i$) and does not immediately generalize to other GEL schedulers.

While Leontyev et al. (2011) provided analysis for arbitrary GEL schedulers, they did not provide substantial guidance on how to select values of Y_i in order to obtain desired scheduler characteristics. Furthermore, although they allowed delay bounds to be specified, they did not provide an efficient method to obtain the tightest possible delay bounds using their analysis, and the bounds provided are not as tight as possible for sporadic task systems given the more general task model considered.

As discussed in the last section, restricted supply analysis can be used to account for level-A and level-B work when considering level-C behavior in MC^2. In order to do so, accurate WCETs must be used in the analysis. Mollison et al. (2010) used the level-C PWCET for each level-A or -B task, in order to maximize the actual utilization of the system. However, because level-A and -B PWCETs for tasks at all levels are more pessimistic than level-C PWCETs, it is possible that level-A or -B tasks may sometimes exceed their level-C PWCETs, if the level-C PWCETs are insufficiently pessimistic. This *overload* compromises guarantees at level C.

In order to address these limitations, we will support the following thesis:

> *G-EDF can be modified to support smaller lateness bounds than previous work allows, with more flexibility to specify desired lateness criteria. Furthermore, such modifications do not violate the assumptions required for multiprocessor locking protocols, and the modified scheduler is easier to implement and/or has lower overheads than known HRT-optimal schedulers. In addition, recovery from overloads caused by tasks in MC^2 overrunning their level-C PWCETs can be facilitated by modifying the scheduler to delay job releases dynamically.*

1.5 Contributions

We now describe our contributions in support of this thesis.

Compared to Devi and Anderson (2008), we provide further improvements on the tightness of tardiness/lateness bounds, and also provide a way to handle arbitrary deadlines (which may differ from minimum separation times) and arbitrary GEL schedulers. Our method does not require the additional pessimism from the more general models considered by Leontyev and Anderson (2010) and Leontyev et al. (2011). We also provide methods to choose the best lateness bounds by optimizing criteria such as maximum or average lateness.

Yet smaller lateness bounds are possible by further modifying the scheduler. In this dissertation, we discuss two techniques to do so: allowing multiple jobs of the same task to run simultaneously, and splitting jobs into smaller subjobs.

In the context of MC^2, we generalize the restricted supply analysis from Leontyev and Anderson (2010) by accounting for level-C PWCET overruns at levels A, B, and C. We also provide a method to recover at runtime from such an overload.

We now discuss each contribution in more detail.

1.5.1 Analysis of GEL Schedulers

In Chapter 3, we discuss improved analysis of GEL schedulers and propose methods to choose GEL schedulers to obtain the best lateness bounds. The basic strategy for our analysis of lateness bounds is essentially that from (Devi and Anderson, 2008), but we make several improvements.

As discussed above, Devi and Anderson define the tardiness bound for τ_i as $x+C_i$, with a single value of x for the entire task system. One fundamental change we make is to define a separate x_i for each τ_i. We also allow for relative PPs that differ from minimum separation times, which allows us to consider both arbitrary deadlines and arbitrary GEL schedulers.

The tardiness bound $x+C_i$ from Devi and Anderson is equivalent to the response-time bound D_i+x+C_i. In our analysis, Y_i replaces D_i, so we derive response-time bounds of the form $Y_i+x_i+C_i$. Stated as lateness bounds, these are of the form $Y_i+x_i+C_i-D_i$.

We define a term

$$S_i(Y_i) \triangleq C_i \cdot \max\left\{0, 1-\frac{Y_i}{T_i}\right\}.$$

that accounts for the difference between Y_i and T_i, and we use it to provide the following bound on x_i.

$$x_i \geq \frac{\sum_{m-1 \text{ largest}}(x_jU_j+C_j-S_j(Y_j))+\sum_{\tau_j\in\tau}S_j(Y_j)-C_i}{m} \tag{1.1}$$

Notice that x_i effectively appears on *both* sides of (1.1), so (1.1) cannot be used directly to compute x_i. However, we show how to define a linear program in order to determine the smallest values of x_i that satisfy (1.1) for all i. Furthermore, if each Y_i is treated as a variable rather than as a constant, we can also use linear programming to select Y_i values in order to optimize any linear criterion of lateness bounds, such as minimizing the maximum or average lateness bound.

We also propose G-FL, the same scheduler that was depicted in Figure 1.9(b). Under G-FL, for each τ_i,

$$Y_i \triangleq D_i - \frac{m-1}{m} \cdot C_i.$$

As can be seen in Figure 1.9, G-FL can provide better lateness than G-EDF. We also show that it provably provides the smallest possible maximum lateness bound, given our analysis.

1.5.2 Removing The Intra-Task Precedence Constraint

In Chapter 4, we propose a task system modification that can further reduce lateness. Recall that, because a task is a single-threaded process, each job must wait to begin executing until its predecessor completes. We refer to this as the *intra-task precedence constraint*. If jobs run in separate threads, however, this constraint can be removed, and multiple jobs of the same task can execute at the same time on different processors. Doing so can further reduce lateness bounds.

Some of the pessimism in previous lateness bounds results directly from the fact that work can be backed up within a task, even when there are idle CPUs. It is possible that a task has several jobs that have sufficient priority to run, but only one can make progress. Without the intra-task precedence constraint, however, multiple pending jobs from the same task can make progress at the same time. This change allows us to derive smaller bounds.

Furthermore, in the presence of the intra-task precedence constraint, the amount by which a task is backed up can grow unboundedly even when there are idle CPUs. Therefore, we must require that $U_i \leq 1$ holds for every task. However, without the intra-task precedence constraint, this requirement is no longer necessary, and the simple system utilization requirement $\sum_{\tau_j \in \tau} U_j \leq m$ is sufficient.

1.5.3 Job Splitting

In Chapter 5, we propose another modification to the scheduler to improve lateness bounds. The current lateness bounds depend heavily on task execution times. A task's execution time can be reduced by an integral factor if each of its jobs is split. For example, a task that has a WCET of 2 ms and a period of 4 ms could have its jobs split in half, resulting in a task with a WCET of 1 ms and a period of 2 ms. Notice that the utilization of the task remains constant. Each consecutive pair of subjobs in the split task corresponds to a real job in the original task.

An example of job splitting under G-EDF is depicted in Figure 1.10. Figure 1.10(a) depicts an example schedule in the absence of splitting. Notice that $\tau_{3,0}$ completes 4 ms late. Figure 1.10(b) depicts the schedule where jobs of τ_3 are split into two subjobs. $\tau_{i,j,k}$ is used to denote subjob k of $\tau_{i,j}$. Notice that $\tau_{3,0}$ now completes only 3 ms late.

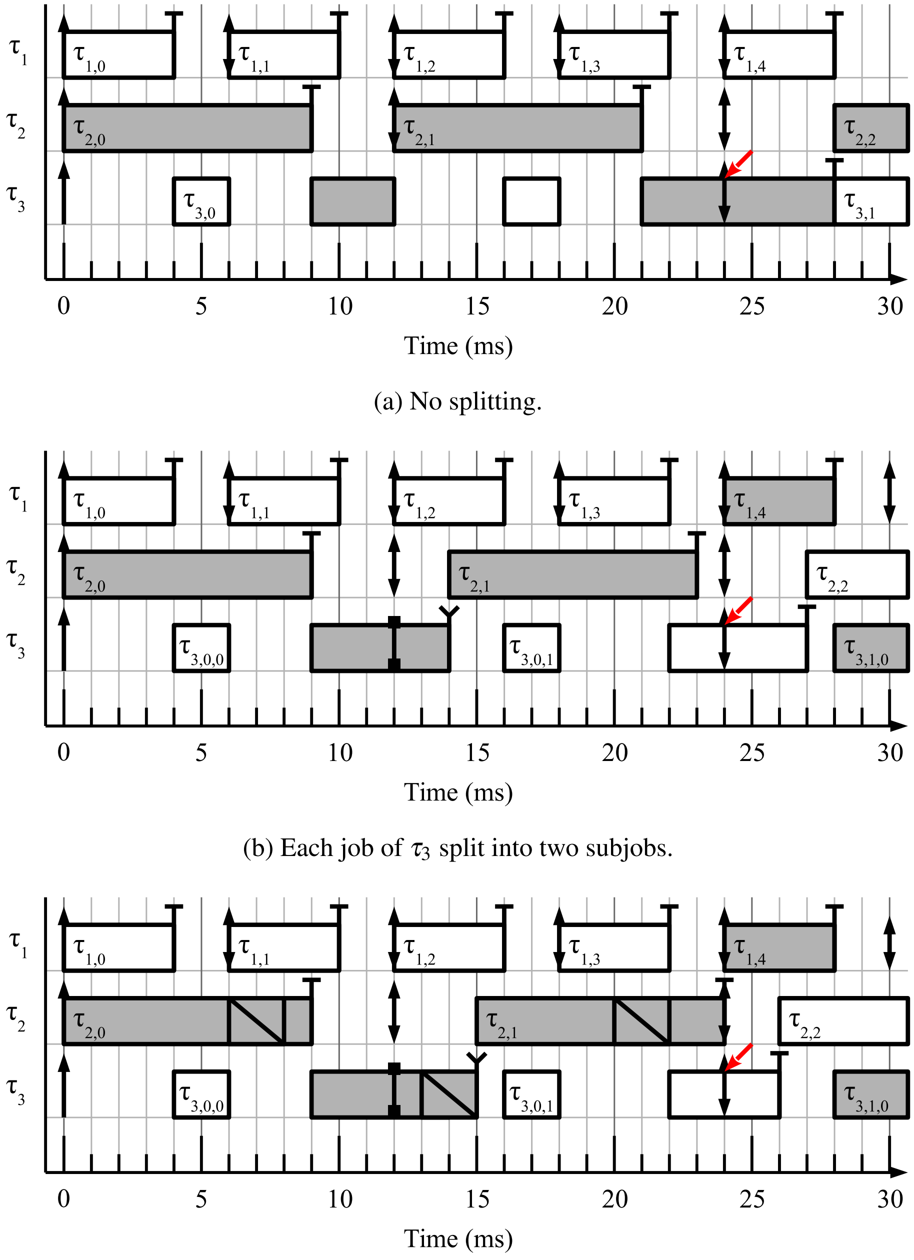
Job splitting becomes more complicated in the presence of critical sections, because many locking protocols require that job priorities do not change during execution, but every time a subjob ends, the priority of the underlying job changes. However, this problem can be overcome by not allowing a subjob to end while holding or waiting for a lock, reducing the length of the subsequent subjob. This procedure is depicted in Figure 1.10(c), where $\tau_{3,0,0}$ runs for 8 ms instead of 7 ms, and $\tau_{3,0,1}$ then runs for only 6 ms.

In the absence of overheads and critical sections, because task utilizations remain constant with splitting, lateness bounds could be made arbitrarily close to zero. However, on a real system, more overheads are incurred as a result of job splitting. Whenever a subjob ends, the OS must decide what job should subsequently be scheduled, creating more scheduling decisions. Additionally, jobs may be preempted at subjob completion, rather than only at job releases, causing a potential loss of cache affinity. These additional overheads effectively *increase* a task's utilization, so it is necessary to account for these overheads in order to determine the actual benefits of job splitting.

Our lateness analysis remains correct if jobs are allowed to begin execution prior to their proper release times, as long as job PPs are determined based on their proper release times. Therefore, when one subjob completes, it is sufficient to simply lower the priority of the underlying job. It is not necessary to unconditionally preempt the job. Furthermore, even if the job does need to be preempted, it can simply be added to the ready queue immediately; it is not necessary to set a timer for a future release. This approach significantly limits the additional overheads that splitting creates.

In order to determine the impacts of job splitting on lateness bounds, it is necessary to use realistic measures of overhead. Therefore, we implemented G-FL with job splitting in LITMUS^{RT},[1] a real-time extension to the Linux kernel developed at UNC, and measured relevant overheads. We used these overheads

[1] `http://www.litmus-rt.org/`



(a) No splitting.

(b) Each job of τ_3 split into two subjobs.

(c) Each job of τ_3 split into two subjobs, in the presence of critical sections.

Figure 1.10: Schedules of a task system with $\tau_1 = (4,6)$, $\tau_2 = (9,12)$, and $\tau_3 = (14,24)$, to illustrate job splitting.

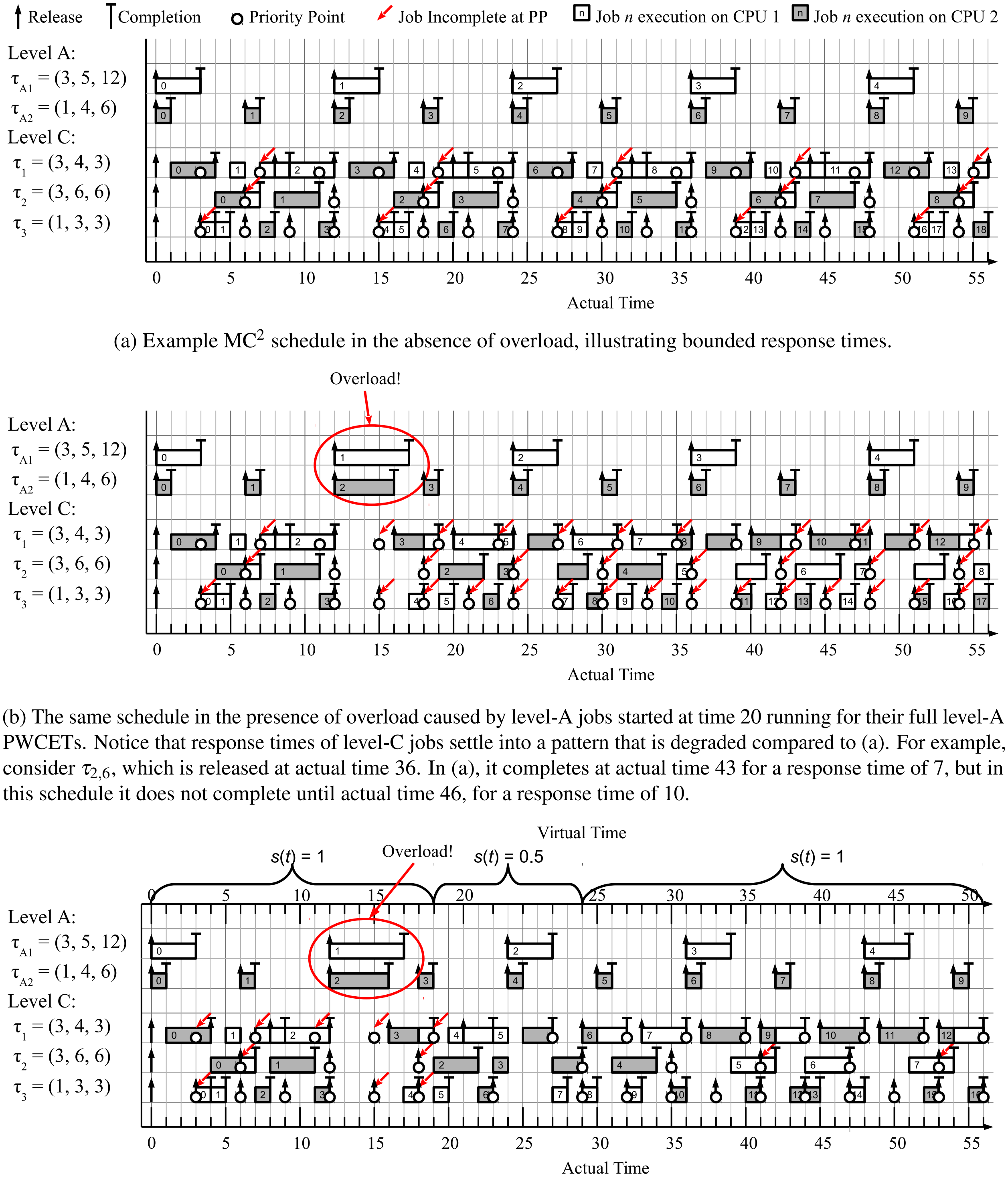
in lateness-bound computations and showed that significant reductions in lateness bounds are possible, even accounting for overheads and even in the presence of critical sections.

1.5.4 Handling Overload in MC^2

In Chapter 6, we consider the problem of overload within MC^2. In order to address scheduling in MC^2, we add restricted supply to our analysis of GEL schedulers. Our basic strategy for handling restricted supply is like that of Leontyev and Anderson (2010), but because we do not use the full generality of window-constrained scheduling, our bounds are tighter. Furthermore, we improve analysis under the case when most processors are not fully available, but have minimal supply restriction. This is the common case under MC^2, because most processors have both level-A and -B tasks, but the level-A and -B PWCETs for those tasks are highly pessimistic, resulting in a great deal of slack for level C.

Because level-C PWCETs are not as pessimistic as level-A or -B PWCETs, it is possible that jobs at any level may overrun their level-C PWCETs. (MC^2 can optionally enforce job *budgets* to ensure that jobs do not overrun their PWCETs at their own criticality levels, but even if this feature is enabled, level-A and -B jobs can still overrun their level-C PWCETs.) The effects of overload are depicted in Figure 1.11, which depicts an MC^2 system that has only level -A and -C tasks. For this example, level-A tasks are depicted using the notation (C_i^C, C_i^A, T_i), where C_i^C is its level-C PWCET and C_i^A is its level-A PWCET, while level-C tasks are depicted using the notation (C_i^C, T_i, Y_i). Figure 1.11(a) depicts a schedule in the absence of overload, while Figure 1.11(b) depicts the results of some level-A jobs running for their full level-A PWCETs. As a result of the overload, all future job release times are impacted.

To analyze this situation, we generalize both the restricted supply model and the task model. We then describe a technique that can be used to recover from such an overload situation. Our technique is depicted in Figure 1.11(c). We use a notion of *virtual time*, as originally introduced by Zhang (1990) and used in uniprocessor real-time scheduling by Stoica et al. (1996). Essentially, we maintain a secondary “virtual” clock that, at actual time t, is operating at a speed of $s(t)$ relative to the actual clock. In the absence of overload, $s(t) = 1$, so that the two clocks operate at the same speed. However, after an overload occurs, the OS can choose to use a slower speed, as occurs from actual time 19 to actual time 29 in Figure 1.11(c). Our technique does not prescribe a particular choice of $s(t)$, but we provide experimental results that provide guidance.



(a) Example MC^2 schedule in the absence of overload, illustrating bounded response times.

(b) The same schedule in the presence of overload caused by level-A jobs started at time 20 running for their full level-A PWCETs. Notice that response times of level-C jobs settle into a pattern that is degraded compared to (a). For example, consider $\tau_{2,6}$, which is released at actual time 36. In (a), it completes at actual time 43 for a response time of 7, but in this schedule it does not complete until actual time 46, for a response time of 10.

(c) The same schedule in the presence of overload and our recovery techniques. Notice that response times of level-C jobs settle into a pattern that is more like (a) than to (b).

Figure 1.11: Example MC^2 task system, illustrating overload and recovery.

Job minimum separation times and relative PPs are defined in terms of the virtual clock, rather than the actual clock. This has the effect of reducing the number of level-C job releases for an interval of time and allows the system to recover from overload. The time required to do so is called a *dissipation time*. We derive *dissipation bounds*, or upper bounds on the dissipation time.

1.6 Organization

In Chapter 2, we discuss relevant background work in SRT scheduling and overload management. Then, in Chapter 3, we discuss our analysis of GEL schedulers, G-FL, and our linear programming techniques to compute and optimize lateness bounds. In Chapter 4, we discuss the impact of removing intra-task precedence constraints, and in Chapter 5 we discuss the impact of job splitting. Afterward, in Chapter 6, we discuss analysis that includes restricted supply, overload recovery, and dissipation bounds. Finally, in Chapter 7, we offer concluding thoughts and discuss future work.

CHAPTER 2: BACKGROUND

In this chapter, we discuss prior work on SRT scheduling and overload management. We first survey the work on bounded tardiness that forms the basis for the work in this dissertation in Section 2.1 and related overhead analysis in Section 2.2. We then briefly survey alternative models for SRT systems in Section 2.3. Then, we survey past work on overload management using a notion of "value functions" in Section 2.4. In Section 2.6 we briefly survey some techniques for overload management that work by modifying the rate at which work is performed. Finally, in Section 2.7, we survey overload management within mixed-criticality systems.

2.1 Prior Bounded Tardiness Work

As mentioned in Chapter 1, Devi and Anderson (2008) provided tardiness bounds for implicit-deadline sporadic task systems scheduled under G-EDF. They compared G-EDF to an *ideal scheduler* that continuously maintains for each task a processor share equal to its utilization. The difference in allocation between what a task receives under G-EDF and under the ideal scheduler is called *lag*. Lag can be analyzed at various points in the schedule in order to derive tardiness bounds. The most significant points in the analysis occur when all CPUs become simultaneously busy. Because some processor was idle, there can be at most $m-1$ tasks that have remaining work just before such a time. That insight allowed Devi and Anderson to define a value x such that the tardiness of a task τ_i is at most $x+C_i$. The value of x they defined is as follows:

$$x \triangleq \frac{C_{\text{sum}} - C_{\text{min}}}{m - U_{\text{sum}}},$$

where C_{sum} is the sum of the $m-1$ largest values of C_i, C_{min} is the smallest value of C_i, and U_{sum} is the sum of the $m-2$ largest values of U_i.

Bounded tardiness is established by mathematical induction over a set of jobs. We denote job k of task τ_i with $\tau_{i,k}$. When analyzing a job $\tau_{i,k}$ with a deadline at $d_{i,k}$, jobs with lower priority than $d_{i,k}$ can be ignored. Induction begins with the highest-priority job in the system, and the inductive assumption is that no job with

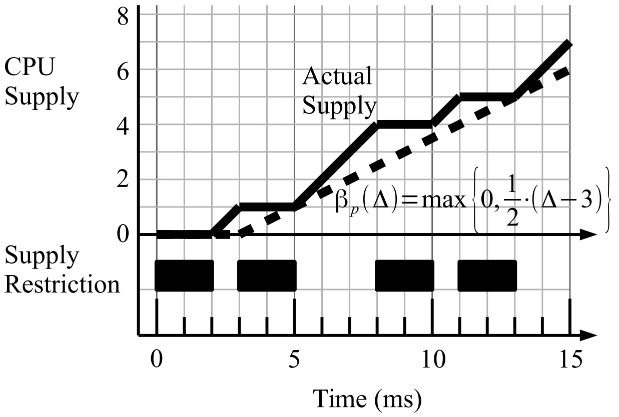


Figure 2.1: Depiction of the service functions used by Leontyev and Anderson (2010).

priority higher than $\tau_{i,k}$ has tardiness larger than stated in the proof. The lag is tracked inductively at key points in the execution of the system, so that a bound on the lag of the system at $d_{i,k}$ can be determined. From that lag bound the tardiness bound for $d_{i,k}$ is established.

Leontyev and Anderson (2010) provided tardiness bounds for a class of "window-constrained" schedulers. Under such a scheduler, jobs are prioritized on the basis of a PP, and the system executes the eligible jobs with the earliest PPs. Furthermore, there must exist constants ϕ_i and ψ_i such that, if a job of τ_i has a release at time r, a deadline at time d, and a PP at time y (priorit*y*), then it must be the case that $r - \phi_i \le y \le d + \psi_i$. By using the absolute deadline of each job as its PP, we see that G-EDF is a window-constrained scheduling algorithm.

In order to model restricted supply, Leontyev and Anderson defined a *service function* (following from Chakraborty et al. (2003)) $\beta_p(\Delta)$ for each CPU p, indicating that in any interval of length Δ, at least $\beta_p(\Delta)$ units of time on CPU p are available to execute tasks. The form of the service functions used by Leontyev and Anderson is depicted Figure 2.1. Each CPU p has an *available utilization* $\widehat{u_p}$ and a *blackout time* σ_p, so that

$$\beta_p(\Delta) \triangleq \max\{0, \widehat{u_k} \cdot (\Delta - \sigma_p)\}.$$

In Figure 2.1, we assume that the same pattern of supply restriction continues indefinitely; in this case, $\widehat{u_p} = \frac{1}{2}$ and $\sigma_p = 3$. $\widehat{u_p}$ indicates the long-term utilization of processor p, and in Figure 2.1, half of the CPU time is occupied by supply restriction. σ_p is set to the x-intercept necessary in order for $\beta_p(\Delta)$ to lower-bound the actual supply, when the slope of $\beta_p(\Delta)$ is $\widehat{u_p}$.

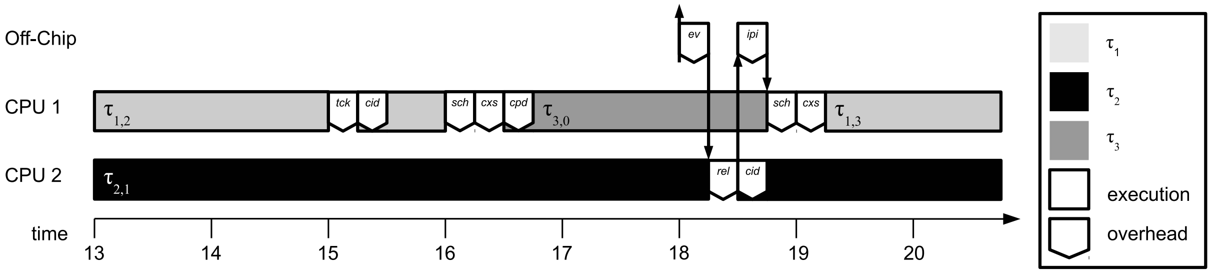


Figure 2.2: A subset of the schedule from Figure 1.10(a) with overheads included. The execution times have been slightly reduced to make room for overheads.

The proof structure used by Leontyev and Anderson is similar to that used by Devi and Anderson, but much additional complexity is added by the generalizations applied.

2.2 Overhead and Locking Analysis

In order to determine the schedulability of a task system in practice, it is necessary to determine relevant system overheads and to account for them in the analysis. Brandenburg (2011) proposed methods to adjust task system parameters to account for overheads under G-EDF. These methods can be applied to arbitrary GEL schedulers. If a locking protocol is used, a similar process of adjusting parameters is generally necessary to account for similar effects that result from the operation of the locking protocol. In this section, we review these accounting methods. We will first consider overhead accounting under GEL schedulers in the absence of locking.

2.2.1 Overhead Analysis for GEL Schedulers Without Locking

In this section, we discuss overhead analysis for GEL schedulers without locking. Consider Figure 2.2, which depicts a subset of the schedule in Figure 1.10(a) with some additional overheads included.

1. From the time when an event triggering a release (e.g., a timer firing) occurs until the time that the corresponding interrupt is received by the OS, there is *event latency*, denoted *ev* (at time 18) in Figure 2.2.

2. When the interrupt is handled, the scheduler must perform release accounting and may assign the released job to a CPU. This delay is referred to as *release overhead*, denoted *rel* (after time 18) in Figure 2.2.

3. If the job is to be executed on a CPU other than the one that ran the scheduler, then an *inter-processor interrupt* (*IPI*) must be sent. In this case, the job will be delayed by the *IPI latency* of the system, denoted *ipi* (after time 18) in Figure 2.2.

4. The scheduler within the OS must run when the IPI arrives (before time 19), creating *scheduling overhead*, denoted *sch* (before time 19) in Figure 2.2.

5. After the scheduling decision is made, a context switch must be performed, creating *context switch overhead*, denoted *cxs* (at time 19) in Figure 2.2.

Observe from Figure 1.10(a) that $\tau_{3,0}$ had previously been preempted by $\tau_{2,1}$ at time 12. As a result of this earlier preemption, it experiences three additional costs when it is scheduled again after time 16.

1. When $\tau_{3,0}$ is scheduled again (time 16), it incurs another scheduling overhead *sch* and context switch overhead *cxs*.

2. Because $\tau_{3,0}$ was preempted, some of its cached data items and instructions may have been evicted from caches by the time it is scheduled again. As a result, $\tau_{3,0}$ will require extra execution time in order to repopulate caches. Although not depicted in Figure 2.2, observe from Figure 1.10(a) that $\tau_{3,0}$ has also migrated from another processor, which may have caused even greater cache effects than if it had resumed on the same CPU. The added time to repopulate caches is called *cache-related preemption and migration delay* (*CPMD*) and is denoted *cpd* (before time 17) in Figure 2.2.

Another overhead that occurs is the presence of interrupts, both from the periodic timer tick and from job releases. The maximum time for the timer tick interrupt service request routine is denoted *tck* in Figure 2.2 (time 15), and the maximum cache-related delay from an interrupt is denoted *cid* in Figure 2.2 (after time 15).

Having discussed the nature of the overheads, we now give the precise expressions used by Brandenburg (2011) to perform overhead accounting. In these expressions, each overhead is represented as a superscript of Δ. For example, the event latency, denoted *ev* in Figure 2.2 as described above, is represented as Δ^{ev}. Virtual tasks are defined for interrupt sources. For the timer interrupts, we let

$$C_0^{tck} \triangleq \Delta^{tck} + \Delta^{cid}, \tag{2.1}$$

and we let T_0^{tck} be the period of the timer tick. For each task τ_i, the corresponding release interrupts are handled by denoting

$$C_i^{irq} \triangleq \Delta^{rel} + \Delta^{cid}, \tag{2.2}$$

$$T_i^{irq} \triangleq T_i. \tag{2.3}$$

We let $U_0^{tck} \triangleq C_0^{tck}/T_0^{tck}$ and $U_i^{irq} \triangleq C_i^{irq}/T_i^{irq}$ and define a parameter representing how long a CPU can be occupied by interrupts in the short term:

$$c^{pre} \triangleq \frac{C_0^{tck} + \Delta^{ev} \cdot U_0^{tck} + \sum_{1 \le j \le n} \left(\Delta^{ev} \cdot U_j^{irq} + C_j^{irq} \right)}{1 - U_0^{tck} - \sum_{1 \le j \le n} U_j^{irq}}. \tag{2.4}$$

Each τ_i (before overhead inflation) is replaced with τ_i' (after overhead inflation). Each τ_i in τ has its C_i and T_i parameters redefined as follows, with the remaining parameters unchanged.

$$C_i' \triangleq \frac{C_i + 2 \cdot (\Delta^{sch} + \Delta^{cxs}) + \Delta^{cpd}}{1 - U_0^{tck} - \sum_{1 \le j \le n} U_j^{irq}} + 2 \cdot c^{pre} + \Delta^{ipi}, \tag{2.5}$$

$$T_i' \triangleq T_i - \Delta^{ev}. \tag{2.6}$$

We sometimes dedicate one processor, called a *release master*, to handling interrupts. If we use a release master, then the only job that can be delayed by a release interrupt is that job itself. Therefore, the following modifications are made.

$$c^{pre} \triangleq \frac{C_0^{tck} + \Delta^{ev} \cdot U_0^{tck}}{1 - U_0^{tck}}, \tag{2.7}$$

$$C_i' \triangleq \frac{C_i + 2 \cdot (\Delta^{sch} + \Delta^{cxs}) + \Delta^{cpd}}{1 - U_0^{tck}} + 2 \cdot c^{pre} + \Delta^{ipi} + \Delta^{rel}. \tag{2.8}$$

After these modifications are made to the task system, analysis that assumes the absence of overheads can then be applied. The soundness of this analysis is established in (Brandenburg, 2011), to which we refer the reader for details.

In the presence of a locking protocol, additional modifications to the task system are necessary. We next describe the modifications under the *mutex queue spinlock* protocol considered in Chapter 5.

2.2.2 Mutex Queue Spinlock Protocol Accounting

Under the mutex queue spinlock protocol, when a job requests a resource, it first spins non-preemptively until it acquires the resource. Then, it enters a *critical section* during which it uses the resource. It continues to execute non-preemptively until the critical section is over, at which time it releases the resource for other jobs to use.

In the presence of a locking protocol, Brandenburg (2011) describes two forms of blocking that can prevent a job from running, even if it has sufficiently high priority. When a task is spinning waiting for a resource (because another job is in a critical section using that resource), it incurs *s-blocking*. When a job cannot execute because it cannot preempt a lower-priority job that is running non-preemptively, it incurs *pi-blocking*. Both types of blocking must be bounded. Here we provide only the results from (Brandenburg, 2011), referring the reader to that work for full justification.

Brandenburg considered clustered scheduling, in which tasks are partitioned onto clusters of CPUs, and global scheduling is used within each cluster. The number of CPUs in cluster v is denoted c_v, and the set of tasks in cluster v is denoted θ_v.

Resource Model. We consider a set $\{\psi_1, \psi_2, \ldots, \psi_{n_r}\}$ of n_r *resources*. Each job can request only one resource at a time, but can request the same resource multiple times. We denote as $N_{i,q}$ the maximum number of requests for ψ_q that any $\tau_{i,k}$ can issue, and as $L_{i,q}$ the maximum length of such a request.

Definitions Used in Analysis. We let R_i denote a response-time bound for any job of τ_i. Job response times can be affected by the amount of blocking a job experiences, even though R_i must be used while analyzing blocking. As a result, blocking terms should be computed iteratively. Initially, each R_i can be computed while ignoring blocking. The newly computed blocking can then be used to compute new values for R_i. This process is repeated until each R_i no longer grows larger.

Brandenburg defines the *request interference bound* for a task τ_i, with respect to resource ψ_q and interval length t, as a set of $N_{i,q} \cdot \left\lceil \frac{t+R_i}{T_i} \right\rceil$ requests, each of length $L_{i,q}$. The request interference bound is denoted $\mathit{tif}(\tau_i, \psi_q, t)$. He also defines, for a request set S, $\mathit{top}(l, S)$ as the l largest requests, and $\mathit{total}(l, S)$ as the sum of the lengths of all requests in $\mathit{top}(l, S)$. In addition, the *aggregate interference bound* of a set of tasks θ, with respect to resource ψ_q, interval length t, and *interference limit* l, is the set of all requests in $\mathit{top}(l, \mathit{tif}(\tau_i, \psi_q, t))$ for any $\tau_i \in \theta$. The aggregate interference bound is denoted $\mathit{tifs}(\theta, \psi_q, t, l)$.

S-Blocking Analysis. Brandenburg shows that the greatest s-blocking that an arbitrary $\tau_{i,k}$ can incur due to requests for ψ_q issued by jobs of tasks assigned to cluster v is

$$spin(\tau_i, v, \psi_q) = \begin{cases} total(N_{i,q} \cdot c_v, tifs(\theta_v, \psi_q, R_i, N_{i,q})) & \text{if cluster } v \text{ is not } \tau_i\text{'s cluster,} \\ total(N_{i,q} \cdot (c_v - 1), tifs(\theta_v \setminus \{\tau_i\}, \psi_q, R_i, N_{i,q})) & \text{if cluster } v \text{ is } \tau_i\text{'s cluster.} \end{cases}$$

The total s-blocking that arbitrary $\tau_{i,k}$ can incur is computed by summing $spin(\tau_i, v, \psi_q)$ over all clusters and resources.

PI-Blocking Analysis. Recall that $\tau_{i,k}$ is "pi-blocked" if a lower-priority job is executing non-preemptively, preventing $\tau_{i,k}$ from beginning execution.

Brandenburg defines $lower(\tau_i)$ as the set of tasks within the same cluster as τ_i that could cause some $\tau_{i,k}$ to incur pi-blocking. For a fixed-priority scheduler, $lower(\tau_i)$ is simply the set of tasks that have lower priority than τ_i. For a GEL scheduler, $lower(\tau_i)$ is the set of tasks τ_j with $Y_j > Y_i$. This is because if $\tau_{j,\ell}$ blocks $\tau_{i,k}$ at $r_{i,k}$, then $r_{j,\ell} < r_{i,k}$ (as $\tau_{j,\ell}$ is already running). Thus, if $Y_j \leq Y_i$, then by the definition of Y_i in Section 1.3, $y_{j,\ell} \leq y_{i,k}$, and $\tau_{j,\ell}$ does not have lower priority.

A job executes non-preemptively only during an interval from the request of a resource until the completion of the corresponding critical section. Therefore, bounding the length of such an interval for any job of any task in $lower(\tau_i)$ is sufficient to bound the pi-blocking incurred by any job of τ_i.

Let $\tau_{j,\ell}$ be an arbitrary job that causes $\tau_{i,k}$ to incur pi-blocking. Brandenburg demonstrates that the maximum amount of time that $\tau_{j,\ell}$ may be blocked waiting for resource ψ_q due to jobs in cluster v is at most

$$spin'_i(\tau_j, v, \psi_q) = \begin{cases} total(c_v, tifs(\theta_v, \psi_q, R_j, 1)) & \text{if cluster } v \text{ is not } \tau_j\text{'s cluster,} \\ total(c_v - 1, tifs(\theta_v \setminus \{\tau_i, \tau_j\}, \psi_q, R_j, 1)) & \text{if cluster } v \text{ is } \tau_j\text{'s cluster.} \end{cases} \tag{2.9}$$

For a given resource, summing this expression over all clusters and adding $L_{j,q}$ yields the maximum time that $\tau_{j,\ell}$ can execute non-preemptively. Note that ℓ is arbitrary, so this bound can be considered a bound for τ_j, not just for $\tau_{j,\ell}$. Selecting the maximum such bound for all resources and tasks in $lower(\tau_i)$, ignoring combinations of τ_j and ψ_q such that τ_j does not use ψ_q, yields the maximum pi-blocking that $\tau_{i,k}$ can incur.

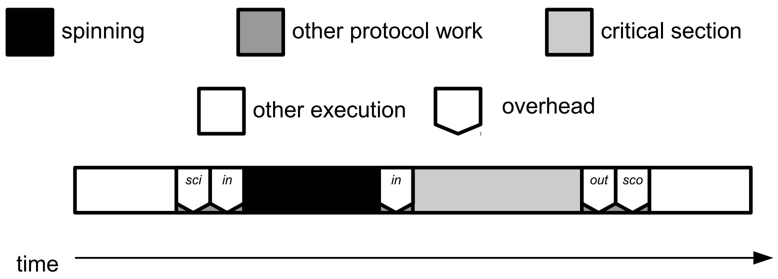


Figure 2.3: Illustration of overheads that occur in a spinlock protocol.

Accounting Method. In order to account for blocking, each task's WCET should be increased by the sum of the maximum s-blocking it may incur and the maximum pi-blocking it can incur. Then, analysis methods that assume the absence of a locking protocol can be used.

We next discuss how to account for overheads related to this locking protocol.

2.2.3 Mutex Queue Spinlock Protocol Overheads

We now consider the overheads resulting from the mutex queue spinlock protocol. Brandenburg's analysis assumes the presence of a release master.

When $\tau_{i,k}$ attempts to access ψ_q, execution proceeds as follows, and as depicted in Figure 2.3.

1. $\tau_{i,k}$ begins to execute non-preemptively by issuing a system call. The related overhead is denoted *sci* in Figure 2.3.

2. $\tau_{i,k}$ uses the locking protocol and attempts to acquire ψ_q. If ψ_q is held by another job, then $\tau_{i,k}$ must spin. The time spent spinning is accounted for using the techniques presented in Section 2.2.2, so we do not consider that time in overhead accounting. However, the cumulative time spent running other parts of the locking protocol is denoted *in* in Figure 2.3. Note that *in* appears both before and after the spinning in Figure 2.3. The overhead term Δ^{in} would refer to the *sum* of the two occurrences.

3. $\tau_{i,k}$ actually executes its critical section. Brandenburg's analysis assumes that no overheads affect critical sections, which is why a release master is assumed.

4. $\tau_{i,k}$ uses the locking protocol to release ψ_q. The related overhead is denoted *out* in Figure 2.3.

5. $\tau_{i,k}$ then begins to execute preemptively by issuing a system call, with the overhead denoted *sco* in Figure 2.3.

Accounting for overheads within a spinlock protocol is a three-step process. The first step is to apply the overheads due to the spinlock protocol, without accounting for other overheads. Note that the overheads *in* and *out* may delay other jobs waiting to acquire ψ_q, causing them to spin for longer times. Therefore, we must add $\Delta^{in} + \Delta^{out}$ to each request length $L_{i,q}$, so that the blocking analysis from Section 2.2.2 correctly bounds the spinning of other tasks. Additionally, during the first step we also account for the extra execution that each $\tau_{i,k}$ experiences due to locking overheads. Each request produces $(\Delta^{sci} + \Delta^{sco} + \Delta^{in} + \Delta^{out})$ units of overhead, and the total number of requests can be computed by summing $N_{i,q}$ over all resources. We adjust C_i accordingly.

During the second step, we apply the lock accounting from Section 2.2.2 to the modified task system, inflating the worst-case execution times of all tasks that use resources.

During the final step, we apply the locking-agnostic overhead accounting from Section 2.2.1 to obtain the final values of C_i and T_i for each task.

Recall that, as discussed in Section 2.2.2, the lock accounting requires a response-time bound R_i for each τ_i, and that the accounting may need to be performed iteratively if it increases R_i for some τ_i. When performing overhead-aware analysis, the second and final steps may need to be repeated iteratively if together they cause some R_i to increase.

2.3 SRT Models

Although this dissertation primarily considers the "bounded lateness" definition of SRT, much past research has been performed describing and using other definitions. Except where noted, these papers consider uniprocessors and the implicit-deadline *periodic* task model, where tasks have exact rather than minimum separation times.

Koren and Shasha (1995a) allowed each task to have a *skip factor s*: each time a job of that task misses a deadline, the next $s-1$ jobs must complete. The scheduler can simply skip any task that would miss a deadline, so some task sets with total utilization larger than one can be scheduled. However, Koren and Shasha showed that even on a uniprocessor, optimal scheduling with their model is NP-hard. Hamdaoui and Ramanathan (1995) considered the more general (h,k) model.[1] In that model, h jobs of a task must meet their deadlines out of any consecutive k jobs of that task. Both of these types of constraints are generalized

[1] We have changed their notation slightly to avoid conflict with other terms.

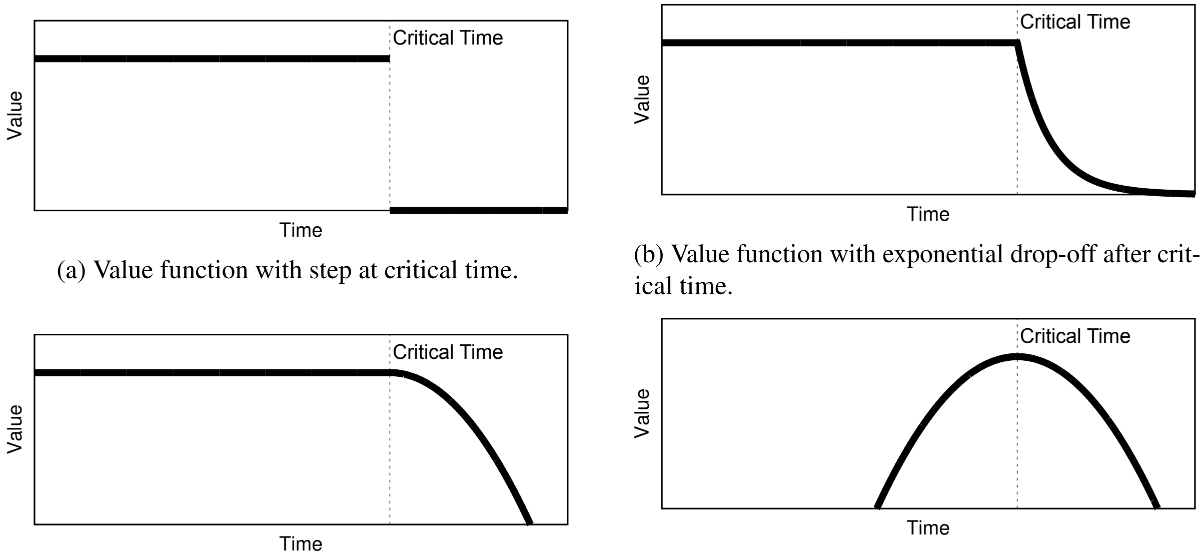
as *weakly hard* constraints by Bernat et al. (2001). They defined a “weakly hard real-time system” as any system with a precise bound on the distribution of met and missed deadlines. (Normal HRT systems are a special case, where every deadline is met.) Bernat et al. described a few variants, which can be combined with logical operators:

- A task can “meet any h in k deadlines,” which is identical to the (h,k) model discussed above.
- A task can “meet row h in k deadlines”, meaning that it must meet h deadlines in a row in every window of k deadlines. If $k = h + 1$, this scheme reduces to a skip factor of h.
- A task can “miss any h in k deadlines,” meaning that it cannot miss more than h deadlines in a window of k.
- A task can “miss row h in k deadlines,” meaning that it cannot miss more than h deadlines in a row in a window of k. (The window size k is not actually required to express this condition.)

A weaker form of the (h,k) model, the *window-constrained* task model, was described by West and Poellabauer (2000). In that model, h jobs must meet their deadlines within periodic *fixed* windows of k jobs of that task. (Any task system that is schedulable using the (h,k) model is also schedulable using the window-constrained model.)

Lin and Natarajan (1988) proposed the *imprecise computation model* for tasks that compute numerical results. Under that model, each job has a mandatory part that must complete before its deadline under any circumstances, and an optional part that can be interrupted at any time. The mandatory part guarantees an approximate solution, and the precision of the solution must be non-decreasing as the optional part executes. The task must be defined to conform to these requirements. Ideally, every task would run its optional part to completion, but part of that computation can be cancelled when that is not possible. This model is not sufficient to provide a well-defined scheduling problem, because there must be some mechanism to determine which optional parts to execute. Several potential strategies, such as minimizing the number of dropped optional portions or minimizing the maximum error, were discussed by Liu et al. (1991). A more precise metric and an optimal scheduling algorithm with regard to that metric were proposed by Aydin et al. (2001).

Locke (1986) assigned to each task a *value function* that specifies the value to a system of completing a job at a particular time after its release. “Value” is a unit-less quantity that can be compared between jobs, to



(a) Value function with step at critical time.

(b) Value function with exponential drop-off after critical time.

(c) Value function with quadratic drop-off after critical time.

(d) Value function with a specific target completion time, and quadratic increase and drop-off before and after that time, respectively.

Figure 2.4: Example value functions from Locke (1986).

determine which job to complete in the event of an overload. Ideally, the system should accrue as much total value as possible.

Examples of value functions are depicted in Figure 2.4. In each example, the x axis represents the completion time of a job after its release, while the y axis represents the value to the system from completing that job. For example, suppose τ_i is the task considered in Figure 2.4(a). If some $\tau_{i,k}$ completes before the time marked "Critical Time," then the system achieves some constant value. However, if the job completes after that time, the system receives no value whatsoever. Thus, the system should only execute $\tau_{i,k}$ if it is possible for $\tau_{i,k}$ to complete before the critical time. Furthermore, suppose there are two tasks τ_i and τ_j that each have value functions of this form and that at time t there are ready jobs $\tau_{i,k}$ and $\tau_{j,\ell}$. If it is possible to complete either $\tau_{i,k}$ or $\tau_{j,\ell}$ before its respective critical time, but not to complete both before their respective critical times, then $\tau_{i,k}$ should generally be selected if τ_i has a higher constant value than τ_j, and $\tau_{j,\ell}$ should be selected if the reverse is true. (This is not strictly true because the choice of $\tau_{i,k}$ or $\tau_{j,\ell}$ may affect the ability to complete other jobs at appropriate times.)

Although not required by the definition of "value function," for tractability Locke (1986) considered value functions that are continuous and have continuous first and second derivatives, except for (possibly) a single discontinuity at the *critical time*. This is why the time of the discontinuity in Figure 2.4(a) is labelled as the “critical time.” Although the step function discussed above is the simplest value function, Locke proposed others. Figure 2.4(b) depicts a value function that drops off exponentially after the critical time, indicating that there is still some value to completing jobs late, but this value rapidly drops off as jobs complete later. Figure 2.4(c) depicts a value function that drops off more slowly after the critical time, indicating that completing jobs slightly late has a smaller impact than for the value function in Figure 2.4(b). Finally, as depicted in Figure 2.4(d), it is also possible to use value functions to indicate that a job should not complete too *early*. In this case, a job that finishes very quickly will achieve zero value, just as if the job finished very late.

Because value functions are used primarily for overload management, scheduling with value functions is discussed in the next section.

Buttazzo and Stankovic (1995) used a closely related notion of value. They divided tasks into “hard” and “critical” categories. Tasks of both types are expected to meet their deadlines when the system is not overloaded, and critical tasks are only allowed to miss deadlines when the system is overloaded even considering only critical tasks. Furthermore, a task can be accepted or rejected. Once a critical task has been accepted it is always guaranteed to complete by its deadline, even during severe overload. In addition, each task can have a *deadline tolerance*, which is an allowable amount of tardiness. Any job that will not complete within its task’s deadline tolerance should be rejected and not run at all. The scheduling algorithm they proposed is discussed in Section 2.5.

2.4 Overload Management Using Value Functions

In this section, we discuss prior work on scheduling algorithms that use value functions to define correct behavior during overload.

2.4.1 Locke’s Best-Effort Heuristic

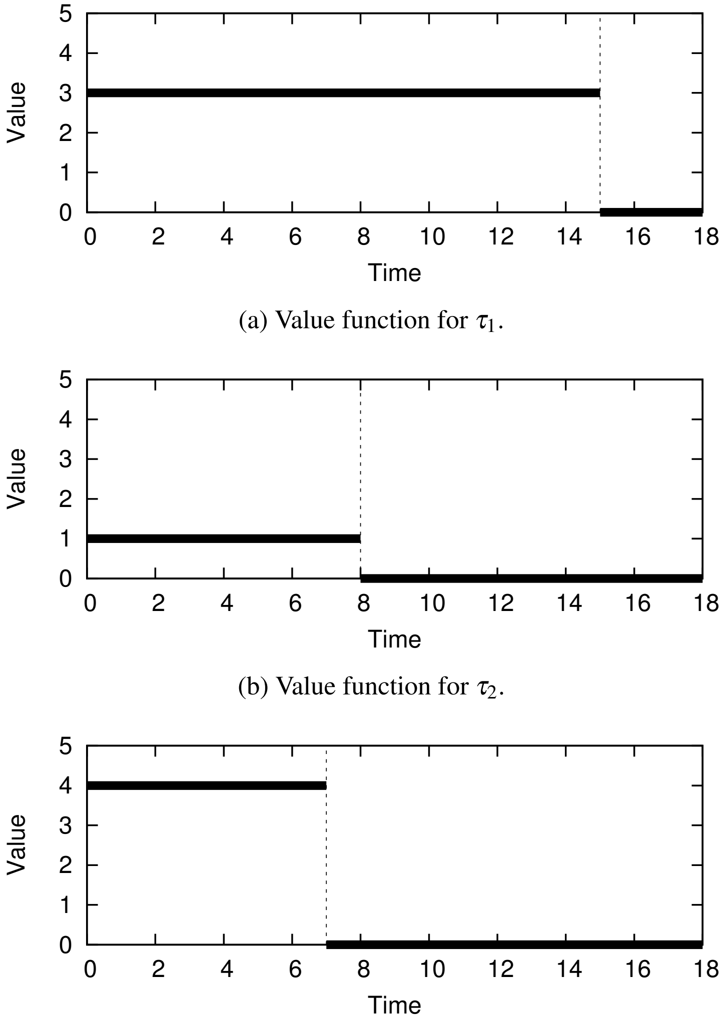
Locke (1986) considered a system that resembles the sporadic task model, scheduled globally on a multiprocessor. However, rather than having a per-task *upper bound* on job execution times, there is a stochastic per-task *distribution* of execution times. Similarly, rather than having a per-task *lower bound* on separation time between job releases, job releases follow a stochastic per-task distribution. Given these modifications to the task system, it is possible for the system to experience overload if there is a burst of jobs that either are released closer together than generally expected, or that run for longer times than generally expected.

As discussed in Section 2.3, the system should try to achieve the maximum cumulative value even if such an overload occurs. However, there are two difficulties that arise in attempting to do so: uncertainty about system behavior and the intractability of the scheduling problem.

Locke assumed that the system does not know the timing of job releases until they occur and does not know the actual run time of each job until it completes. We show by example that even the first assumption is itself sufficient to prevent the system from always maximizing the cumulative value. Consider the task system with value functions depicted in Figure 2.5, as scheduled on a uniprocessor. Suppose that τ_1 releases $\tau_{1,0}$ and τ_2 releases $\tau_{2,0}$ at time 0 and that no other job is released before time 15. Further suppose that $\tau_{1,0}$ is known to require 14 ms of execution, while the job of $\tau_{2,0}$ is known to require 7 ms of execution. This scenario is depicted in Figure 2.6(b)–(c). Because these two jobs together require 21 ms of execution, while their last critical time is at time 15, the system cannot complete both jobs before their critical times. It is therefore better for it to select $\tau_{1,0}$ as in Figure 2.6(b), in order to achieve a cumulative value of three, rather than selecting $\tau_{2,0}$ as in Figure 2.6(c), which would only achieve a cumulative value of one.

Suppose, however, that τ_3 actually releases $\tau_{3,0}$ at time 8, and that $\tau_{3,0}$ is known to require 6 ms of execution. Because $\tau_{3,0}$ can complete with a value of 4, which is greater than the value that can be achieved by either $\tau_{1,0}$ or $\tau_{2,0}$, the system should execute $\tau_{3,0}$ to maximize the cumulative value. If the system initially chose to execute $\tau_{1,0}$, as depicted in Figure 2.6(d), then because $\tau_{1,0}$ does not actually complete, the cumulative value achieved is only four. However, if the system initially chose to execute $\tau_{2,0}$, as depicted in Figure 2.6(e), then because $\tau_{1,0}$ completes, the cumulative value achieved is five. Therefore, *the optimal choice at time* 0 *depends on whether* $\tau_{3,0}$ *is released at time* 8, so making an optimal choice is impossible under Locke's assumptions. A similar example could be constructed if the execution times were unknown.

Locke also noted that, even if optimal decision-making were possible, the problem is likely to be NP-complete. However, the results of this decision-making process are most important *precisely when the system is already overloaded and cannot complete all jobs*. Furthermore, *running the scheduling algorithm requires the same computing resource as the jobs themselves*. Therefore, running an optimal scheduling algorithm would cause more harm than it prevents. Thus, Locke could have chosen to develop either a heuristic algorithm or an approximation algorithm with a corresponding provable bound on achieved value, but he chose the former.

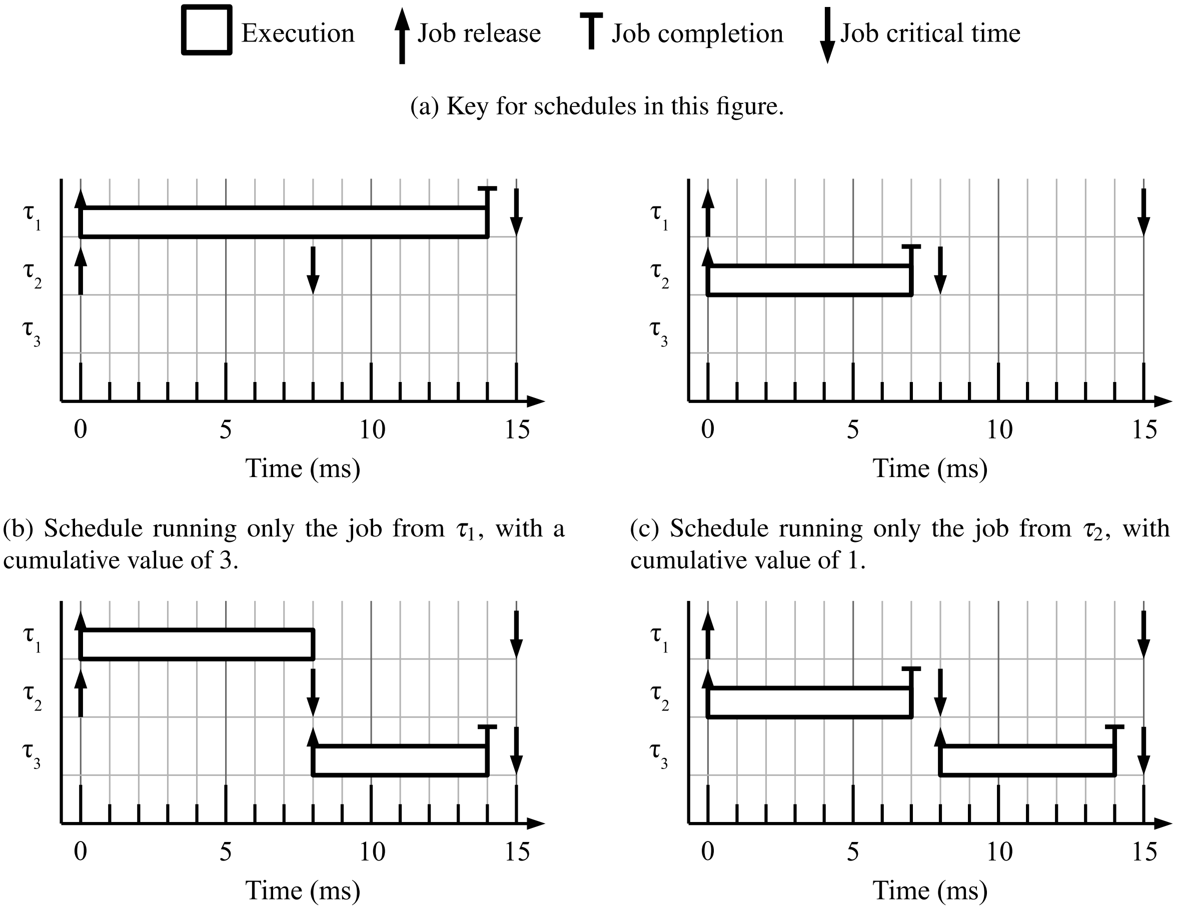


(a) Value function for τ_1.

(b) Value function for τ_2.

(c) Value function for τ_3.

Figure 2.5: Value functions for an example task system.



(a) Key for schedules in this figure.

(b) Schedule running only the job from τ_1, with a cumulative value of 3.

(c) Schedule running only the job from τ_2, with a cumulative value of 1.

(d) Schedule running jobs from τ_1 (though not to completion) and τ_3, with a cumulative value of 5.

(e) Schedule running jobs from τ_1 and τ_3, with a cumulative value of 4.

Figure 2.6: Several possible schedules for the task system with value functions depicted in Figure 2.5. $\tau_{1,0}$ is released at time 0 and is known to have an execution time of 14. $\tau_{2,0}$ is also released at time 0 and is known to have an execution time of 7. In (d) and (e), $\tau_{3,0}$ is released at time 8 and is known to have an execution time of 6.

Locke's heuristic algorithm is based on the assumption that, under typical circumstances, it will be possible for nearly every job to complete at a time that allows it to achieve nearly all of its possible value. This assumption simply means that the system was properly provisioned for the common case. In order to exploit this assumption, Locke assigned for each job a *deadline* that is the latest time it can complete while continuing to achieve a user-configurable fraction of its maximum achievable value. In the case of a step value function, as in Figure 2.4(a), a job's deadline is simply its critical time. However, under any of the other types of value functions depicted in Figure 2.4, a job's deadline is usually after its critical time. His system simply prioritizes all jobs by deadline until the probability of a deadline miss exceeds a user-configurable threshold.

Once a deadline miss is likely, the system switches prioritization to a heuristic based on *value density*. The value density for $\tau_{i,k}$ at time t is computed as follows. Let $e^r_{i,k}(t)$ be the expected remaining execution time for $\tau_{i,k}$ at time t, conditioned on how long it has already executed. The expected value $V(t)$ of $\tau_{i,k}$ at time t is defined to be the value that $\tau_{i,k}$ will accumulate if it completes at time $t + e^r_{i,k}(t)$. The value density of $\tau_{i,k}$ at time t is simply $V(t)/e^r_{i,k}(t)$. For example, consider the schedule in Figure 2.6(d), At time $t = 0$, $\tau_{1,0}$ is expected to have 14 units of execution remaining, completing at time 14, and has an expected value of three, resulting in a value density of $3/14$. At time $t = 8$, $\tau_{1,0}$ is expected to have $14 - 8 = 6$ units of execution remaining, still completing at time 14, and has an expected value of three. Its value density is now $3/6 = 1/2$.

The heuristic that the system uses during overload, when it is likely that some job will miss its deadline, is to prioritize jobs by decreasing value density. This heuristic is actually illustrated in Figure 2.6(d). As discussed above, at time $t = 0$, the value density of $\tau_{1,0}$ is $3/14$. Similarly, the value density of $\tau_{2,0}$ at $t = 0$ is $1/(7-0) = 1/7 < 3/14$, so $\tau_{1,0}$ is selected for execution. At time $t = 8$, as discussed above, $\tau_{1,0}$ now has a value density of $1/2$. However, $\tau_{3,0}$ then has a value density of $4/(14-8) = 2/3 > 1/2$, so $\tau_{3,0}$ is selected in place of $\tau_{1,0}$.

Another detail in Locke's heuristic relates to jobs with value functions of the form depicted in Figure 2.4(d), where completing a job too early can result in smaller achieved value. Because a job's value is based only on its *completion* time, it is possible to start running a job long before its critical time, as long as its execution is discontinued when it approaches completion. However, because job execution times are determined stochastically, there is a risk that the job will complete earlier than expected. Locke's system assumes that a job will not complete if it has executed for less than some user-specified number of standard deviations below its expected execution time. After it has executed for this amount of time, an incomplete job will be suspended until closer to its desired completion time.

Locke demonstrated the effectiveness of his heuristic through experiments that simulate global multiprocessor schedules where one CPU is dedicated to making scheduling decisions for the rest of the system. He demonstrated that his scheme provides significantly higher achieved value than other considered schedulers in the presence of overload, while also meeting most deadlines in the absence of overload. However, he did not provide any theoretical guarantees comparing the achieved value to the maximum possible achieved value.

2.4.2 D*

Unless otherwise noted, all papers discussed in the remainder of Section 2.4 consider only step value functions, as depicted in Figure 2.4(a). In such cases, we say that the *deadline* of each job is simply its critical time, and that its *value* is the value that it achieves if it completes before its critical time. Furthermore, the job's *value density* is simply its value divided by its total execution time. (This differs from the notion of "value density" used by Locke (1986), who used remaining execution time.) We use the constant q to denote the *importance ratio*, or the ratio of the largest value density in the system to the smallest value density in the system.[2]

Baruah et al. (1991) considered scheduling on uniprocessors. They observed that Locke (1986) provided only heuristics, but did not provide any guarantee about the value that could be achieved during an overload. In order to provide such a guarantee, they developed a new scheduling algorithm, D*. They assumed that job release times are not known ahead of time, but that the exact execution time of each job is known upon release.

D* is similar to the later-proposed *earliest deadline until zero laxity* (*EDZL*) scheduling algorithm (Baker et al., 2008; Lee, 1994). A job's *laxity* is the time until its deadline minus its remaining execution time. If it reaches a *zero-laxity* state, then it must be scheduled immediately, or it will miss its deadline. Like EDZL, D* behaves identically to EDF until some $\tau_{i,k}$ reaches a zero-laxity state. If no other job is in a zero-laxity state when this occurs, then $\tau_{i,k}$ runs immediately. To handle the case when some $\tau_{j,\ell}$ is already in a zero-laxity state, D* maintains the sum of the values of all jobs that have been preempted in a zero-laxity state since the last successful job completion. Such preempted jobs have been abandoned, as it was impossible for them to

[2] Several of the papers cited herein use k for the importance ratio, but we use q ($q$uotient) to avoid conflict with the job index k.

meet their deadlines. If $\tau_{i,k}$ has a value greater than this sum plus the value of $\tau_{j,\ell}$, then $\tau_{i,k}$ preempts $\tau_{j,\ell}$, $\tau_{j,\ell}$ is abandoned, and the sum is updated. Otherwise, $\tau_{i,k}$ is abandoned.

Baruah et al. (1991) proved that, if value densities are normalized such that the smallest is at least one, the total value achieved during an overloaded interval using D* is at least $1/5$ the *length* of that interval. Although this value is small, they proved that no algorithm can guarantee more than $1/4$ of the length of such an interval without knowing job releases ahead of time. Baruah et al. also showed experimentally that D* performs similarly to Locke's best effort scheduler in the common case, but provides drastically better behavior in certain pessimistic cases.

2.4.3 D^{over}

Koren and Shasha (1995b) provided a scheduler D^{over} that can guarantee an achieved value of $1/4$ of the length of an overloaded interval, closing the gap between D* and the theoretical limit. The design of D^{over}, like the design of D*, is based on EDF and is identical to EDF until an overload occurs.

Even during underload, D^{over} maintains two sets of ready jobs, not including the currently running job: *waiting* jobs and *privileged* jobs. If a job is preempted by a normal job release, then it becomes a privileged job. The system keeps track of the amount of time that a newly arriving job can execute without causing the current job or any privileged job to miss its deadline. When a new job arrives, if its execution cost is less than this time, it preempts the current job. Otherwise, because adding the new job could cause some existing job to miss its deadline, an overload has occurred. Therefore, the new job is instead added to the queue of waiting jobs. This strategy ensures that a privileged job can never reach a zero-laxity state.

Let $V_{j,\ell}$ denote the value of $\tau_{j,\ell}$. When waiting $\tau_{i,k}$ reaches a zero-laxity state, its value is compared to $(1-\sqrt{q})\cdot(\sum_{\tau_{i,k}\in\Theta} V_{j,\ell})$, where Θ is the set containing all privileged jobs and the currently running job. If its value is larger than this expression, then $\tau_{i,k}$ preempts the currently running job, and all privileged jobs become waiting jobs. Otherwise, $\tau_{i,k}$ is discarded.

Koren and Shasha (1995b) also demonstrated that D^{over} achieves the optimal *competitive ratio* of $\frac{1}{(1+\sqrt{q})^2}$. In other words, D^{over} is guaranteed to achieve at least $\frac{1}{(1+\sqrt{q})^2}$ times as much value in an overloaded interval as could be achieved by a clairvoyant algorithm. Baruah et al. (1991) demonstrated that no better competitive ratio is possible.

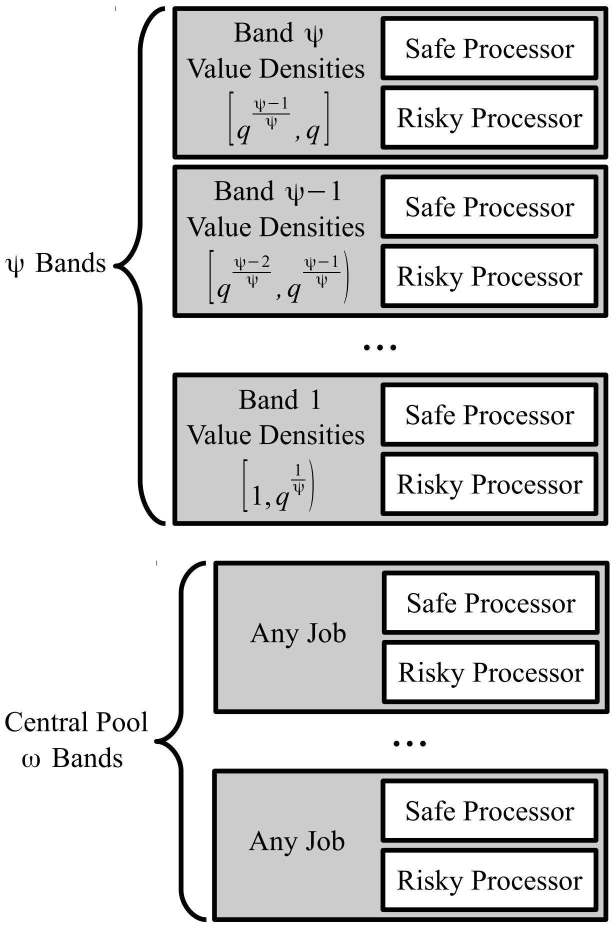


Figure 2.7: Grouping of CPUs used by MOCA.

2.4.4 MOCA

While D* and D^{over} provide guarantees on a *uniprocessor*, Koren and Shasha (1994) proposed the *multiprocessor on-line competitive algorithm* (*MOCA*) to provide such guarantees on a *multiprocessor*. MOCA requires an even number of processors and works by dividing the system into $m/2$ *bands* of two processors, as depicted in Figure 2.7. ψ of the bands are designated with specific value densities, and ω form a *central pool*. It must be the case that $\psi + \omega = m/2$, so that each CPU is assigned to exactly one band. Each band contains a *safe processor* for executing jobs that can be guaranteed to meet their deadlines and a *risky processor* for executing jobs for which such a guarantee cannot be made.

When a job is released, the system first tries to assign it to the band designated for its value density. If it can be assigned to the safe processor without compromising the guarantees made to other jobs that are already on that safe processor, the system assigns it there. Otherwise, the system tries to assign it to the safe processor for a band designated for lower value density, considering such bands in decreasing value density order. If even that fails, the system then tries to assign it to a safe processor in the central pool, considering such processors in arbitrary order. If all else fails, the system adds it to a list of *waiting* jobs and does not consider it until it reaches a zero-laxity state.

When a waiting job $\tau_{i,k}$ reaches a zero-laxity state, the system tries to schedule it on a risky processor. It first looks for an idle risky processor, considering first that in the band designated for $\tau_{i,k}$'s value density, followed by those in bands designated for lower value densities in decreasing order, then by those in the central pool. If it finds an idle risky processor, it begins executing $\tau_{i,k}$ there. Otherwise, it considers the same set of risky processors as before and finds the one running the $\tau_{j,\ell}$ with the earliest deadline. If $\tau_{i,k}$ has a *later* deadline than $\tau_{j,\ell}$, then $\tau_{j,\ell}$ is abandoned, and $\tau_{i,k}$ begins running in its place. Otherwise, $\tau_{i,k}$ is abandoned. This heuristic is used to minimize the risk of unnecessary idleness on a risky processor, as a job in a zero-laxity state will run continuously until its deadline.

Whenever some safe processor becomes idle, the safe and risky processors within that band switch roles. This guarantees that the job running on the risky processor will complete (as it is now on a safe processor) and ensures that an idle risky processor is now ready to schedule a waiting job that reaches a zero-laxity state.

Koren and Shasha (1994) also showed that no scheduler executing on m processors can achieve a competitive ratio above

$$\frac{q-1}{qm \cdot (q^{\frac{1}{m}}-1)},$$

and that MOCA achieves a competitive ratio of

$$\frac{1}{1+2m \cdot \left(\max_{1 \le i \le \psi} \frac{q^{\frac{i}{\psi}}}{\omega + \frac{q^{\frac{i}{\psi}}-1}{q^{\frac{1}{\psi}}-1}} \right)}.$$

ψ should be chosen to maximize this ratio. Observe that MOCA is not necessarily optimal in the sense of achieving the best possible competitive ratio. However, unlike heuristic approaches, it does provide a guarantee.

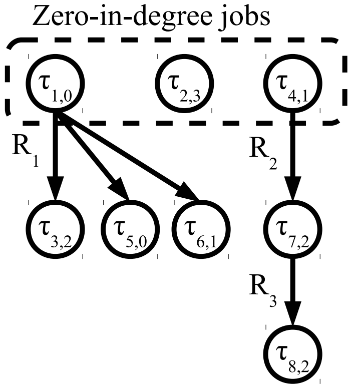


Figure 2.8: Example job dependency DAG to illustrate the work of Garyali (2010).

2.4.5 Schedulers Accounting for Dependencies

Some work has been performed on scheduling with value functions in the presence of dependencies such as shared resources (Cho, 2006; Clark, 1990; Garyali, 2010; Li, 2004; Li et al., 2006). Such dependencies are not considered in our work on overload management, so we do not review most of these works in detail. However, Garyali (2010) proposed two multiprocessor heuristics that may be useful even without such dependencies, so we discuss his work as a representative example.

Both of Garyali's heuristics assume that each task has a parameter specifying the expected execution time of each of its jobs and another parameter specifying the exact period between releases. In order to handle dependencies and shared resources, both heuristics maintain a *directed acyclic graph* (*DAG*) representing dependencies among jobs.

An example DAG is depicted in Figure 2.8. Resource i is identified as R_i. In this example, $\tau_{1,0}$ is currently holding the lock for R_1, and $\tau_{3,2}$, $\tau_{5,0}$, and $\tau_{6,1}$ are waiting to acquire that lock. $\tau_{2,3}$ is not holding any lock that has been requested by any other job. $\tau_{4,1}$ holds the lock for R_2. $\tau_{7,2}$ is waiting to acquire the lock for R_2, but holds the lock for R_3. Finally, $\tau_{8,2}$ is waiting to acquire the lock for R_3.

Whenever a job requests, acquires, or releases a resource, the DAG is updated. If a cycle is created, causing the "acyclic" property of the graph to be violated, then a *deadlock* has occurred. Such a situation implies that the jobs involved in the cycle cannot make progress. Both heuristics proposed by Garyali choose to abort one of the involved jobs in order to resolve the deadlock.

If a job has no incoming edges in the DAG, then it already holds any resources it requires to run. Such a job is called a *zero-in-degree* job.[3] In Figure 2.8, $\tau_{1,0}$, $\tau_{2,3}$, and $\tau_{4,1}$ are zero-in-degree jobs. Both of Garyali's heuristics consider only zero-in-degree jobs for scheduling, as other jobs cannot perform useful work until they acquire their requested locks.

Furthermore, both heuristics use a notion of *global value density* (*GVD*), similar to the notion of "value density" used by Locke (1986). Recall that the "value density" of $\tau_{i,k}$ at time t is the value $\tau_{i,k}$ is expected to accrue if it runs continuously until its expected remaining execution is exhausted, divided by its expected remaining execution $e^r_{i,k}(t)$. Garyali's heuristics also take into account the value achieved by completing jobs waiting on $\tau_{i,k}$—the descendants of $\tau_{i,k}$ within the DAG. Let $V_{j,\ell}$ be the value achieved if $\tau_{j,\ell}$ completes before its deadline, and Θ be the set containing $\tau_{i,k}$ and all of $\tau_{i,k}$'s descendants. The GVD of $\tau_{i,k}$ at t is $\frac{\sum_{\tau_{j,\ell}\in\Theta} V_{j,\ell}}{\sum_{\tau_{j,\ell}\in\Theta} e^r_{j,\ell}(t)}$.

The first heuristic Garyali proposed is the *non-greedy global utility accrual* (*NG-GUA*) algorithm.[4] NG-GUA attempts to create a G-EDF schedule. The deadline used for scheduling $\tau_{i,k}$ is actually the earliest deadline of $\tau_{i,k}$ itself or any descendent in the DAG, thus implementing the *priority inheritance protocol* (*PIP*) (Sha et al., 1990). Each time a scheduling decision must be made, NG-GUA simulates a G-EDF schedule of all zero-in-degree jobs. If no deadline is missed in this schedule, then NG-GUA simply makes scheduling decisions according to G-EDF. On the other hand, if some deadline is missed in that schedule, NG-GUA removes jobs from consideration in order of increasing GVD until the simulated schedule no longer misses deadlines. Jobs are prioritized from the resulting schedule.

The second heuristic Garyali proposed is the *greedy global utility accrual* (*G-GUA*) algorithm. G-GUA is "greedy" in the sense that it tries to optimize accrued value even in the absence of expected deadline misses. G-GUA works by partitioning jobs (not tasks) onto CPUs and simulating an EDF schedule (with the PIP) on each CPU. Jobs are considered in decreasing GVD order. If a job cannot be scheduled on any CPU without causing an expected deadline miss, the partitioning heuristic continues to the next job. Jobs are prioritized based on the resulting schedule. Every time a new scheduling decision must be made, the partitioning heuristic is re-executed, so it is possible that a job will be migrated to a different CPU, thus G-GUA is truly a global algorithm.

Garyali demonstrated the effectiveness of his heuristics using a real in-kernel implementation. He showed that both heuristics significantly outperform traditional approaches like G-EDF when overloads are present,

[3] Garyali (2010) referred to jobs as "phases."

[4] Garyali (2010) used the term "utility" for the same concept as "value" in (Locke, 1986).

and that G-GUA can often accrue more utility at the expense of unnecessary deadline misses in the underload case.

2.5 Rate-Based Earliest Deadline Scheduling

Buttazzo and Stankovic (1995) proposed the *robust earliest-deadline* (*RED*) scheduler that uses a similar model to value functions. Each task has an associated deadline, value, and *deadline tolerance*. If a job completes within its deadline *plus its deadline tolerance*, then it achieves its value; otherwise, it achieves no value. However, scheduling decisions are based only on deadline, without accounting for deadline tolerance.

Under RED, each task has a WCET that will not be exceeded, but the arrival pattern of jobs is not known. In this respect, its assumptions are like those used by D*, D^{over}, and MOCA.

When a job is released, it can be accepted or rejected. If it is rejected, it will not run unless slack is created in the future by jobs that underrun their WCETs. In addition to considering value, RED also divides tasks into two classes: *hard* and *critical*. If a hard job is accepted, then it must complete unless overload later occurs. If a critical job is accepted, then it must complete under all circumstances.

At runtime, RED keeps a list of all unfinished accepted jobs, both hard and critical, ordered by deadline. Whenever a new $\tau_{i,k}$ is released, RED uses the list to determine whether adding $\tau_{i,k}$ will cause a deadline miss. If it will not, $\tau_{i,k}$ is immediately accepted. Otherwise, RED will attempt to find one or more hard jobs that can be dropped. Buttazzo and Stankovic (1995) considered two different techniques to select such jobs, one that identifies the *single* job of lowest value that can be dropped (if such a job exists), and one that may drop multiple jobs if $\tau_{i,k}$ is critical. If either strategy succeeds in finding a set of hard jobs with total value less than that of $\tau_{i,k}$, and which if dropped will ensure no deadline misses for other accepted jobs, then all jobs in that set are rejected. Otherwise, $\tau_{i,k}$ is rejected. When RED rejects a job, it adds that job to a list of rejected jobs. If another job completes early, then RED will examine that list to determine whether it can then schedule a job it previously rejected.

RED always executes the job at the beginning of its list of accepted jobs, thus running the job with the earliest deadline. In the absence of overload, RED reduces simply to EDF. Buttazzo and Stankovic (1995) provided experimental evidence that RED can achieve significantly higher value than other schedulers such as EDF when an overload occurs.

Spuri et al. (1995) proposed the *robust total bandwidth* (*RTB*) scheduler, a similar scheduler to RED. RTB also supports a class of *hard periodic* tasks that are not subject to being rejected. It does so by scheduling the aperiodic tasks (i.e., the same types of tasks as the hard and critical tasks under RED) inside a *server*. A server is a budgeted container for other tasks. The server can be scheduled with EDF, using a budget to guarantee that it will not interfere with hard periodic tasks. When RTB chooses to schedule that server, it actually executes one of the aperiodic jobs running inside that server. Tasks are accepted or rejected using a similar strategy to RED.

2.6 Overload Management by Changing Execution Rates

Most of the overload management techniques surveyed thus far in this chapter have worked by dropping certain jobs. An alternative technique is to adjust the minimum separation time of a task, slowing down the rate at which it releases jobs.

Adaptive scheduling algorithms allow such a scaling of minimum separation times. Such algorithms were surveyed in detail by Block (2008). However, most of these algorithms are intended for use in systems where high variability in job execution times is expected, and minimum separation times must be decided online for that reason. We are concerned primarily with systems that are provisioned for the common case, but that need to recover from *transient* overloads.

The related problem of choosing new minimum separation times was addressed by Buttazzo et al. (2002), who proposed the *elastic model*. Under the elastic model, tasks are assigned initial and maximum periods, as well as *elasticity factors* that are used to determine the extent of "stretching" of each task. During a transient overload, minimum separation times can be determined based on elasticity factors.

One adaptive scheduling algorithm, *earliest eligible virtual deadline first* (*EEVDF*) (Stoica et al., 1996), uses a notion of *virtual time* similar to that used in Chapter 6 of this dissertation. Therefore, we provide a description of EEVDF here.

EEVDF is a *proportional share* scheduling algorithm. Each task is assigned a weight, and each task should receive a processor share that is commensurate with its weight. For example, consider the task system in Figure 2.9. The actual progression of time is graphed on the bottom axis. From time 0 to time 2, only τ_1 is present in the system. Therefore, it receives all of the CPU time. At time 2, τ_2 enters the system. Because τ_1 has a weight of 4 and τ_2 has a weight of 2, τ_1 receives twice as much processor time as τ_2. Until τ_3 arrives at

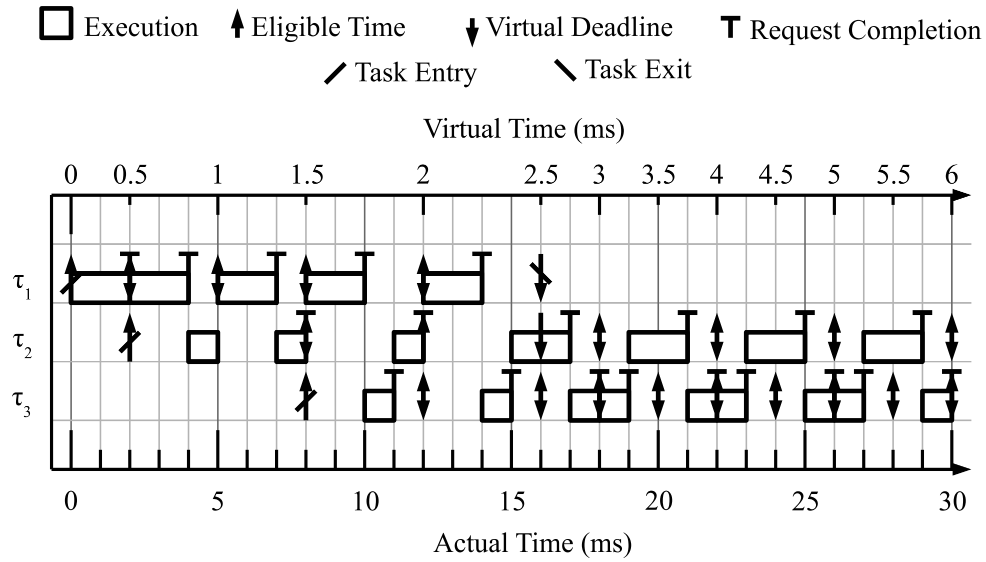


Figure 2.9: EEVDF schedule of a task system. τ_1 has a weight of 4 and always issues requests of size 2 ms. τ_2 has a weight of 2 and always issues requests of size 2 ms, although its second request issued at actual time 8 completes early. τ_3 has a weight of 2 and always issues requests of size 1 ms.

time 8, τ_1 receives $2/3$ of the processor time and τ_2 receives $1/3$. As long as some task is present, the CPU is never idle.

In order to distribute processor time in accordance with the weights, EEVDF maintains the current *virtual time*. The speed of virtual time relative to actual time depends on the total weight of all tasks in the system. Specifically, if $A(t)$ is the set of active tasks at time t and W_i is the weight of τ_i, then the speed of virtual time at actual time t is $\frac{1}{\sum_{\tau_i \in A(t)} W_i}$, and the virtual time $v(t)$ corresponding to actual time t is

$$v(t) = \int_0^t \frac{1}{\sum_{\tau_i \in A(t)} W_i} dt.$$

For example, between time 0 and time 2 in Figure 2.9, only τ_1 is present, with a weight of 4. Therefore, the speed of virtual time in this interval is $\frac{1}{4}$, and $v(2) = \int_0^2 \frac{1}{4} dt = 0.5$.

Each task repeatedly makes *requests* for CPU time, making a new request as soon as its previous request has completed. When a task enters the system, it makes its first request. That request is said to have an *eligible time* at that time. Each request also has an associated size s, indicating the amount of actual time desired for computation. However, it is possible for the task to complete executing its request before it has used a full s units of execution, as τ_2 does in Figure 2.9 for the request issued at actual time 8.

Once a task completes executing its request, it usually initiates another request. If the just-finished request had an eligible *virtual* time of r and an *actual* execution time of a, the new request has an eligible time at *virtual* time $r+\frac{a}{W_i}$. Alternatively, the task may exit the system at the time its next request would otherwise be eligible, as τ_1 does in Figure 2.9 at actual time 16.[5]

For example, the first request of τ_1 in Figure 2.9 has an eligible virtual time of 0 and executes for 2 ms. Therefore, the eligible virtual time for the second request is $0+\frac{2}{4}=0.5$. Similarly, the second request of τ_1 has an eligible virtual time of 0.5, as just computed, and also executes for 2 ms. Thus, the eligible virtual time for the third request is $0.5+\frac{2}{4}=1$. Observe that the delay between eligible *virtual* times is 0.5 ms in both cases, but the delay between eligible *actual* times is 2 ms between the first and second request, but 3 ms between the second and third requests. This occurs because the virtual time clock runs more slowly once τ_2 enters the system.

Each virtual request has a *virtual deadline* that is used to determine scheduling priority. If the request has a *virtual* eligible time of r and a request size of s, then its *virtual* deadline is at time $d=r+\frac{s}{W_i}$. For example, the first request of τ_1 in Figure 2.9 has a virtual eligible time of 0 and a request size of 2 ms, so its virtual deadline is $0+\frac{2}{4}=0.5$. If a request runs for its full request size, then its virtual deadline is identical to the virtual eligible time of the next request. However, if a request completes early, as happens to the second request of τ_2 that completes at actual time 12, the virtual eligible time of the next request may be earlier than the virtual deadline of the just-finished request.

EEVDF prioritizes requests by earliest virtual deadline, considering only requests that have reached their eligible times but have not completed. For example, at actual time 2 in Figure 2.9, τ_1 has a request with a virtual deadline of 1 and τ_2 has a request with a virtual deadline of 1.5. Because τ_1 has an earlier virtual deadline, its request runs for the requested 2 ms. The next request of τ_1 does not have an eligible time until virtual time 1, or actual time 5, so τ_2's request runs from actual time 4 to actual time 5. In Figure 2.9, deadline ties are broken by task index, so when τ_1's third request becomes eligible at actual time 5, it preempts the executing request of τ_2.

Observe in Figure 2.9 that τ_3 and τ_2 receive the same processor share, even though their request sizes differ. The request size of τ_2 is always 2 ms (even though the full size may not be used) and the request size of τ_3 is always 1. However, from actual time 8 onward (when τ_3 enters the system), τ_3 releases jobs twice

[5] Stoica et al. (1996) provide more complex rules that allow a task to leave at other times, but we do not consider those here.

as frequently as τ_2, except for the shift in release time for τ_2 caused by the early completion. This occurs because both tasks have the same weight.

2.7 Overload Management within Mixed-Criticality Scheduling

Recall that, as discussed in Section 1.2, each task in a mixed-criticality system has multiple PWCETs, and guarantees at level ℓ are conditioned on all jobs running for at most their respective level-ℓ PWCETs. However, it is possible that some task's level-ℓ PWCET was insufficiently pessimistic and is overrun by some job. This is a form of overload.

As Santy et al. (2012) pointed out, many mixed-criticality scheduling algorithms respond to such a PWCET overrun by simply dropping all jobs of level-ℓ tasks from that point forward. However, this is usually an unacceptable response. In Chapter 6 we discuss how to handle level-C PWCET overruns in MC^2, and here we discuss prior work addressing this problem within other mixed-criticality schedulers. In Section 2.7.1, we consider methods that reduce the number of low-criticality jobs that are dropped, and in Section 2.7.2, we discuss methods that scale minimum separation times as an alternative to dropping jobs.

2.7.1 Techniques to Reduce Dropped Low-Criticality Jobs

Baruah et al. (2010) introduced the *own-criticality based priority* (*OCBP*) technique for determining static task priorities for mixed-criticality scheduling. Traditionally, when some job scheduled using this technique overruns its level-ℓ PWCET, all jobs at levels ℓ and below are dropped from that point forward. However, Santy et al. (2012) proposed three improvements to this technique. For each, suppose that some job $\tau_{i,k}$ of τ_i overruns its PWCET.

1. If some τ_j has a lower criticality *but a higher priority* than τ_i, it is not necessary to drop jobs from τ_i. This follows from a property of the analysis.

2. It is possible to set an *allowance* for each such τ_i and criticality level ℓ below that of τ_i, so that if $\tau_{i,k}$ exceeds its PWCET by less than that allowance, it is not necessary to drop jobs at level ℓ. This technique is based on the work of Bougueroua et al. (2007), and is enforced by the *Latest Completion Time* LCT mechanism that Santy et al. propose.

3. If no jobs at the level of τ_i are eligible for execution, then jobs no longer need to be dropped, and the system can be returned to normal operation.

Santy et al. demonstrated that these techniques can significantly reduce the number of dropped jobs, primarily due to the ability to only temporarily drop jobs from a task.

Santy et al. (2013) proposed two similar mechanisms to stop dropping jobs for low-criticality tasks, but on multiprocessors.

The first mechanism Santy et al. (2013) proposed applies to fixed-priority systems. In order to restore the system to level ℓ, the system keeps track of a series of times $f_i^{\checkmark}$, ordered by decreasing task priority. $f_0^{\checkmark}$ is the last completion time of a job that overran its level-ℓ PWCET. For $i > 0$, $f_i^{\checkmark}$ is the earliest time not earlier than $f_{i-1}^{\checkmark}$ such that there is no active job of τ_i. Once $f_n^{\checkmark}$ has been detected, where n is the number of tasks, all tasks with criticalities at least ℓ can execute jobs. Furthermore, Santy et al. demonstrated that summing bounds on the response time of all tasks provides a bound on the time it will take for such an $f_n^{\checkmark}$ to occur after an overload finishes.

The second mechanism Santy et al. (2013) proposed applies to any system where job priorities are fixed. The mechanism to return the system to level ℓ works by tracking the actual schedule relative to a *reference schedule* in which all jobs run for their level-ℓ PWCETs. In order to do so, the system must simulate the reference schedule and compare the remaining execution for each job between the actual schedule and the reference schedule. Once all jobs have sufficiently short remaining execution to complete ahead of the reference schedule, all tasks with criticalities at least ℓ can execute jobs.

These mechanisms prevent low-criticality tasks from being permanently impacted by an overload. However, they do not allow these tasks to run at all for a period of time.

2.7.2 Scaling Separation Times of Low-Criticality Jobs Instead of Dropping Jobs

Su and Zhu (2013) proposed an alternative task model that allows for low-criticality tasks to have both a *desired* period and a *maximum* period. For a properly provisioned system, it is possible to guarantee that low-criticality tasks can execute with their maximum periods *even when high-criticality tasks run for their full PWCETs*, while executing tasks at or close to their desired periods in the expected case. This task model is called the *elastic mixed-criticality* (*E-MC*) task model. Unlike the similarly named model from Buttazzo et al. (2002), E-MC does not use an elasticity factor to determine the extent of scaling of each task.

In order to schedule E-MC task systems, Su and Zhu (2013) also proposed a modified version of the *earliest deadline first with virtual deadlines* (*EDF-VD*) scheduler (Baruah et al., 2012), called the *early-release EDF* (*ER-EDF*) scheduler. ER-EDF maintains a set of *wrapper-tasks* (Zhu and Aydin, 2009) that keep track of the slack that is created when high-criticality jobs finish ahead of their high-level PWCETs. Each low-criticality job is guaranteed to release no later than its task's maximum period after the release of its predecessor. However, such a job also has a set of *early release points*. Each time such a point arrives, if there is enough slack (as indicated by the wrapper-tasks) for the job to be released early, ER-EDF does so. In the common case, high-criticality jobs usually run for less than their high-level PWCETs, so low-criticality jobs run more frequently than their minimum guarantee. However, even during an overload, low-criticality jobs continue to receive a minimum level of service.

Su et al. (2013) later extended this work to multicore systems. The extension is basically a partitioned variant of ER-EDF. Su et al. considered partitioning the task system using several different partitioning heuristics. For high-criticality tasks, they used utilizations based on high-criticality PWCETs, and for low-criticality tasks, they used utilizations based on low-criticality PWCETs and maximum periods. A worst-fit decreasing heuristic based on those utilizations, ignoring criticalities, tended to perform the best and to significantly outperform the global EDF-VD algorithm (Li and Baruah, 2012).

Su et al. also considered two different techniques to reclaim slack. The simplest is to use the same strategy as ER-EDF, allowing low-criticality tasks to reclaim slack from high-criticality tasks on the same processor. They also considered a *global* slack reclamation technique. Under that technique, when there is not enough slack to release a job early on the core to which its task has been assigned, if there is enough slack on a remote processor, then that single job is migrated to the remote processor. Su et al. demonstrated that this technique can significantly improve the performance of their algorithm.

Jan et al. (2013) provided a different mechanism to minimize the separation time of low-criticality releases. They assumed that high-criticality jobs are statically prioritized over low-criticality jobs, and that the system optimistically schedules low-criticality jobs with deadlines that match their desired separation times. However, when a likely deadline miss is expected, the deadline is pushed back at that time. Furthermore, Jan et al. provided a mechanism where the system can specify a prioritization for which tasks to scale back first.

2.8 Summary

In this chapter, we reviewed prior work on SRT scheduling and overload. We discussed both prior SRT work using the bounded tardiness model that is closely related to the bounded lateness model used in this dissertation, and prior SRT work using other models of SRT. We then focused in more detail on prior work dealing with overload management, including that focusing on MC systems. In the following chapters, we will discuss our approach to SRT scheduling and overload management within MC^2, building on some of this prior work.

CHAPTER 3: FAIR LATENESS SCHEDULING[1]

In this chapter, we extend the tardiness bounds from Devi and Anderson (2008) to provide lateness bounds for arbitrary GEL schedulers. We also propose a specific scheduler—the global fair lateness (G-FL) scheduler mentioned in Section 1.3—that minimizes maximum lateness bounds, as well as a general linear programming technique to minimize other criteria related to lateness.

The remainder of this chapter is organized as follows. In Section 3.1, we describe the task model under consideration and define basic terms. In Section 3.2, we present a novel technique called *compliant-vector analysis* (*CVA*). In Section 3.3, we present the basic linear programming technique for working with CVA, and we use it to prove several properties about CVA. In Section 3.4, we present G-FL and prove that it minimizes the maximum lateness bound over all tasks. Afterwards, in Section 3.5, we describe the use of linear programming to define GEL schedulers to optimize values of other functions of lateness. In Section 3.6, we present experiments comparing lateness bounds for G-FL, G-EDF, and other LP-derived GEL schedulers. These experiments show that G-FL and LP-derived GEL schedulers can provide significantly lower lateness bounds than G-EDF.

3.1 Task Model

For the reader's convenience, a table of notation used in this chapter is provided in Appendix A. We consider the arbitrary-deadline sporadic task model discussed in Section 1.1.1. All quantities are real-valued.

[1] This work appeared in preliminary form in the following papers:

Erickson, J., Devi, U., and Baruah, S. (2010a). Improved tardiness bounds for global EDF. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems*, pages 14–23.

Erickson, J., Guan, N., and Baruah, S. (2010). Tardiness bounds for global EDF with deadlines different from periods. In *Proceedings of the 14th International Conference on Principles of Distributed Systems*, pages 286–301.

Erickson, J. and Anderson, J. (2012). Fair lateness scheduling: Reducing maximum lateness in G-EDF-like scheduling. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems*, pages 3–12.

Ward, B. C., Erickson, J. P., and Anderson, J. H. (2013). A linear model for setting priority points in soft real-time systems. In *Proceedings of Real-Time Systems: The Past, the Present, and the Future–A Conference Organized in Celebration of Alan Burns' Sixtieth Birthday*, pages 192–205.

Erickson, J. P., Anderson, J. H., and Ward, B. C. (2014). Fair lateness scheduling: reducing maximum lateness in G-EDF-like scheduling. *Real-Time Systems*, 50(1):5–47.

In addition to the notation introduced in Section 1.1.1, we let $C_{\max}$ denote the largest WCET in the system, i.e.,

$$C_{\max} \triangleq \max_{\tau_i \in \tau}\{C_i\}, \tag{3.1}$$

and we define

$$U^+ \triangleq \left\lceil \sum_{\tau_i \in \tau} U_i \right\rceil. \tag{3.2}$$

We assume that $n > m$. If this is not the case, then each task can be assigned its own processor, and no job of each τ_i will have a response time exceeding C_i.

Furthermore, we assume that for all i,

$$U_i \leq 1, \tag{3.3}$$

and that

$$\sum_{\tau_i \in \tau} U_i \leq m, \tag{3.4}$$

as both conditions were demonstrated by Leontyev and Anderson (2010) to be necessary conditions for bounded tardiness. (This also implies that they are necessary conditions for bounded lateness.)

We define a job's *proportional lateness* (*tardiness*) as its lateness (tardiness) divided by its relative deadline. We consider per-task bounds on response time, lateness, and proportional lateness.

We use for each τ_i the notation R_i to refer to its response time bound, L_i to refer to its lateness bound, and I_i to refer to its proportional lateness bound. From the definition of lateness in Section 1.1.1,

$$L_i \triangleq R_i - D_i. \tag{3.5}$$

From the definition of proportional lateness above,

$$I_i \triangleq L_i / D_i. \tag{3.6}$$

We assume

$$\forall i, Y_i \geq 0. \tag{3.7}$$

For all variables subscripted with an i, we also use vector notation to refer to the set of all values for the task system. For example, $\vec{T} = \langle T_1, T_2, \ldots T_n \rangle$.

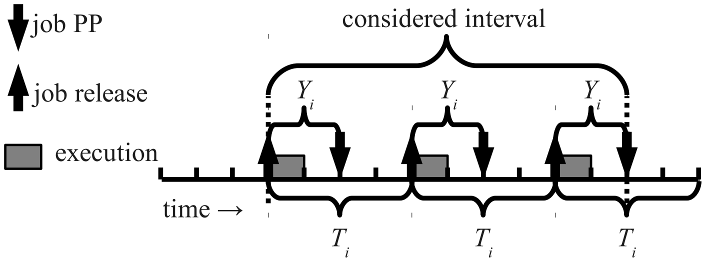


Figure 3.1: Illustration of the worst-case arrival pattern for analyzing demand within an interval.

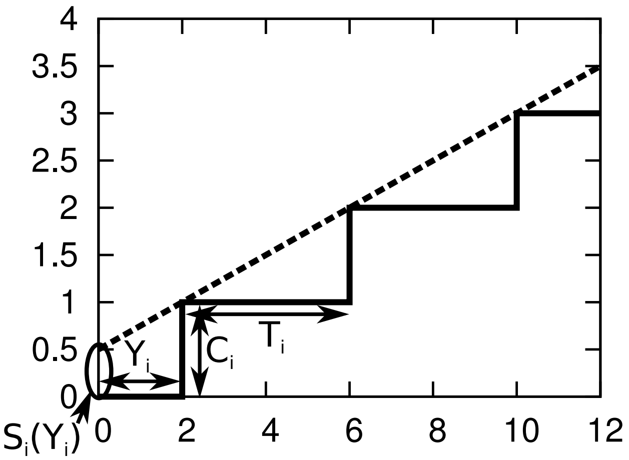


Figure 3.2: Illustration of the technique for bounding the demand for the task in Figure 3.1.

3.2 Basic Compliant-Vector Analysis

In this section, we present the CVA necessary to analyze arbitrary GEL schedulers, including G-FL. We first describe the lateness bounds for an arbitrary GEL scheduler, providing a general condition that results in correct lateness bounds, and then prove it correct.

We will first provide definitions needed to specify our lateness bounds, accompanied by some basic intuition. We will then prove our bounds correct.

While analyzing a task system, we need to account for the total processor demand that each task can require over certain intervals, accounting only for jobs that have both releases and PPs within that interval. We will use a linear upper bound for this quantity. In the context of implicit-deadline systems (where $D_i = T_i$), we can simply multiply the length of such an interval by the utilization of the task being analyzed. When $Y_i < T_i$, this technique may underestimate the demand, as depicted in Figure 3.1, where $3C_i$ units of demand are present in an interval shorter than $3T_i$ units. We will use the term

$$S_i(Y_i) \triangleq C_i \cdot \max\left\{0, 1 - \frac{Y_i}{T_i}\right\}, \tag{3.8}$$

illustrated in Figure 3.2, to account for this extra demand. We will also use the total of such demand across the whole system,

$$S(\vec{Y}) \triangleq \sum_{\tau_i \in \tau} S_i(Y_i). \tag{3.9}$$

We define our lateness bounds recursively, defining for each τ_i a real value x_i such that each job of τ_i has a response time of at most

$$R_i \triangleq Y_i + x_i + C_i \tag{3.10}$$

and thus an lateness bound of

$$L_i \triangleq Y_i + x_i + C_i - D_i. \tag{3.11}$$

As part of our lateness expression, we define the term

$$G(\vec{x}, \vec{Y}) \triangleq \sum_{(U^+ - 1)\text{ largest}} (x_i U_i + C_i - S_i(Y_i)), \tag{3.12}$$

which accounts for demand from certain critical tasks that can contribute to the lateness in the system. $G(\vec{x})$ accounts for demand over the same intervals as $S(\vec{Y})$, and like $S(\vec{Y})$, $G(\vec{x})$ actually accounts for demand in excess of what would be predicted simply from using utilization multiplied by the interval length. Furthermore, as shown in the analysis below, a task that contributes to $G(\vec{x})$ does not actually need to contribute to $S(\vec{Y})$. In order to simplify the expression for the bounds, we therefore include the negative $S_i(Y_i)$ term in (3.12) so that $S(\vec{Y})$ remains a sum across all tasks.

We will show at the end of this section, as Theorem 3.1, that if $\vec{x}$ is *compliant* (defined below) then (3.10) is a correct response time bound for each τ_i. The analysis is most interesting for the common case when $U^+ > 1$, but our definition also accounts for the simpler degenerate case that $U^+ = 1$.

Definition 3.1. $\vec{x}$ is *near-compliant* iff

$$\forall i, x_i \geq \frac{G(\vec{x}, \vec{Y}) + S(\vec{Y}) - C_i}{m}. \tag{3.13}$$

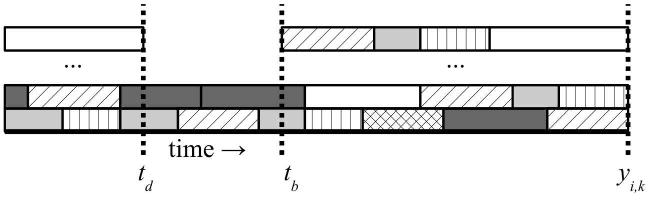


Figure 3.3: Example schedule depicting t_d, t_b, and $y_{i,k}$. Different lines represent different processors, and different hatchings or shadings represent different tasks.

A near-compliant vector is *compliant* iff $\forall i, x_i \geq 0$ or $U^+ = 1$.

For the remainder of this section, we consider an arbitrary but fixed schedule. The eventual proof of Theorem 3.1 will proceed by induction over the jobs of that schedule considered in priority order. We will bound the response time of a job $\tau_{i,k}$ of task τ_i, based on the inductive assumption that jobs with higher priority than $\tau_{i,k}$ have response times no greater than specified by (3.10).

We define H as the set of all jobs with priority at least that of $\tau_{i,k}$. By the definition of priority, we do not need to consider any work due to jobs not in H when analyzing the response time of $\tau_{i,k}$. We define $W_j(t)$ as the total amount of work remaining at time t for jobs of τ_j in H. We define $W(t)$ as the total amount of work remaining at time t for all jobs in H, i.e., $W(t) = \sum_{\tau_j \in \tau} W_j(t)$.

Several important time instants are depicted in Figure 3.3. We denote as $y_{i,k}$ (priority) the PP of $\tau_{i,k}$. We denote as t_b (*b*usy) the earliest time instant such that during every time in $[t_b, y_{i,k})$, at least U^+ processors are executing jobs from H. We denote as t_d (i*d*le) the earliest time instant such that during every time in $[t_d, t_b)$, fewer than U^+ processors are executing jobs from H.

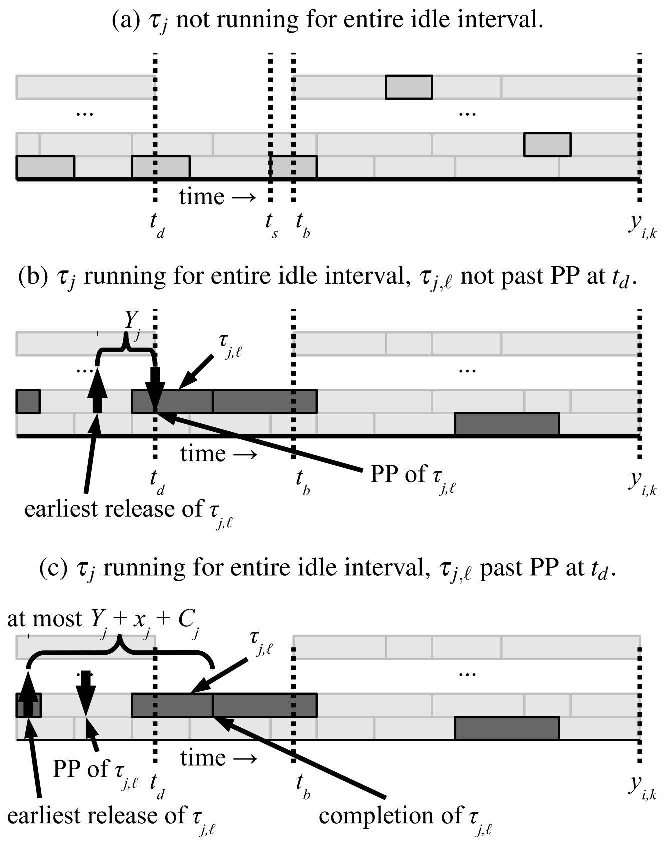
Throughout our proofs, we will refer to the total demand that a task τ_i can produce within an interval of length Δ, accounting for jobs that have both release times and PPs within the interval. This quantity is denoted $\text{DBF}(\tau_i, Y_i, \Delta)$.

The next two lemmas, both proved in Appendix B, provide slightly pessimistic bounds on $\text{DBF}(\tau_i, Y_i, \Delta)$. Lemma 3.1 provides a less pessimistic bound than Lemma 3.2 when $Y_i > T_i$, but it only applies for intervals of length greater than Y_i.

Lemma 3.1. *If $\Delta \geq Y_i$, then*

$$DBF(\tau_i, Y_i, \Delta) \leq U_i \Delta + C_i \cdot \left(1 - \frac{Y_i}{T_i}\right).$$

Lemma 3.2. $\forall \Delta \geq 0, DBF(\tau_i, Y_i, \Delta) \leq U_i \Delta + S_i(Y_i)$.


(a) τ_j not running for entire idle interval.

(b) τ_j running for entire idle interval, $\tau_{j,\ell}$ not past PP at t_d.

(c) τ_j running for entire idle interval, $\tau_{j,\ell}$ past PP at t_d.

Figure 3.4: Proof details for Theorem 3.1 and supporting lemmas. Different lines in each figure represent different processors. In each figure, the jobs of the task under consideration are outlined in black, while any other jobs are outlined in gray.

Because at least U^+ processors are assumed to be busy in $[t_b, y_{i,k})$, we know that at least $U^+(y_{i,k} - t_b)$ units of work will be completed during that interval. Therefore, in Lemma 3.3, we determine an upper bound on the remaining work at t_b. In order to do so, we will compute an upper bound on the remaining work at t_b for each task in the system. Lemma 3.3 is the core of the argument behind Theorem 3.1, so we prove it in full here.

Lemma 3.3. *If $\vec{x}$ is compliant and all jobs of τ_j with higher priority than $\tau_{i,k}$ complete with response time no greater than $Y_j + x_j + C_j$, then $W(t_b) \leq G(\vec{x}, \vec{Y}) + S(\vec{Y}) + U^+(y_{i,k} - t_b)$.*

Proof. We consider each task τ_j individually. We account for the remaining work for τ_j at time t_b, considering all possible cases, described next.

Case 1 (Figure 3.4(a)). It may be the case that τ_j is executing immediately before t_b, but either is not executing at some time in $[t_d, t_b)$, or is not executing before t_b because $t_b = 0$. In this case, denote as t_s (*s*tart) the earliest time such that τ_j executes continuously in $[t_s, t_b)$. τ_j must not have any unfinished work just before t_s, because at least one processor is available then. Therefore, any work that contributes to $W_j(t_s)$ must consist only of jobs that are released at or after t_s and that have PPs at or before $y_{i,k}$. Therefore,

$$\begin{aligned}
W_j(t_s) &\leq \text{DBF}(\tau_j, Y_j, y_{i,k} - t_s) \\
&\leq \{\text{By Lemma 3.2}\} \\
&\quad U_j(y_{i,k} - t_s) + S_j(Y_j). \qquad (3.14)
\end{aligned}$$

Furthermore, because τ_j executes continuously in $[t_s, t_b)$,

$$\begin{aligned}
W_j(t_b) &= W_j(t_s) - (t_b - t_s) \\
&\leq \{\text{By (3.14)}\} \\
&\quad U_j(y_{i,k} - t_s) + S_j(Y_j) - (t_b - t_s) \\
&\leq \{\text{Because } U_j \leq 1)\} \\
&\quad U_j(y_{i,k} - t_s) + S_j(Y_j) - U_j(t_b - t_s) \\
&= \{\text{Rearranging}\} \\
&\quad U_j(y_{i,k} - t_b) + S_j(Y_j). \qquad (3.15)
\end{aligned}$$

Case 2 (Figure 3.4(b)). It may be the case that τ_j is executing continuously in $[t_d, t_b)$, and its job executing at t_d has a PP at or after t_d. In this case, we call that job $\tau_{j,\ell}$. The release time of $\tau_{j,\ell}$ must be no earlier than $t_d - Y_j$, or Case 3 below would instead apply. Thus, the work for τ_j remaining at t_d must consist only of jobs that have releases at or after $t_d - Y_j$ and PPs at or before $y_{i,k}$. Therefore,

$$\begin{aligned}
W_j(t_d) &\leq \text{DBF}(\tau_j, Y_j, y_{i,k} - (t_d - Y_j)) \\
&\leq \{\text{By Lemma 3.1; note that } y_{i,k} - (t_d - Y_j) = Y_j + (y_{i,k} - t_d) \geq Y_i\} \\
&\quad U_j(y_{i,k} - (t_d - Y_j)) + C_j\left(1 - \frac{Y_j}{T_j}\right) \\
&= \{\text{Rearranging and using } U_j = C_j/T_j\} \\
&\quad U_j(y_{i,k} - t_d) + \frac{C_j Y_j}{T_j} + C_j - \frac{C_j Y_j}{T_j}
\end{aligned}$$

$$
\begin{aligned}
&= \{\text{Cancelling}\}\\
&\quad U_j(y_{j,\ell} - t_d) + C_j. && (3.16)
\end{aligned}
$$

Because τ_j runs continuously in $[t_d, t_b)$,

$$
\begin{aligned}
W_j(t_b) &= W_j(t_d) - (t_b - t_d)\\
&\leq \{\text{By (3.16)}\}\\
&\quad U_j(y_{i,k} - t_d) + C_j - (t_b - t_d)\\
&\leq \{\text{Because } U_j \leq 1)\}\\
&\quad U_j(y_{i,k} - t_d) + C_j - U_j(t_b - t_d)\\
&= \{\text{Rearranging}\}\\
&\quad U_j(y_{i,k} - t_b) + C_j. && (3.17)
\end{aligned}
$$

Case 3 (Figure 3.4(c)). It may be the case that τ_j is executing continuously in $[t_d, t_b)$, and its job executing at t_d has a PP earlier than t_d. We call that job $\tau_{j,\ell}$. We refer to the PP of $\tau_{j,\ell}$ as $y_{j,\ell}$. Because $y_{j,\ell} < t_d \leq t_b \leq y_{i,k}$, $\tau_{j,\ell}$ must have priority higher than $\tau_{i,k}$, i.e., $\tau_{j,\ell}$ is not $\tau_{i,k}$. Therefore, by the precondition of the lemma, $\tau_{j,\ell}$'s response time must be no greater than

$$Y_j + x_j + C_j. \tag{3.18}$$

We define δ such that the remaining execution of $\tau_{j,\ell}$ at t_d is $C_j - \delta$. If $\tau_{j,\ell}$ runs for its full WCET, then δ is simply its execution before t_d, but δ may be larger if $\tau_{j,\ell}$ completes early. $\tau_{j,\ell}$ must finish no earlier than $t_d + C_j - \delta$. By the definition of response time from Section 3.1 and (3.18), τ_j must be released no earlier than $t_d + C_j - \delta - (Y_j + x_j + C_j)$. Thus, the work for τ_j remaining at t_d must consist only of jobs that have releases at or after $t_d + C_j - \delta - (Y_j + x_j + C_j)$ and PPs at or before $y_{i,k}$. Below we use Lemma 3.1 to bound the work from these jobs, so we need to establish that the length of this interval is at least Y_j. We first observe that, resulting from the definition of t_b and the fact that τ_j is running just before t_b, $U^+ > 1$. Therefore, by the precondition of the lemma and Definition 3.1,

$$x_j \geq 0. \tag{3.19}$$

Thus,

$$
\begin{aligned}
y_{i,k} - (t_d + C_j - \delta - (Y_j + x_j + C_j)) &= \{\text{Rearranging}\} \\
&\quad (y_{i,k} - t_d) + x_j + \delta + Y_j \\
&\geq \{\text{Because } y_{i,k} \geq t_d,\ \delta \geq 0, \text{ and by (3.19)}\} \\
&\quad Y_j. \qquad (3.20)
\end{aligned}
$$

Recall that only $C_j - \delta$ units of work remain for $\tau_{j,\ell}$ at time t_d, but $\mathrm{DBF}(\tau_j, Y_j, \Delta)$ assumes a demand of C_j for every job within the considered interval. Therefore,

$$
\begin{aligned}
W_j(t_d) &\leq \mathrm{DBF}(\tau_j, Y_j, y_{i,k} - (t_d + C_j - \delta - (Y_j + x_j + C_j))) - \delta \\
&\leq \{\text{By Lemma 3.1, and by (3.20) }\} \\
&\quad U_j(y_{i,k} - (t_d + C_j - \delta - (Y_j + x_j + C_j))) + C_j \left(1 - \frac{Y_j}{T_j}\right) - \delta \\
&\leq \{\text{Rearranging, using } U_j = C_j/T_j\text{, and because } U_j \leq 1)\} \\
&\quad U_j(y_{i,k} - t_d) - U_jC_j + U_j\delta + \frac{C_jY_j}{T_j} + U_jx_j + U_jC_j + C_j - \frac{C_jY_j}{T_j} - U_j\delta \\
&= \{\text{Cancelling}\} \\
&\quad U_j(y_{i,k} - t_d) + U_jx_j + C_j. \qquad (3.21)
\end{aligned}
$$

Because τ_j runs continuously in $[t_d, t_b)$,

$$
\begin{aligned}
W_j(t_b) &= W_j(t_d) - (t_b - t_d) \\
&\leq \{\text{By (3.21)}\} \\
&\quad U_j(y_{i,k} - t_d) + U_jx_j + C_j - (t_b - t_d) \\
&\leq \{\text{Because } U_j \leq 1)\} \\
&\quad U_j(y_{i,k} - t_d) + U_jx_j + C_j - U_j(t_b - t_d) \\
&= \{\text{Rearranging}\} \\
&\quad U_j(y_{i,k} - t_b) + U_jx_j + C_j. \qquad (3.22)
\end{aligned}
$$

Total Remaining Work at t_b. We consider two possible cases, depending on the value of U^+.

If $U^+ = 1$, then by the definition of t_b, all tasks must be in Case 1. In addition, by the definition of $G(\vec{x},\vec{Y})$ in (3.12),

$$G(\vec{x},\vec{Y}) = 0. \tag{3.23}$$

Therefore, the total work is at most

$$\begin{aligned}
\sum_{\tau_j \in \tau} (U_j(y_{i,k} - t_b) + S_j(Y_j)) = \{&\text{Rearranging}\} \\
&\left(\sum_{\tau_j \in \tau} U_j\right)(y_{i,k} - t_b) + \sum_{\tau_j \in \tau} S_j(Y_j) \\
\leq \{&\text{By the definition of } U^+ \text{ in (3.2), the definition of } S(\vec{Y}) \text{ in (3.9), and (3.23)}\} \\
&G(\vec{x},\vec{Y}) + S(\vec{Y}) + U^+(y_{i,k} - t_b),
\end{aligned}$$

and the lemma holds.

On the other hand, if $U^+ > 1$, then by Definition 3.1,

$$\forall j, x_j \geq 0. \tag{3.24}$$

Because we assume that $\forall i, Y_i \geq 0$, and by the definition of $S_i(Y_i)$ in (3.8),

$$\forall j, S_j(Y_j) \leq C_j. \tag{3.25}$$

By (3.24) and (3.25), the worst case for each τ_j is Case 3. Any task can be in Case 1, but due to the definition of t_b, at most $U^+ - 1$ tasks can be in Case 2 or 3 due to the definition of t_b. Therefore, the total demand due to all tasks can be upper bounded by selecting a set M of $U^+ - 1$ tasks in τ such that M maximizes

$$\begin{aligned}
&\sum_{\tau_j \in M} (U_j(y_{i,k} - t_b) + U_j x_j + C_j) + \sum_{\tau_j \in (\tau \setminus M)} (U_j(y_{i,k} - t_b) + S_j(Y_j)) \\
&= \{\text{Rearranging}\} \\
&\left(\sum_{\tau_j \in \tau} U_j\right)(y_{i,k} - t_b) + \sum_{\tau_j \in M} (x_j U_j + C_j) + \sum_{\tau_j \in (\tau \setminus M)} S_j(Y_j)
\end{aligned}$$

$$
\begin{aligned}
&= \{\text{Adding } \textstyle\sum_{\tau_j \in M} S_j(Y_j) - \sum_{\tau_j \in M} S_j(Y_j) = 0\} \\
&\quad \left(\sum_{\tau_j \in \tau} U_j\right)(y_{i,k} - t_b) + \sum_{\tau_j \in M} (x_j U_j + C_j - S_j(Y_j)) + \sum_{\tau_j \in \tau} S_j(Y_j) \\
&\leq \{\text{By the definition of } U^+ \text{ in (3.2), the definition of } G(\vec{x}, \vec{Y}) \text{ in (3.12),} \\
&\quad \text{the definition of } S(\vec{Y}) \text{ in (3.9), and the definition of } M \text{ above}\} \\
&\quad U^+(y_{i,k} - t_b) + G(\vec{x}, \vec{Y}) + S(\vec{Y}).
\end{aligned}
$$

Therefore, the lemma holds. □

The next lemma simply bounds $W(y_{i,k})$, given the bound in Lemma 3.3.

Lemma 3.4. *If $\vec{x}$ is compliant and all jobs $\tau_{j,\ell}$ with higher priority than $\tau_{i,k}$ complete with response time no greater than $Y_j + x_j + C_j$, then $W(y_{i,k}) \leq G(\vec{x}, \vec{Y}) + S(\vec{Y})$.*

Proof. By the definition of t_b, at least U^+ processors are completing work from jobs in H throughout the interval $[t_b, y_{i,k})$. Therefore,

$$
\begin{aligned}
W(y_{i,k}) &\leq W(t_b) - U^+(y_{i,k} - t_b) \\
&\leq \{\text{By Lemma 3.3}\} \\
&\quad G(\vec{x}, \vec{Y}) + S(\vec{Y}).
\end{aligned}
$$
□

Our final lemma, proved in Appendix B, bounds the response time of $\tau_{i,k}$.

Lemma 3.5. *If $\vec{x}$ is compliant and each job $\tau_{j,\ell}$ with higher priority than $\tau_{i,k}$ finishes with response time no greater than $Y_j + x_j + C_j$, then $\tau_{i,k}$ finishes with response time no greater than $Y_i + x_i + C_i$.*

With these lemmas in place, we can now state and prove the primary result of this section.

Theorem 3.1. *If $\vec{x}$ is compliant, then no job of any task τ_i will have a response time exceeding $Y_i + x_i + C_i$.*

Proof. This result follows immediately from Lemma 3.5 by induction over all jobs in the system, considered in order of decreasing priority. As the base case, the precondition for Lemma 3.5 holds vacuously when considering the first job. □

3.3 Minimum Compliant Vector

In this section, we will demonstrate that there is a unique *minimum* compliant vector for a given selection of PPs. We begin by narrowing our consideration of compliant vectors based on the idea that if we can reduce the bound for one task without increasing any other bound in the system, we clearly do not have the "best" compliant vector. Based on this insight, Definition 3.2 below defines a notion that will help us define the "best" compliant vector. (We prove in Lemma 3.7 below that the minimum near-compliant vector is unique.)

Definition 3.2. A *minimum near-compliant vector* is a near-compliant vector such that, if any component x_i is reduced by any $\delta > 0$, the vector ceases to be near-compliant.

We next precisely characterize any minimum near-compliant vector. Lemma 3.6 is proved in Appendix B.

Lemma 3.6. *$\vec{x}$ is a minimum near-compliant vector if and only if*

$$\forall i, x_i = \frac{G(\vec{x},\vec{Y}) + S(\vec{Y}) - C_i}{m}. \tag{3.26}$$

Lemma 3.6, when combined with the fact that a minimum near-compliant vector is in fact compliant (proved as Corollary 3.2 below), allows us to constrain the form of the vectors considered. We state this observation as Corollary 3.1 below, which follows immediately from Lemma 3.6.

Corollary 3.1. *$\vec{x}$ is a minimum near-compliant vector if and only if there exists s such that*

$$\forall i, x_i = \frac{s - C_i}{m} \tag{3.27}$$

and

$$s = G(\vec{x},\vec{Y}) + S(\vec{Y}). \tag{3.28}$$

Observe that s is independent of the task index i.

The following lemma, proved in Appendix B, shows that we can unambiguously refer to *the* minimum near-compliant vector for the system.

Lemma 3.7. *If the minimum near-compliant vector for the task system exists, it is unique.*

Later in this section, we prove that the minimum near-compliant vector does indeed exist. We do so by formulating a feasible LP such that the optimal solution must contain the minimum near-compliant vector.

In addition to showing that the minimum near-compliant vector exists, this LP is also useful for actually computing the minimum near-compliant vector.

In order to prove that the minimum near-compliant vector exists, we will show that the minimum near-compliant vector is the optimal solution to a more general LP based on Lemma 3.8 below.

Lemma 3.8. *$\vec{x}$ is a near-compliant vector if* (3.27) *holds and*

$$s \geq G(\vec{x}, \vec{Y}) + S(\vec{Y}). \tag{3.29}$$

Proof. Follows immediately from Definition 3.1. □

Recall that a compliant vector is described by (3.8)–(3.9) and (3.12)–(3.13). We now show that Lemma 3.8 and the equations it depends on can be reformulated as constraints for the LP mentioned above. Throughout this section, whenever we use a subscript i (e.g., Y_i), there is one variable or constant for each task (e.g., a Y_i for each task). We assume that x_i, S_i, S_{sum}, G, and s are variables, and we also introduce the auxiliary variables b and z_i. All other values are constants (i.e., Y_i, U_i, C_i, T_i, D_i, U^+, and m), but Y_i will be used as a variable later in Sections 3.4 and 3.5.

Constraint Set 3.1. *The linear constraints corresponding to the definition of $\vec{x}$ in* (3.27) *used in Lemma 3.8 are given by*

$$\forall i : x_i = \frac{s - C_i}{m}.$$

Constraint Set 3.2. *The linear constraints corresponding to the definition of $S_i(Y_i)$ in* (3.8) *are given by*

$$\forall i : S_i \geq 0; \quad S_i \geq C_i(1 - Y_i/T_i).$$

The two constraints in Constraint Set 3.2 model the two terms of the max in the definition of $S_i(Y_i)$ in (3.8), ensuring that $S_i \geq S_i(Y_i)$ holds for each i. Because S_i is not bounded from above, Constraint Set 3.2 only ensures that

$$\forall i, S_i \geq S_i(Y_i). \tag{3.30}$$

However, we will show in Lemma 3.10 below that this will still result in a near-compliant vector.

The next equation to be linearized, the definition of $G(\vec{x},\vec{Y})$ in (3.12), is less straightforward. We first show how to minimize the sum of the largest k elements in a set $A=\{a_1,\ldots,a_n\}$ using only linear constraints, by an approach similar to one proposed by Ogryczak and Tamir (2003). The intuition behind this approach is shown in Figure 3.5. This figure corresponds to the following LP,

$$\begin{aligned}
&\text{Minimize:} && G\\
&\text{Subject to:} && G = kb+\sum_{i=1}^{n} z_i,\\
& && \forall i : z_i \geq 0,\\
& && \forall i : z_i \geq a_i - b,
\end{aligned}$$

in which $G,\ b$, and z_i are variables and both k and a_i are constants. In Figure 3.5, the term kb corresponds to the gray-shaded area, and $\sum_{i=1}^{n} z_i$ corresponds to the black-shaded area. When G is minimized, it is equal to the sum of the largest k elements in A. This is achieved when $z_i = 0$ for each element a_i that is not one of the k largest elements in A, and b is at most the k^{th} largest element in A, as is shown in Figure 3.5(b).

Using this technique, we can formulate the definition of $G(\vec{x},\vec{Y})$ in (3.12), as a set of linear constraints.

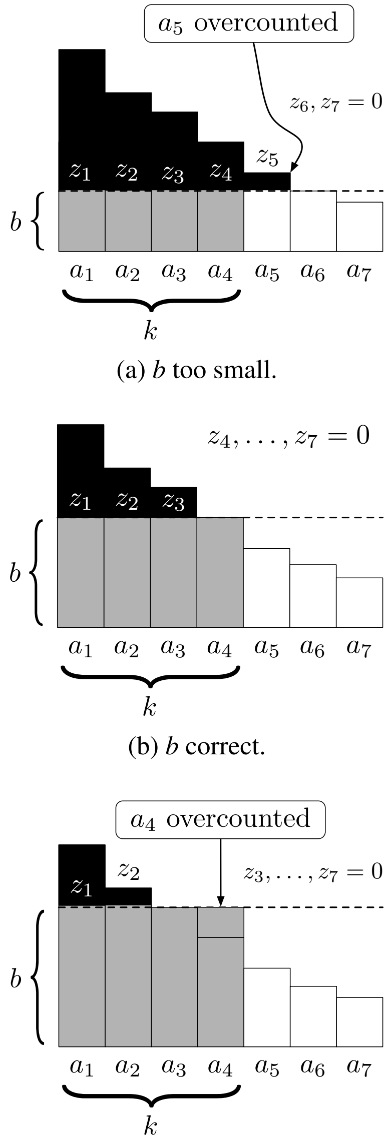
Constraint Set 3.3. *The linear constraints corresponding to the definition of* $G(\vec{x},\ \vec{Y})$ *in* (3.12) *are given by*

$$\begin{aligned}
&G = b\cdot(U^+-1)+\sum_{\tau_i\in\tau} z_i,\\
&\forall i : z_i \geq 0,\\
&\forall i : z_i \geq x_iU_i + C_i - S_i - b.
\end{aligned}$$

In some of the optimization objectives we consider in Section 3.5, G is not itself explicitly minimized, as in the example LP above. Furthermore, this constraint uses S_i in place of $S_i(Y_i)$. As a result, this set of constraints only ensures that

$$G \geq \sum_{U^+-1 \text{ largest}} (x_iU_i + C_i - S_i). \tag{3.31}$$

As shown in Lemma 3.10 below, this is sufficient to ensure a near-compliant vector.


a_5 overcounted
$z_6, z_7 = 0$
z_1 z_2 z_3 z_4 z_5
b
a_1 a_2 a_3 a_4 a_5 a_6 a_7
k
$z_4, \ldots, z_7 = 0$
z_1 z_2 z_3
b
a_1 a_2 a_3 a_4 a_5 a_6 a_7
k
a_4 overcounted
z_2
z_1
$z_3, \ldots, z_7 = 0$
b
a_1 a_2 a_3 a_4 a_5 a_6 a_7
k

(a) *b* too small.

(b) *b* correct.

(c) *b* too large.

Figure 3.5: Illustration of the auxiliary variables used to sum the largest k elements in the set $A = \{a_1, \ldots, a_n\}$. The total of the gray and black shaded areas is equal to G. The gray areas correspond to kb while the black areas correspond to positive z_i's. When G is minimized, as in (b), G is equal to the sum of the largest k elements in A. As is shown in (a) and (c), if b is too small or too large then G will be larger than the maximum k elements in A. Note that elements of A are depicted in sorted order only for visual clarity.

Constraint Set 3.4. *The linear constraint corresponding to the definition of* $S(\vec{Y})$ *in* (3.9) *is given by*

$$S_{\text{sum}} = \sum_{\tau_i \in \tau} S_i.$$

Constraint Set 3.5. *The linear constraint corresponding to* (3.29) *in Lemma 3.8 is given by*

$$s \geq G + S_{\text{sum}}.$$

As noted above, Constraint Sets 3.2–3.3 ensure only that (3.30) and (3.31) are inequalities rather than equalities. The next two lemmas, proved in Appendix B, demonstrate that Constraint Sets 3.1–3.5 are still sufficient to ensure that $\vec{x}$ is a near-compliant vector.

Lemma 3.9. *For any assignment of variables satisfying Constraint Sets 3.1–3.4,*

$$G + S_{\text{sum}} \geq G(\vec{x}, \vec{Y}) + S(\vec{Y}).$$

Lemma 3.10. *For any assignment of variables satisfying Constraint Sets 3.1–3.5,* $\vec{x}$ *is a near-compliant vector.*

We now show that there is a single best minimum near-compliant vector, which is in fact a compliant vector, for the system by minimizing the objective function s under Constraint Sets 3.1–3.5. We show that an optimal value of s exists, and that the corresponding $\vec{x}$ under Constraint Set 3.1 is both the minimum near-compliant vector and in fact compliant.

We will first demonstrate that an optimal $\vec{x}$ exists. Observe from Constraint Set 3.1 that $\vec{x}$ is uniquely determined by the assignment of s. The next two lemmas, proved in Appendix B, show that an optimal value of s exists. Lemma 3.11 demonstrates that there is a lower bound on s for any feasible solution, and Lemma 3.12 demonstrates that a feasible solution exists.

Lemma 3.11. *If* $s < 0$, *then Constraint Sets 3.1–3.5 are infeasible.*

Lemma 3.12. *Constraint Sets 3.1–3.5 are feasible.*

By Lemmas 3.11–3.12 and the fact that Constraint Sets 3.1–3.5 are linear, an optimal minimum value of s must exist. The next lemma shows that the corresponding $\vec{x}$ must be the minimum near-compliant vector.

Lemma 3.13. *For any assignment of variables that satisfies Constraint Sets 3.1–3.5 such that* s *obtains its minimum value,* $\vec{x}$ *is the minimum near-compliant vector.*

Proof. We use proof by contradiction. Assume an assignment of variables that satisfies Constraint Sets 3.1–3.5 such that $\vec{x}$ is not a minimum near-compliant vector and s obtains its minimum value.

We show an assignment of variables, denoted with a prime (e.g. s' for the new assignment of s), that is also feasible, but with $s' < s$.

Let

$$s' = G(\vec{x},\vec{Y}) + S(\vec{Y}), \tag{3.32}$$

$$\forall i, x'_i = \frac{s' - C_i}{m}, \tag{3.33}$$

$$\forall i, S'_i = S_i(Y_i), \tag{3.34}$$

$$G' = G(\vec{x}',\vec{Y}), \tag{3.35}$$

$$b' = (U^+ - 1)^{th} \text{ largest value of } x'_i U_i + C_i - S_i, \tag{3.36}$$

$$\forall i, z'_i = \max\{0, x'_i U_i + C_i - S_i - b'\}, \tag{3.37}$$

$$S'_{\text{sum}} = S(\vec{Y}). \tag{3.38}$$

We show

$$\begin{aligned} s > &\{\text{By Definition 3.1, Constraint Set 3.1, and Lemma 3.10, because } \vec{x} \text{ is not minimum}\} \\ &G(\vec{x},\vec{Y}) + S(\vec{Y}) \\ = &\{\text{By (3.32)}\} \\ &s'. \end{aligned} \tag{3.39}$$

We now show that the new assignment of variables satisfies Constraint Sets 3.1–3.5.

Constraint Set 3.1 holds by (3.33).

Constraint Set 3.2 holds by (3.34).

Constraint Set 3.3 holds by (3.35)–(3.37).

Constraint Set 3.4 holds by (3.34) and (3.38).

To see that Constraint Set 3.5 holds, note that

$$
\begin{aligned}
s' &= \{\text{By (3.32)}\} \\
&\quad G(\vec{x},\vec{Y}) + S(\vec{Y}) \\
&= \{\text{By the definition of } G(\vec{x},\vec{Y}) \text{ in (3.12)}\} \\
&\quad \sum_{U^+-1 \text{ largest}} (x_i U_i + C_i - S_i(Y_i)) + S(\vec{Y}) \\
&= \{\text{By Constraint Set 3.1}\} \\
&\quad \sum_{U^+-1 \text{ largest}} \left(\left(\frac{s - C_i}{m} \right) U_i + C_i - S_i(Y_i) \right) + S(\vec{Y}) \\
&> \{\text{By (3.39)}\} \\
&\quad \sum_{U^+-1 \text{ largest}} \left(\left(\frac{s' - C_i}{m} \right) U_i + C_i - S_i(Y_i) \right) + S(\vec{Y}) \\
&= \{\text{By (3.33)}\} \\
&\quad \sum_{U^+-1 \text{ largest}} (x_i' U_i + C_i - S_i(Y_i)) + S(\vec{Y}) \\
&= \{\text{By the definition of } G(\vec{x},\vec{Y}) \text{ in (3.12)}\} \\
&\quad G(\vec{x}',\vec{Y}) + S(\vec{Y}) \\
&= \{\text{By (3.35) and (3.38)}\} \\
&\quad G' + S'_{\text{sum}}.
\end{aligned}
$$

Because the new assignment satisfies Constraint Sets 3.1–3.5 with $s' < s$ (by (3.39)), the original assignment did not achieve the minimum value of s. □

The previous lemmas are sufficient to demonstrate that the minimum near-compliant vector for a system must exist. Our remaining proof obligation is to show that the minimum near-compliant vector is in fact compliant. Lemma 3.14, proved in Appendix B, demonstrates that for the optimal assignment of variables when $U^+ > 1$, s must be at least the maximum C_i in the system. We denote this maximum C_i as $C_{\max}$.

Lemma 3.14. *If $U^+ > 1$ and $s < C_{\max}$, then Constraint Sets 3.1–3.5 are infeasible.*

Lemma 3.14 is used in Lemma 3.15 below, which demonstrates that for any assignment of variables satisfying Constraint Sets 3.1–3.5, $\vec{x}$ is compliant. By Lemma 3.15, the minimum near-compliant vector must also be compliant.

Lemma 3.15. *For any assignment of variables satisfying Constraint Sets 3.1–3.5, $\vec{x}$ is compliant.*

Proof. If $U^+ = 1$, then the lemma follows trivially from Definition 3.1. We therefore assume $U^+ > 1$.

Because the assignment of variables satisfies Constraint Sets 3.1–3.5, by Lemma 3.14,

$$s \geq C_{\max}. \tag{3.40}$$

For arbitrary i,

$$\begin{aligned}
x_i &= \{\text{By Constraint Set 3.1}\} \\
&\quad \frac{s - C_i}{m} \\
&\geq \{\text{By (3.40)}\} \\
&\quad \frac{C_{\max} - C_i}{m} \\
&\geq \{\text{Because } \forall i, C_{\max} \geq C_i\} \\
&\quad 0.
\end{aligned}$$

Therefore, by Definition 3.1, $\vec{x}$ is compliant. □

Corollary 3.2. *The minimum near-compliant vector for any task system is compliant.*

3.4 Global Fair Lateness Scheduling

Having shown that a minimum compliant vector exists for any combination of feasible task system and scheduler, we now turn our attention to determining the scheduler that minimizes the maximum lateness bound for a system. As described later in Section 3.5, the LP described by Constraint Sets 3.1–3.5 can be used to select optimal relative PPs for an arbitrary linear lateness constraint. However, determining the relative PPs requires the use of an LP solver. In this section, we present a particular scheduler that has a simple expression for its relative PPs and that is optimal with respect to minimizing the maximum lateness bound for a task system.

In order to provide the best analysis for a given scheduler, we observe that some GEL schedulers are identical with respect to the scheduling decisions made at runtime, even though the CVA bounds may not be identical. We formally define and motivate this notion below in Definition 3.3 and Lemma 3.16.

Definition 3.3. Two PP assignments $\vec{Y}$ and $\vec{Y'}$ are *equivalent* if there exists a constant c such that $Y_i = Y_i' + c$ for any i. Two GEL schedulers are *equivalent* if their respective PP assignments are equivalent.

Lemma 3.16. *If two GEL schedulers are equivalent with PP assignments $\vec{Y}$ and $\vec{Y'}$, respectively, then the response time bounds derived by using $\vec{Y}$ will also apply to a system scheduled using $\vec{Y'}$, and vice versa.*

Proof. Using either $\vec{Y}$ or $\vec{Y'}$ will result in the same scheduler decisions, because each absolute PP has been increased or decreased by the same constant. □

We now define in Definition 3.4 a scheduler that, although it may not itself have the lowest maximum CVA bound, is equivalent to a scheduler that does. The value of $\vec{Y}$ in Definition 3.4 is in concise form, and is provably equivalent to a GEL scheduler with the lowest possible maximum CVA bound. A system designer can use the definition of $\vec{Y}$ in Definition 3.4 when setting scheduler parameters, and the lowest available maximum CVA bound will apply to the resulting system.

Definition 3.4. The **Global Fair Lateness (G-FL)** scheduler is the GEL scheduler with relative PP assignment

$$\forall i, Y_i = D_i - \frac{m-1}{m} C_i.$$

We first show as Theorem 3.2 below that the PPs for an arbitrary GEL scheduler can be modified to ensure that all lateness bounds are the same, without increasing the maximum lateness bound for the scheduler, and the resulting scheduler is equivalent to G-FL. We then discuss how to obtain the best lateness bounds for any scheduler equivalent to G-FL.

Theorem 3.2. *Let V be an arbitrary assignment of variables satisfying Constraint Sets 3.1–3.5. There exists an assignment V' (with each variable denoted with a prime) such that V' also satisfies Constraint Sets 3.1–3.5, the scheduler using $\vec{Y'}$ is equivalent to G-FL, and the maximum lateness bound using $\vec{x'}$ in Theorem 3.1 is no greater than using $\vec{x}$.*

Proof. By Lemma 3.15, $\vec{x}$ is compliant. Therefore, by Theorem 3.1, the maximum lateness bound for the system is

$$
\begin{aligned}
& \max\left(Y_j + x_j + C_j - D_j\right) \\
= & \{\text{By Constraint Set 3.1}\} \\
& \max\left(Y_j + \frac{s - C_j}{m} + C_j - D_j\right) \\
= & \{\text{Rearranging; observe that } s \text{ does not depend on task index } j\} \\
& \frac{s}{m} + \max\left(Y_j + \frac{m-1}{m}C_j - D_j\right) \qquad (3.41)
\end{aligned}
$$

We present the following assignment of variables for V':

$$Y_i' = \max\left(Y_j + \frac{m-1}{m}C_j - D_j\right) - \frac{m-1}{m}C_i + D_i \qquad (3.42)$$

$$s' = s, \qquad (3.43)$$

$$\forall i, x_i' = \frac{s' - C_i}{m}, \qquad (3.44)$$

$$\forall i, S_i' = S_i(Y_i'), \qquad (3.45)$$

$$G' = G(\vec{x}', \vec{Y}'), \qquad (3.46)$$

$$b' = (U^+ - 1)^{th} \text{ largest value of } x_i'U_i + C_i - S_i', \qquad (3.47)$$

$$\forall i, z_i = \max\{0, x_i'U_i + C_i - S_i' - b'\}, \qquad (3.48)$$

$$S_{\text{sum}} = S(\vec{Y}'). \qquad (3.49)$$

Consider an arbitrary task τ_i. By rearranging the following expression (which clearly holds),

$$Y_i + \frac{m-1}{m}C_i - D_i \leq \max_{\tau_j \in \tau}\left(Y_j + \frac{m-1}{m}C_j - D_j\right),$$

we have

$$Y_i \leq \max_{\tau_j \in \tau}\left(Y_j + \frac{m-1}{m}C_j - D_j\right) - \frac{m-1}{m}C_i + D_i,$$

so by (3.42),

$$Y_i \le Y_i'. \tag{3.50}$$

Therefore, by the definition of $S_i(Y_i)$ in (3.8),

$$S_i(Y_i') \le S_i(Y_i). \tag{3.51}$$

Also, by (3.43)–(3.44),

$$x_i = x_i'. \tag{3.52}$$

Constraint Set 3.1 holds for V' by (3.44).

Constraint Set 3.2 holds for V' by (3.45).

Constraint Set 3.3 holds for V' by (3.46)–(3.48).

Constraint Set 3.4 holds for V' by (3.45) and (3.49).

To see that Constraint Set 3.5 holds for V', note that

$$\begin{aligned}
s' &= \{\text{By (3.43)}\} \\
&\quad s \\
&\ge \{\text{By Constraint Set 3.5}\} \\
&\quad G + S_{\text{sum}} \\
&\ge \{\text{By Lemma 3.9}\} \\
&\quad G(\vec{x}, \vec{Y}) + S(\vec{Y}) \\
&= \{\text{By the definition of } G(\vec{x}, \vec{Y}) \text{ in (3.12), and the definition of } S(\vec{Y}) \text{ in (3.9)}\} \\
&\quad \sum_{U^+ - 1 \text{ largest}} (x_i U_i + C_i - S_i(Y_i)) + \sum_{\tau_i \in \tau} S_i(Y_i) \\
&\ge \{\text{By (3.51); observe that each } S_i(Y_i) \text{ in the first summation also appears in the second}\} \\
&\quad \sum_{U^+ - 1 \text{ largest}} \big(x_i U_i + C_i - S_i(Y_i')\big) + \sum_{\tau_i \in \tau} S_i(Y_i') \\
&= \{\text{By the definition of } S(\vec{Y}) \text{ in (3.9), the definition of } G(\vec{x}, \vec{Y}) \text{ in (3.12), and (3.52)}\} \\
&\quad G(\vec{x}', \vec{Y}') + S(\vec{Y}') \\
&= \{\text{By (3.46) and (3.49)}\}
\end{aligned}$$

$G' + S'_{\text{sum}}$.

For an arbitrary τ_i, the lateness bound under Theorem 3.1 corresponding to V' is

$$\begin{aligned}
x'_i + Y'_i + C_i - D_i &= \{\text{By (3.42) and (3.52)}\} \\
&\quad x_i + \max_{\tau_j \in \tau}\left(Y_j + \frac{m-1}{m}C_j - D_j\right) + \frac{1}{m}C_i. \\
&= \{\text{By Constraint Set 3.1}\} \\
&\quad \frac{s - C_i}{m} + \max_{\tau_j \in \tau}\left(Y_j + \frac{m-1}{m}C_j - D_j\right) + \frac{1}{m}C_i \\
&= \{\text{Cancelling}\} \\
&\quad \frac{s}{m} + \max_{\tau_j \in \tau}\left(Y_j + \frac{m-1}{m}C_j - D_j\right). \qquad (3.53)
\end{aligned}$$

By (3.41) and (3.53), the maximum lateness bound under V is identical to the maximum lateness bound under V'.

Furthermore, by Definitions 3.3 and 3.4 and (3.42), the GEL scheduler using $\vec{Y}'$ is equivalent to G-FL.

□

Theorem 3.2 demonstrates that G-FL is equivalent to a GEL scheduler with a lateness bound under Theorem 3.1 no greater than that of any other GEL scheduler. We would like to determine the "best" equivalent scheduler to G-FL so that we can obtain the smallest bounds applicable to G-FL. Towards that end, in the next lemma, we show that lateness bounds from equivalent schedulers can be compared in a straightforward manner.

Lemma 3.17. *Suppose two PP assignments $\vec{Y}$ and $\vec{Y}'$ are equivalent and denote their corresponding lateness bounds, when using minimum near-compliant vectors $\vec{x}$ and $\vec{x}'$, as $\vec{L}$ and $\vec{L}'$. There is a constant k such that, for all i, $L_i = L'_i + k$. Each L_i and L'_i differ by a system-wide constant.*

Proof.

$$\begin{aligned}
L_i - L'_i &= \{\text{By Theorem 3.1 and Corollary 3.2}\} \\
&\quad x_i + Y_i + C_i - D_i - (x'_i + Y'_i + C_i - D_i)
\end{aligned}$$

$$
\begin{aligned}
&= \{\text{By Corollary 3.1, for some } s \text{ and } s'\}\\
&\quad \frac{s-C_i}{m}+Y_i+C_i-D_i-\left(\frac{s'-C_i}{m}+Y_i'\right)\\
&= \{\text{Cancelling}\}\\
&\quad \frac{s}{m}+Y_i-\left(\frac{s'}{m}+Y_i'\right)\\
&= \{\text{Rearranging}\}\\
&\quad \frac{s-s'}{m}+(Y_i-Y_i')\\
&= \{\text{By Definition 3.3}\}\\
&\quad \frac{s-s'}{m}+c. \qquad (3.54)
\end{aligned}
$$

Because s, s', and c do not depend on task index i, we let $k = \frac{s-s'}{m}+c$, and the lemma follows. □

We now describe additional constraint sets that can be used with Constraint Sets 3.1–3.5 in order to determine the lateness bounds under the equivalent scheduler to GEL with the smallest lateness bounds.

Constraint Set 3.6. *The constraints to ensure that $\vec{Y}$ meets our assumption in* (3.7) *are*

$$\forall i, Y_i \geq 0.$$

In order to determine the best GEL scheduler equivalent to G-FL, we add an auxillary variable c and the following constraint. Because c can be any arbitrary value, this constraint specifies that any $\vec{Y}$ equivalent to G-FL is acceptable.

Constraint Set 3.7. *The constraints allowing any $\vec{Y}$ equivalent to G-FL are*

$$\forall i, Y_i = c + D_i - \frac{m-1}{m}C_i.$$

By minimizing s under Constraint Sets 3.1–3.7, we can obtain the desired lateness bounds, and therefore the best maximum lateness bound under any GEL scheduler.

3.5 Alternate Optimization Criteria

G-FL was proven in Section 3.4 to be optimal relative to the specific criterion of minimizing maximum lateness bound under CVA. Under G-FL, the system implementer does not need to use an LP solver to define PPs but instead can assign PPs using Definition 3.4. For G-FL, the LP solver is only necessary in order to analyze the lateness bounds of the system. In this section, we show how to use linear programming in order to achieve alternative lateness criteria. For example, we show how to minimize *average* lateness, or to minimize maximum proportional lateness. In order to achieve these criteria, it is necessary to use a set of PPs that *differ* from G-FL, and the system implementer must use the LP solver to determine the PPs.

Next we show how Constraint Sets 3.1–3.6 can be coupled with objective functions, and possibly additional constraint sets, to find optimal priority point settings under CVA with respect to alternative criteria. We define several different schedulers based on their lateness criteria. In all cases, our criterion is to minimize some lateness metric, such as maximum lateness. We will denote each criterion with two letters indicating the type of lateness to be minimized. The first is "*A*" for *average* or "*M*" for *maximum*, and the second is "*L*" for *lateness* or "*P*" for *proportional lateness*. No definition is provided for ML (maximum lateness), because G-FL optimizes the same criterion, as discussed above. Where two criteria are provided, the first is optimized, then the second. For example, ML-AL below minimizes the average lateness *subject to having the smallest maximum lateness possible*.

Minimizing Maximum Proportional Lateness: MP. As described in Section 3.4, G-FL has the smallest maximum lateness bound for any GEL scheduler under CVA. However, for some applications, more tardiness may be acceptable for tasks that have longer relative deadlines. Therefore, it may be desirable to minimize the maximum *proportional* lateness, rather than the maximum lateness.

In order to minimize maximum proportional lateness, we define an auxiliary variable $I_{\max}$ that corresponds to the maximum proportional lateness for the task system. We add a set of constraints to ensure the appropriate value for $I_{\max}$ and then minimize it.

$$\begin{array}{lr} \text{Minimize:} & I_{\max} \\ \text{Subject to:} & \forall i : (Y_i + x_i + C_i - D_i)/D_i \leq I_{\max} \\ & \text{Constraint Sets 3.1–3.6} \end{array}$$

Minimizing Average Lateness: AL. G-FL guarantees the smallest maximum lateness bound available under CVA. However, depending on the nature of the application, it may be more desirable to obtain the smallest average lateness bound, rather than the maximum.

The following LP may be solved to minimize average lateness under CVA.

$$\begin{array}{ll} \text{Minimize:} & \displaystyle\sum_{\tau_i \in \tau} Y_i + x_i \\ \text{Subject to:} & \text{Constraint Sets 3.1–3.6} \end{array}$$

Note that average lateness is given by $\sum_{\tau_i \in \tau}(Y_i + x_i + C_i - D_i)/n$, but C_i, D_i, and n are all constants that are not necessary to include in the optimization objective.

While AL is optimal with respect to average lateness, as is shown experimentally in Section 3.6, the lateness of some tasks may be larger than the maximum lateness bound of G-FL, which we denote $L_{\max}$. Next, we show how to optimize the average lateness of all tasks while maintaining a maximum lateness no greater than $L_{\max}$.

Minimizing Average Lateness from G-FL: ML-AL. Although G-FL provides the smallest maximum lateness bound available under CVA, it does so by giving all tasks the same lateness bound. It may be possible to reduce the lateness bounds for some tasks without altering the maximum lateness bound for the system. Therefore, if the requirement to run an offline LP solver to determine PPs is not problematic, further optimizing the lateness bounds can be desirable. Here we show how to minimize the average lateness bound for the system with respect to having the same maximum lateness bound as G-FL.

The following LP may be solved to minimize the average lateness under CVA while maintaining the same maximum lateness bound as G-FL.

$$\begin{array}{ll} \text{Minimize:} & \displaystyle\sum_{\tau_i \in \tau} Y_i + x_i \\ \text{Subject to:} & \forall i : Y_i + x_i + C_i - D_i \leq L_{\max},^{2} \\ & \text{Constraint Sets 3.1–3.6} \end{array}$$

As before, the constants C_i, D_i and n are omitted from the objective function.

[2] Application-specific per-task lateness tolerances could be used instead of $L_{\max}$.

Minimizing Average Proportional Lateness: AP. For applications where performance is more sensitive to the average lateness rather than the maximum lateness, but where tasks with longer deadlines can permit more tardiness, it may be desirable to minimize average proportional lateness.

The LP for minimizing average lateness can be modified in a straightforward manner to minimize average proportional lateness.

$$\begin{array}{ll} \text{Minimize:} & \sum_{\tau_i \in \tau} (x_i + Y_i)/D_i \\ \text{Subject to:} & \text{Constraint Sets 3.1–3.6} \end{array}$$

As was the case with average lateness, unnecessary constant terms have been omitted from the objective function.

Minimizing Average Proportional Lateness from Smallest Maximum Proportional Lateness: MP-AP. Just as it is desirable to reduce average lateness when it is possible to do so without increasing maximum lateness, it is desirable to reduce average proportional lateness when it is possible to do so without increasing maximum proportional lateness.

As we did with average lateness constrained by the maximum lateness from G-FL, we can also minimize the average proportional lateness constrained by the maximum proportional lateness from MP.

$$\begin{array}{ll} \text{Minimize:} & \sum_{\tau_i \in \tau} (x_i + Y_i)/D_i \\ \text{Subject to:} & \forall i : (Y_i + x_i + C_i - D_i)/D_i \leq I_{\max},^{3} \\ & \text{Constraint Sets 3.1–3.6} \end{array}$$

Once again, unnecessary constant terms have been omitted from the objective function.

We note that the LP formulation of CVA can be used and extended to other optimization objectives, perhaps most notably, application-specific optimization objectives. For example, an LP solver can be used to assign PPs to ensure application-specific lateness tolerances are satisfied (if possible under CVA), or to maximize total system utility under some linear definitions of lateness-based utility.

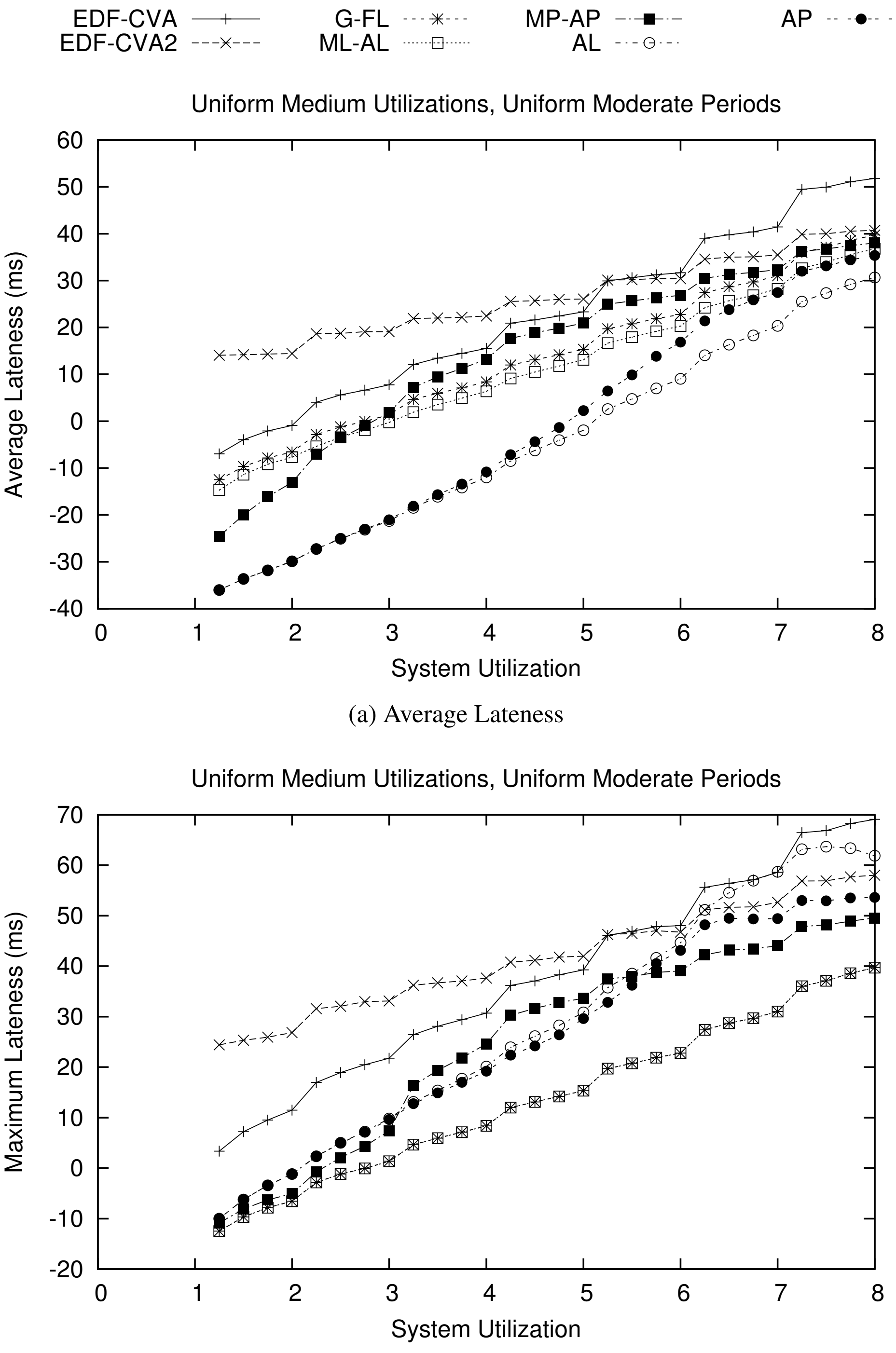
[3] As before, application-specific per-task proportional lateness tolerances could be used instead of $I_{\max}$.

3.6 Experiments

In this section, we present experiments that demonstrate how G-FL and the LP-based schedulers described in Section 3.5 can improve lateness bounds over existing scheduling algorithms. In these experiments, we evaluated the lateness bounds of randomly generated task systems. We generated random task sets using a similar experimental design as in previous studies (e.g., (Bastoni et al., 2011)). We generated implicit-deadline task sets in which per-task utilizations were distributed either uniformly or bimodally. For task sets with uniformly distributed utilizations, per-task utilizations were chosen to be *light*, *medium* or *heavy*, which correspond to utilizations uniformly distributed in the range $[0.001, 0.1]$, $[0.1, 0.4]$, or $[0.5, 0.9]$, respectively. For task systems with bimodally distributed utilizations, per-task utilizations were chosen from either $[0.001, 0.5]$, or $[0.5, 0.9]$ with respective probabilities of $8/9$ and $1/9$, $6/9$ and $3/9$, or $4/9$ and $5/9$. The periods of all tasks were generated using an integral uniform distribution between $[3\text{ ms}, 33\text{ ms}]$, $[10\text{ ms}, 100\text{ ms}]$ and $[50\text{ ms}, 250\text{ ms}]$ for tasks with *short*, *moderate*, and *long* periods, respectively. We considered a system with $m = 8$ processors, as clustered scheduling typically is preferable to global scheduling for larger processor counts (Brandenburg, 2011). For each per-task utilization and period distribution combination, 1,000 task sets were generated for each total system utilization value in $\{1.25, 1.50, \ldots, 8.0\}$. We did not consider task systems with utilizations of one or less, as they are schedulable on one processor with no tardiness.

For each generated task system, we evaluated the average and maximum per-task lateness bounds under Devi and Anderson's analysis of G-EDF (Devi and Anderson, 2008) (EDF-DA), CVA analysis of G-EDF scheduling by selecting the best equivalent scheduler as for G-FL in Section 3.4 (EDF-CVA), CVA analysis of G-EDF using an alternative optimization rule[4] derived from Devi and Anderson (2008) (EDF-CVA2), G-FL, and most of the LP-based schedulers discussed in Section 3.5 (ML-AL, AP, and MP-AP). We do not evaluate MP because MP-AP is preferable. We present representative results in Figures 3.6–3.9. In those figures, we show the mean average and maximum lateness bounds for each total system utilization value over all generated task systems. Note that the lateness-bound results are analytical, and that in an actual schedule observed latenesses may be smaller.

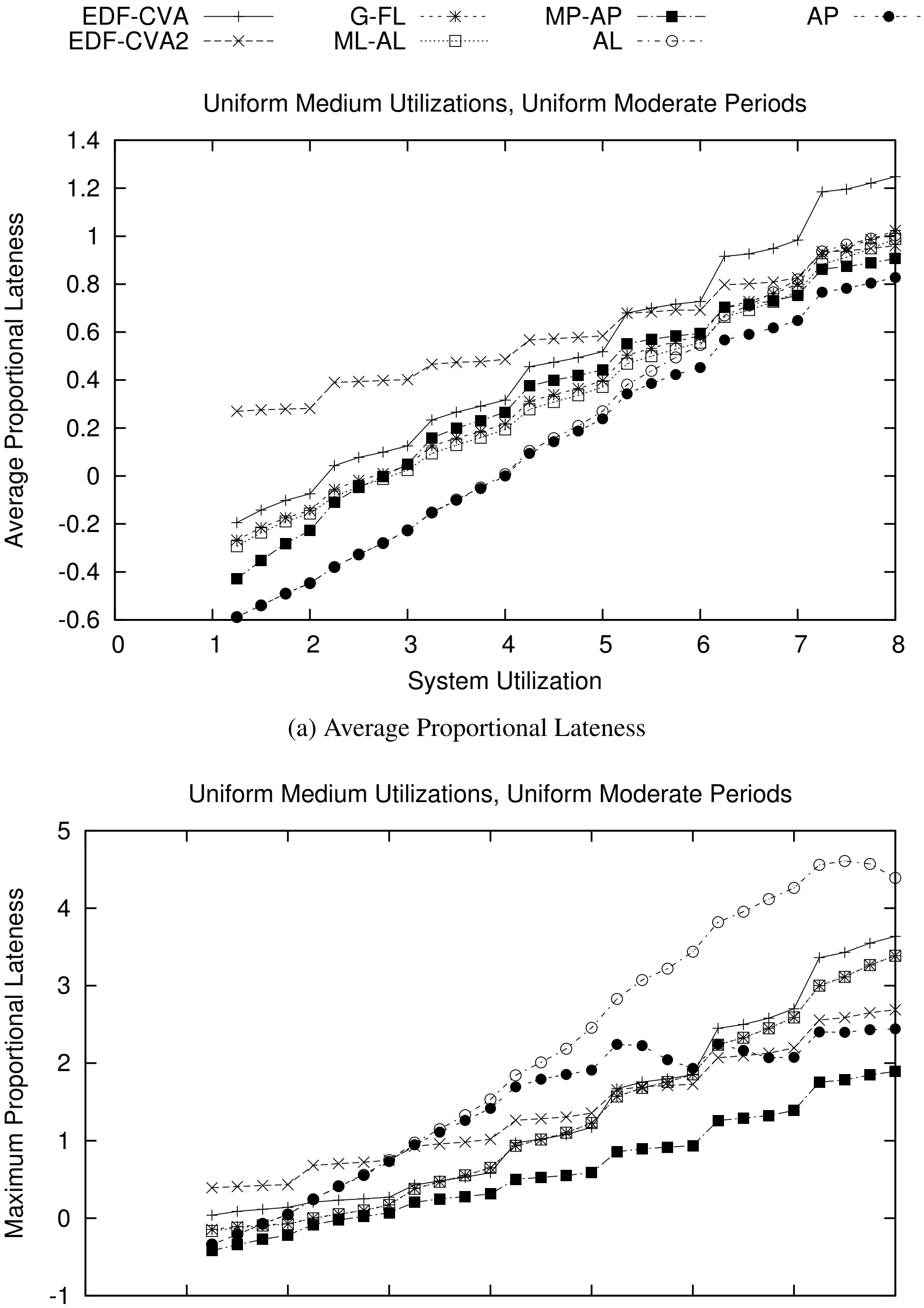
[4] When $Y_i = T_i$ for all i, $S_i(Y_i) = 0$, and rather than defining $G(\vec{x}, \vec{Y})$ as the largest sum of $U^+ - 1$ values of $x_i U_i + C_i$, we can instead define $G(\vec{x}, \vec{Y})$ as the largest sum of only $U^+ - 2$ values of $x_i U_i + C_i$ plus an additional C_i.



(a) Average Lateness

(b) Maximum Lateness

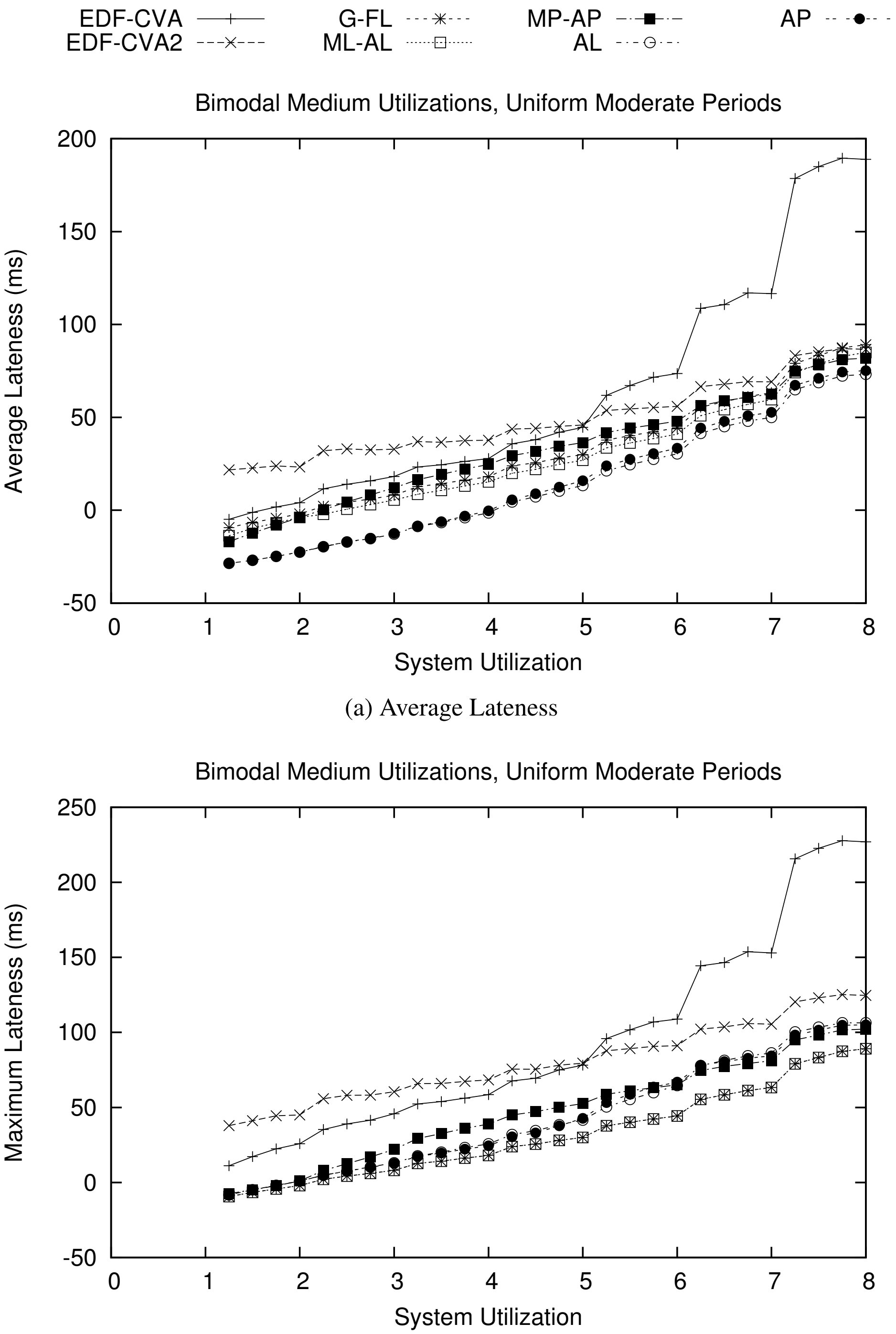
Figure 3.6: **(a)** Average and **(b)** maximum lateness bound with respect to system utilization for task systems with uniform medium utilizations and uniform moderate periods.



(a) Average Proportional Lateness

(b) Maximum Proportional Lateness

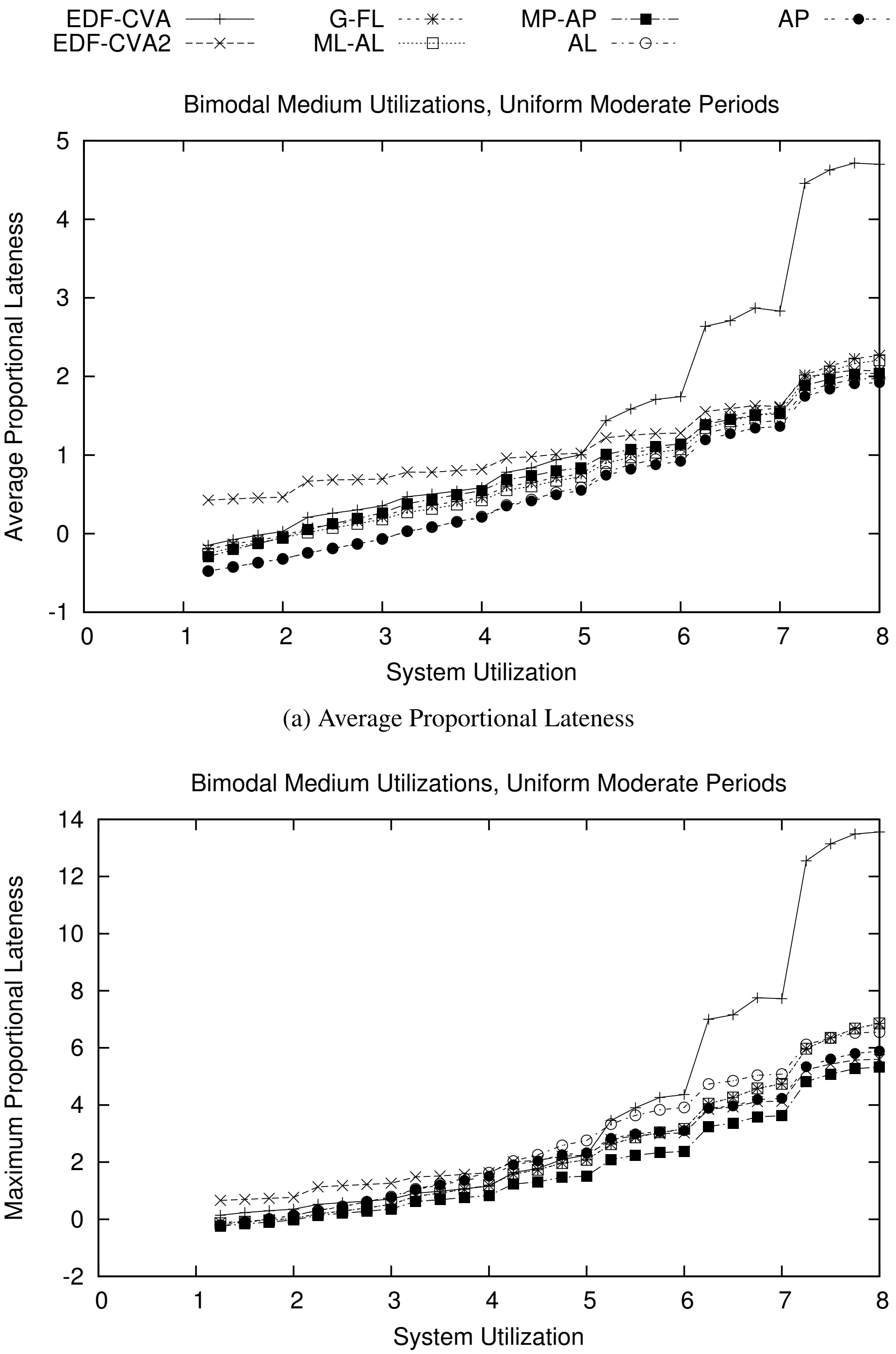
Figure 3.7: **(a)** Average and **(b)** maximum proportional lateness bound with respect to system utilization for task systems with uniform medium utilizations and uniform moderate periods.



(a) Average Lateness

(b) Maximum Lateness

Figure 3.8: **(a)** Average and **(b)** maximum lateness bound with respect to system utilization for task systems with bimodal medium utilizations and uniform moderate periods.



(a) Average Proportional Lateness

(b) Maximum Proportional Lateness

Figure 3.9: **(a)** Average and **(b)** maximum proportional lateness bound with respect to system utilization for task systems with bimodal medium utilizations and uniform moderate periods.

Observation 3.1. *For task systems will small utilizations, the "equivalent scheduler" optimization in EDF-CVA outperforms the optimization in EDF-CVA2; however, the converse is true for large utilizations.*

This can be seen in Figures 3.6–3.9. Although the alternative optimization in EDF-CVA2 performs very well for task systems with large utilizations, it is only applicable if $Y_i = T_i$, for all i. Therefore, it is likely to not be useful for systems with deadlines significantly different from minimum separation times. The techniques proposed here are applicable to any GEL scheduler and are fully compatible with arbitrary deadlines.

Observation 3.2. *All of the GEL schedulers we considered with PPs different from G-EDF, had smaller average lateness bounds than G-EDF.*

The GEL schedulers we considered with PPs different from G-EDF typically had smaller average proportional lateness bounds than G-FL.

All these scheduling algorithms optimize, with respect to CVA, either the average or maximum (proportional) lateness of all tasks by moving PPs. Therefore, these algorithms should have smaller average lateness bounds than G-EDF. This can be observed in in part (a) of Figures 3.6–3.9.

Observation 3.3. *The maximum lateness bounds of ML-AL and G-FL are the same, but the average lateness bound of ML-AL is at worst the average lateness bound of G-FL.*

Based on the constraints and the optimization objective of ML-AL, the average and maximum lateness bounds are provably no worse than G-FL. As is seen in Figures 3.6 and 3.8, the improvement in average lateness in the task systems seen in our experiments was usually only a few ms.

Observation 3.4. *Average lateness bounds were lower under AL than under G-FL and ML-AL. This improvement in average lateness is made possible by allowing for increased maximum lateness.*

Average proportional lateness bounds were lower under AP than under MP-AP. This improvement in average proportional lateness is made possible by allowing for increased maximum proportional lateness.

As a result of the LP objective function, AL is optimal with respect to average lateness under CVA. In Figures 3.6 and 3.8, we see that the average lateness bound of AL is always smaller than all other schedulers, often by 10-20 ms or more. However, the maximum lateness bounds of AL are larger than G-FL and ML-AL. In most observed cases, lateness bounds of AL were less than or commensurate with G-EDF lateness bounds as determined by either CVA or Devi and Anderson's analysis, though in some cases the maximum lateness was greater than G-EDF by 10-20 ms. From these results, AL may be practical in many applications.

Similarly, due to its optimization criteria, AP is optimal with respect to average proportional lateness under CVA, so its maximum is typically several deadlines shorter than other schedulers. In almost all cases, AP had a lower average proportional lateness than all other schedulers[5].

We note that the lateness bounds of AL in comparison to G-FL and ML-AL, and the bounds of AP in comparison to MP-AP, demonstrate that the LP solver has considerable flexibility in choosing priority points to optimize for certain lateness criteria. If some tasks have larger lateness tolerances than others, the PPs of the more tolerant tasks can be increased to improve the lateness bounds of the less tolerant tasks. This gives system designers much more flexibility to optimize the scheduler for application-specific needs.

Observation 3.5. *Benefits to average or maximum proportional lateness come at a cost to average or maximum lateness, and vice versa.*

If two tasks have the same lateness bound but different relative deadlines, the task with the smaller relative deadline will have larger proportional lateness. The reverse is true if the proportional lateness bounds are the same. Therefore, improving lateness can worsen proportional lateness or vice versa. This effect is particularly strong when deadlines of different tasks have a large variance. This can be seen in all figures.

3.7 Conclusion

We have demonstrated that G-FL provides maximum absolute lateness bounds no larger than those currently available for G-EDF and have provided experimental evidence that G-FL is superior to G-EDF with regards to maximum lateness bounds. Furthermore, G-FL provides equal lateness bounds for all tasks in the system, and therefore provides a closer relationship between deadlines and response time bounds than G-EDF currently does. The implementation of G-FL is identical to that of G-EDF with arbitrary deadlines, and G-FL maintains the desirable JLSP property (enabling known locking techniques.) Therefore, G-FL is a better choice than G-EDF for SRT systems.

[5] With uniform heavy utilizations and uniform long periods, EDF-CVA2 provided a very slightly smaller average proportional latenesss bound when the system utilization was 8. Recall that the technique used by EDF-CVA2 only applies when relative PPs equal minimum separation times.

We have also demonstrated that LP techniques can be used to optimize for other linear constraints beyond minimizing maximum lateness bounds. These techniques can dominate G-FL at the cost of an offline computation step.

CHAPTER 4: REMOVING INTRA-TASK PRECEDENCE CONSTRAINTS[1]

In this chapter, we provide response-time bounds for a class of workloads in which jobs of the same task are independent of each other and can be executed in parallel, such as servers handling independent requests. In addition to providing analysis that applies to arbitrary GEL schedulers, we show that lateness under G-EDF is greatly lessened if jobs of the same task are not constrained to execute in sequence. We show this by deriving per-job response-time bounds, from which lateness bounds can be deduced. After deriving such bounds, we compare them experimentally to prior bounds, which were derived assuming no intra-task parallelism.

The remainder of this chapter is organized as follows. In Section 4.1, we formally present the task model assumed in this chapter. Then, in Section 4.2, we describe compliant vectors for this task model, like those described in Chapter 3. Afterward, in Section 4.3, we prove that each task system has a unique minimum compliant vector, as was the case in Chapter 3. We then provide in Section 4.4 an algorithm to compute the minimum compliant vector, and in Section 4.5, we provide experimental evidence that removing precedence constraints can significantly reduce lateness bounds. We conclude in Section 4.6.

4.1 System Model

In the task model discussed in Section 1.1.1, successive jobs of each task were required to execute in sequence. This constraint arises naturally when jobs correspond to separate invocations of the same code segment, as discussed in Section 1.1.1. However, in some settings, jobs are released as separate threads in response to interrupts, in which case, successive jobs of the same task may execute concurrently. In prior hard real-time analysis of G-EDF by Baker and Baruah (2009), the impact of such concurrently executing jobs has been considered, but to our knowledge, no such analysis exists for SRT systems for which bounded lateness is acceptable.

[1] This work appeared in preliminary form in the following paper:

Erickson, J. P. and Anderson, J. H. (2011). Response time bounds for G-EDF without intra-task precedence constraints. In *Proceedings of the 15th International Conference on Principles of Distributed Systems*, pages 128–142.

Our work assumes that jobs are independent. However, in the presence of a locking protocol, a technique similar to that discussed in Section 2.2.2 could be used. In other words, job parameters could be inflated to account for delays due to locking, and the system with inflated parameters could be used with the techniques described in this chapter. New accounting similar to that in Section 2.2.2 might be necessary, but such accounting is left as future work.

Compared to the task model described in Section 1.1.1, we make two modifications. First, as implied by the discussion above, successive jobs of the same task are allowed to execute in parallel. Second, *early release* behavior (Anderson and Srinivasan, 2000) is allowed: a job may have an *actual release* (*a-release*) *time* that is earlier than its *scheduler release* (*s-release*) *time*. A job's deadline and PP are defined based on its s-release time, and consecutive s-releases of each task τ_i are still constrained to be no closer than T_i time units apart. However, a job may begin execution as early as its a-release time. These changes to the traditional sporadic model allow us to support general event models, as the following example illustrates.

Example. In high-frequency trading systems, short response times are critical to minimize risk (Durbin, 2010). Consider such a system that responds to data from the market about two stocks. One stock is highly critical and should receive new information every 2 ms (but due to network uncertainty may not be timed precisely). It may take up to 3 ms to process and should be processed as quickly as possible, so its deadline is 3 ms. Observe that this stock overutilizes a single processor and could not be supported using the traditional sporadic task model, even on a multiprocessor. However, it can be supported using the methodology provided in this chapter. A second stock is less critical, should receive new information every 4 ms, and can take up to 2 ms to process. One possible execution under G-EDF ($Y_i = D_i$) on two processors is depicted in Figure 4.1. Observe that the a-release times sometimes do occur before the s-release times (because incoming packets can arrive early or late) and that some jobs do miss deadlines.

In the task model considered here, no job may run concurrently with itself, but distinct jobs within the same task may run concurrently. For convenience, we define the *task system utilization* $U(\tau) \triangleq \sum_{\tau_i \in \tau} U_i$, and $U^+ \triangleq \lceil U(\tau) \rceil$.

Under the task model described in Section 1.1.1 with implicit precedence constraints, providing bounded response time required that no τ_i had $U_i > 1$ and that $U(\tau) \leq m$, as in Chapter 3. However, under the task model considered here, a job with $U_i > 1$ can have bounded response time if subsequent invocations run on separate processors, as depicted for τ_1 in Figure 4.1. $U(\tau) \leq m$ remains necessary so that the entire system

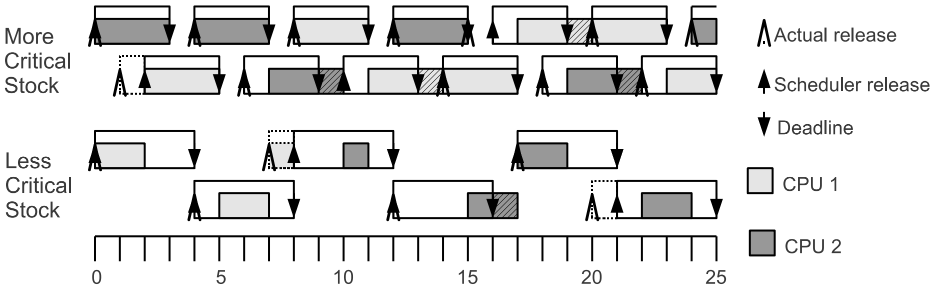


Figure 4.1: Example high-frequency trading system scheduled with G-EDF.

is not overutilized. In this work we demonstrate that $U(\tau) \leq m$ is also a sufficient condition for bounded response times and provide response-time bounds relative to the s-release time of each job.

4.2 Response Time Characterization

Over an interval of any given length Δ, the total amount of work from jobs of τ_i (with both s-release times and PPs inside the interval) is bounded. The *same* demand-bound function $\text{DBF}(\tau_i, Y_i, \Delta)$ used in Chapter 3 continues to apply to our broader task model, as long as we consider the release time of each job to be its s-release. If a job actually has an a-release prior to the beginning of an interval but an s-release within an interval, that can only reduce the demand within the interval compared to the situation in which the job's a-release time was equal to its s-release time.

We will use Lemma 3.2 to provide an upper bound on $\text{DBF}(\tau_i, Y_i, \Delta)$. We state that lemma again here for convenience. (Recall that, by (3.8), $S_i(Y_i) \triangleq C_i \cdot \max\{0, 1 - Y_i/T_i\}$.)

Lemma 3.2. *$\forall \Delta \geq 0, DBF(\tau_i, Y_i, \Delta) \leq U_i\Delta + S_i(Y_i)$.*

For an n-task system τ, we wish to define a vector of non-negative real numbers $\langle x_1, x_2, \ldots x_n \rangle$ such that the response time of each task τ_i, $1 \leq i \leq n$, is at most $x_i + C_i$ when τ is scheduled using a GEL scheduler on m unit-speed processors. Each x_i value depends upon the other x_i values. Therefore, we initially define the vectors using an implicit criterion, and as in Chapter 3 we define the notion of a "compliant vector" as one that meets this criterion.

Definition 4.1. For each task τ_i, non-negative integer $p < U^+ - 1$, and non-negative real number x_i, let

$$g(\tau_i, x_i, p) \triangleq \min\{C_i,\ \max\{0,\ x_i + C_i - pT_i\}\}. \tag{4.1}$$

For any $\vec{x} \triangleq \langle x_1, x_2, \ldots, x_n \rangle$, an ordered list of n non-negative real numbers, let

$$G(\vec{x}) \triangleq \sum_{U^+ - 1 \text{ largest}} g(\tau_i, x_i, p), \tag{4.2}$$

$$S(\vec{Y}) \triangleq \sum_{\tau_i \in \tau} S_i(Y_i). \tag{4.3}$$

We define $\vec{x}$ as a *compliant vector* if and only if

$$\frac{G(\vec{x}) + S(\vec{Y}) + U(\tau)Y_i - C_i}{m} \leq x_i \tag{4.4}$$

is satisfied for all i, $1 \leq i \leq n$.

Observe that, unlike $G(\vec{x}, \vec{Y})$ as defined in (3.12), $G(\vec{x})$ as defined in (4.2) does not depend on $\vec{Y}$. However, the definition of $S(\vec{Y})$ in (4.3) is identical to that in (3.9). As compared to (3.13), in (4.4) we also add the term $U(\tau)Y_i$ in the numerator. This term is necessary because we are considering response-time bounds instead of lateness bounds.

We now derive a response-time bound by considering a compliant vector $\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$ and an arbitrary collection H' of jobs generated by τ. We order jobs by PP with ties broken arbitrarily (as per standard GEL scheduling). We analyze the response time of an arbitrary job $\tau_{i,k}$ with s-release time $r_{i,k}$ and PP $y_{i,k}$, assuming that each $\tau_{j,\ell}$ ordered prior to $\tau_{i,k}$ completes within $(C_j + x_j)$ units of its s-release time. We denote as H the set of all jobs ordered at or before $\tau_{i,k}$, which (by the definition of GEL scheduling) contains all jobs that affect the scheduling of $\tau_{i,k}$. We also denote $H_c \triangleq H \setminus \{\tau_{i,k}\}$ (i.e., the work *c*ompeting with $\tau_{i,k}$).

We denote as $W_j(t)$ the remaining execution for jobs in H of task τ_j at time t, and let $W(t) \triangleq \sum_{\tau_j \in \tau} W_j(t)$. Furthermore, we define an interval as *busy* if at least U^+ processors are executing work throughout the interval, and *nonbusy* otherwise. We define t_b (*b*usy) as the earliest time such that $[t_b, r_{i,k})$ is continuously busy. At the latest, $t_b = r_{i,k}$, in which case the interval is empty. Observe that, unlike in Chapter 3, the busy interval ends at $r_{i,k}$ rather than $y_{i,k}$.

In Lemma 4.1 below, we will bound $W(t_b)$. Then, in Lemma 4.2 below, we will use that result to bound $W(r_{i,k})$. Then, in Lemma 4.3 we will provide a response-time bound for $\tau_{i,k}$, and in Theorem 4.1, we will prove by induction that this bound is correct for all $\tau_{i,k}$.

Lemma 4.1. *If $\vec{x}$ is a compliant vector and the response time of each $\tau_{j,\ell} \in H_c$ is at most $x_j + C_j$, then*

$$W(t_b) \leq U(\tau)(y_{i,k} - t_b) + G(\vec{x}) + S(\vec{Y}).$$

Proof. We will say that a job $\tau_{j,\ell}$ is "executing at time instant t_b^-" if there is an ε greater than 0 such that $\tau_{j,\ell}$ is executing over the entire interval $[t_b - \varepsilon, t_b)$. In Chapter 3, the presence of an idle CPU implied that at most $U^+ - 1$ *tasks* have work available for execution at time instant t_b^-, whereas here the same condition implies that at most $U^+ - 1$ *jobs* are available for execution. In Chapter 3 it was necessary to account for released jobs that were not running due to a precedence constraint, despite the presence of an idle CPU. In order to do so, assuming that $U_i \leq 1$ for each τ_i was necessary. Here we do not need to account for such a case, but do need to account for the fact that several jobs running in a non-busy interval could be from the same task. The assumption that $U_i \leq 1$ is no longer necessary.

We now consider two categories of jobs that may contribute to $W(t_b)$: jobs that have s-release times before t_b and are executing at time instant t_b^- (Category 1) and jobs that have s-release times at or after t_b (Category 2). Because there is an idle processor at time instant t_b^-, if a job has an s-release time before t_b but is not executing at time instant t_b^-, then it has already completed by t_b and does not contribute to $W(t_b)$.

Category 1: Jobs With S-release Times Before t_b Executing at t_b^-. By the definition of t_b, there may be at most $U^+ - 1$ jobs executing at time instant t_b^-. We consider the jobs of each task τ_j that has jobs with s-release times before t_b executing at time instant t_b^-. We will use p (*p*eriod) to index each executing job relative to the job with the most recent non-future s-release within τ_j: $p = 0$ indicates the job with the most recent s-release, $p = 1$ the next most recent s-release, etc. By the assumption of the lemma, if $p > 0$ for some $\tau_{j,\ell}$, then $\tau_{j,\ell}$ must complete by $x_j + C_j$ units after its s-release time, and must have an s-release time before $t_b - pT_j$. Therefore, $\tau_{j,\ell}$ must complete by time $t_b + x_j + C_j - pT_j$, and its contribution to $W_j(t_b)$ is at most $\min\{C_j, \max\{0, x_j + C_j - pT_j\}\} \overset{\text{By (4.1)}}{=} g(\tau_j, x_j, p)$.

When $p = 0$ for some $\tau_{j,\ell}$, $x_j + C_j - pT_j \geq C_j$. Therefore $g(\tau_j, x_j, p) = C_j$ by (4.1), so $\tau_{j,\ell}$'s contribution to $W_j(t_b)$ is also at most $g(\tau_j, x_j, p)$.

Category 2: Jobs With S-release Times at or After t_b. We now consider jobs with s-release time at or after t_b. By Lemma 3.2, each task τ_j contributes at most $U_j(y_{i,k} - t_b) + S_j$ units of work over $[t_b, y_{i,k})$. Cumulatively, all tasks contribute at most $U(\tau)(y_{i,k} - t_b) + S(\vec{Y})$ units of work over $[t_b, y_{i,k})$.

Total Remaining Work. $W(t_b)$ contains at most $U^+ - 1$ jobs from Category 1, in addition to all jobs from Category 2, so $W(t_b) \leq U(\tau)(y_{i,k} - t_b) + S(\vec{Y}) + G(\vec{x})$. □

We now provide a bound on $W(r_{i,k})$.

Lemma 4.2. *If $\vec{x}$ is a compliant vector and the response time of each $\tau_{j,\ell} \in H_c$ is at most $x_j + C_j$, then*

$$W(r_{i,k}) \leq U(\tau)Y_i + G(\vec{x}) + S(\vec{Y}).$$

Proof. We have

$$\begin{aligned}
W(r_{i,k}) &= \{\text{Because at least } U^+ \text{ CPUs are busy in } [t_b, r_{i,k})\} \\
&\quad W(t_b) - U^+ \cdot (r_{i,k} - t_b) \\
&\leq \{\text{Because } U^+ \geq U(\tau)\} \\
&\quad W(t_b) - U(\tau) \cdot (r_{i,k} - t_b) \\
&\leq \{\text{By Lemma 4.1}\} \\
&\quad U(\tau)(y_{i,k} - t_b) + G(\vec{x}) + S(\vec{Y}) - U(\tau) \cdot (r_{i,k} - t_b) \\
&\leq \{\text{Rewriting}\} \\
&\quad U(\tau)(y_{i,k} - r_{i,k}) + G(\vec{x}) + S(\vec{Y}) \\
&= \{\text{By the definition of } Y_i\} \\
&\quad U(\tau)Y_i + G(\vec{x}) + S(\vec{Y}).
\end{aligned}$$

□

We now use the previous lemma to bound the response time of a job under the same assumptions.

Lemma 4.3. *If $\vec{x}$ is a compliant vector and the response time of each $\tau_{j,\ell} \in H_c$ is at most $x_j + C_j$, then the response time of $\tau_{i,k}$ is at most $x_i + C_i$.*

Proof. After $r_{i,k}$, $\tau_{i,k}$ is continuously running until it is finished, except when all other CPUs are occupied by jobs from H_c. Recall that, by definition, $W(r_{i,k})$ is the total remaining work after time $r_{i,k}$ for jobs in H. We define $W_c(r_{i,k})$ as the total amount of remaining work after time $r_{i,k}$ for jobs in H_c. Because the upper bound in Lemma 4.2 assumes that all jobs (including $\tau_{i,k}$) run for their full worst-case execution times, Lemma 4.2

implies that

$$W_c(r_{i,k}) \leq U(\tau)Y_i + S(\vec{Y}) + G(\vec{x}) - C_i. \tag{4.5}$$

The total amount of time after $r_{i,k}$ during which m CPUs are busy with work from H_c can be at most

$$\begin{aligned}
\frac{W_c(r_{i,k})}{m} &\leq \{\text{By (4.5)}\} \\
&\quad \frac{G(\vec{x}) + S(\vec{Y}) + U(\tau)Y_i - C_i}{m} \\
&\leq \{\text{By (4.4)}\} \\
&\quad x_i.
\end{aligned}$$

Thus, $\tau_{i,k}$ is prevented from executing after its s-release time for at most x_i time units, so its response time is at most $x_i + C_i$. □

This lemma leads directly to the main result of this section:

Theorem 4.1. *If $\vec{x}$ is a compliant vector, then each $\tau_{i,k}$ completes within $x_i + C_i$ units of its s-release time.*

Proof. By inducting over the jobs of H' using Lemma 4.3. □

4.3 The Minimum Compliant Vector

Theorem 4.1 uses compliant vectors to express response-time bounds. Our objective is to compute response-time bounds that are as small as possible. We show that for any arbitrary-deadline sporadic task system τ without implicit precedence constraints and corresponding assignment of $\vec{Y}$, there exists a unique *minimum* compliant vector.

The analysis of the minimum compliant vector in Chapter 3 uses linear programming to compute and analyze the minimum compliant vector. Unfortunately, the definition of $g(\tau_i, x_i, p)$ in (4.1) includes both a "min" and a "max". This violates the convexity needed to use such LP techniques. Nonetheless, in Section 4.4, we do present a polynomial-time algorithm that can be used to compute the minimum compliant vector.

We first characterize the behavior of $G(\vec{x})$. We consider two vectors $\vec{x}$ and $\vec{z}$ that differ by a constant for some of their values, and are the same elsewhere. For example, $\vec{x} = \langle 1, 2, 3 \rangle$ and $\vec{z} = \langle 2, 2, 4 \rangle$ differ by exactly 1 in two places (the first and third) and are the same in the second; Lemma 4.4 would apply to $\vec{x}$ and $\vec{z}$ with $k = 2$ and $\delta = 1$.

Lemma 4.4. *Suppose length-n vectors $\vec{x}$ and $\vec{z}$ differ at exactly k values, where $k > 0$, and for these values $z_i = x_i + \delta$, where δ is a positive constant. Denote $w \triangleq \min\{k, U^+ - 1\}$.*

The following inequality holds:

$$G(\vec{x}) \leq G(\vec{z}) \leq G(\vec{x}) + \delta \cdot w. \tag{4.6}$$

Proof. We will define a *candidate sum* for $\vec{x}$ as any sum of $U^+ - 1$ distinct $g(\tau_i, x_i, p)$ values as defined in (4.1). By (4.2), $G(\vec{x})$ is the largest candidate sum for $\vec{x}$.

First, we prove $G(\vec{x}) \leq G(\vec{z})$. Consider the candidate sum N for $\vec{z}$ computed by selecting the same i and p values as in $G(\vec{x})$. Because for all $i, x_i \leq z_i$, $G(\vec{x}) \leq N$. Because $G(\vec{z})$ must be the largest candidate sum for $\vec{z}$, $N \leq G(\vec{z})$. Therefore, $G(\vec{x}) \leq G(\vec{z})$.

Next, we prove $G(\vec{z}) \leq G(\vec{x}) + \delta \cdot w$ by contradiction. Suppose $G(\vec{z}) > G(\vec{x}) + \delta \cdot w$. Consider the candidate sum T for $\vec{x}$ computed by selecting the same i and p values as in $G(\vec{z})$. Observe that at most w terms contribute to the difference between $G(\vec{z})$ and T. When two such terms differ, we have $x_i = z_i - \delta$ ($x_i = z_i$ otherwise). Thus, $T \geq G(\vec{z}) - \delta \cdot w$, and hence, $T > G(\vec{x})$, which contradicts the fact that $G(\vec{x})$ is a *maximal* candidate sum for $\vec{x}$. □

We say that length-n $\vec{x}$ is *strictly smaller* than length-n $\vec{z}$ if for all $i, x_i \leq z_i$ and there exists a j such that $x_j < z_j$. Clearly $\vec{z}$ cannot be considered "minimum" if there exists such an $\vec{x}$. We next use Lemma 4.4 to characterize the minimum compliant vector.

Lemma 4.5. *If $\vec{z}$ is compliant and there is a j such that $z_j > (G(\vec{z}) + S(\vec{Y}) + U(\tau)Y_i - C_i)/m$, then there exists a strictly smaller vector $\vec{x}$ that is also compliant.*

Proof. Define $\vec{x}$ such that $x_i = z_i$ for $i \neq j$, and

$$x_j = \frac{G(\vec{z}) + S(\vec{Y}) + U(\tau)Y_j - C_j}{m}. \tag{4.7}$$

In this case, $\vec{x}$ and $\vec{z}$ are of the form of Lemma 4.4 with $k = 1$ and $\delta = z_j - (G(\vec{z}) + S(\vec{Y}) + U(\tau)Y_j - C_j)/m$. Therefore, $G(\vec{x}) \leq G(\vec{z})$.

We now have for all $i \neq j$,

$$\begin{aligned}
&\frac{G(\vec{x}) + S(\vec{Y}) + U(\tau)Y_i - C_i}{m} \\
\leq\ &\{\text{Since } G(\vec{x}) \leq G(\vec{z})\} \\
&\frac{G(\vec{z}) + S(\vec{Y}) + U(\tau)Y_i - C_i}{m} \\
\leq\ &\{\text{Since } \vec{z} \text{ is compliant, by (4.4)}\} \\
&z_i \\
=\ &x_i.
\end{aligned}$$

Also, by construction,

$$\begin{aligned}
&\frac{G(\vec{x}) + S(\vec{Y}) + U(\tau)Y_j - C_j}{m} \\
\leq\ &\{\text{Since } G(\vec{x}) \leq G(\vec{z})\} \\
&\frac{G(\vec{z}) + S(\vec{Y}) + U(\tau)Y_j - C_j}{m} \\
=\ &\{\text{By (4.7)}\} \\
&x_j.
\end{aligned}$$

Therefore, $\vec{x}$ is compliant. □

Lemma 4.5 demonstrates that each inequality in (4.4) should actually be an equality, or the vector cannot be the minimum. A minimum compliant vector must therefore be of the form

$$x_i = \frac{G(\vec{x}) + S(\vec{Y}) + U(\tau)Y_i - C_i}{m} \quad \forall i. \tag{4.8}$$

Because $G(\vec{x})$ does not depend on i, there must exist a real number

$$s = \frac{G(\vec{x})}{m} \tag{4.9}$$

such that

$$x_i = s + \frac{S(\vec{Y}) + U(\tau)Y_i - C_i}{m} \;\; \forall i. \tag{4.10}$$

We define some functions:

$$\vec{v}(s) \triangleq \vec{x} \text{ such that (4.10) holds,} \tag{4.11}$$

$$G(s) \triangleq G(\vec{v}(s)), \tag{4.12}$$

$$M(s) \triangleq G(s) - ms. \tag{4.13}$$

By (4.8)–(4.11), any minimum compliant vector must be $\vec{v}(s)$ for some s. Furthermore, $G(s)$ must equal ms, by (4.9). Therefore, $M(s) = 0$ if and only if $\vec{v}(s)$ is a compliant vector in the form of (4.8), and thus the minimum compliant vector. We are now ready to prove this section's main result:

Theorem 4.2. *For any given task set τ, there exists a unique minimum compliant vector.*

Proof. We wish to demonstrate that exactly one real s exists such that $M(s) = 0$. We will use the Intermediate Value Theorem from calculus.

A necessary precondition for the Intermediate Value Theorem is that $M(s)$ is a continuous function. The following lemma leads to the desired result as a corollary.

Lemma 4.2.1. *$G(s)$ is continuous over the reals.*

Proof. Let $\varepsilon > 0$ and $\delta_c \triangleq \frac{\varepsilon}{U^+ - 1}$. Consider s_0 such that $|s - s_0| < \delta_c$. If $s = s_0$, then it is trivially the case that $|G(s) - G(s_0)| \leq \varepsilon$. Otherwise, without loss of generality, assume $s < s_0$ (otherwise we can swap them.) Then $\vec{v}(s)$ and $\vec{v}(s_0)$ are of the form of $\vec{x}$ and $\vec{z}$, respectively, in Lemma 4.4, with $k = n$ and $\delta = |s - s_0|$. Thus,

$$\begin{aligned}
G(\vec{v}(s)) \leq{} & \{\text{By Lemma 4.4}\} \\
& G(\vec{v}(s_0)) \\
\leq{} & \{\text{By Lemma 4.4}\} \\
& G(\vec{v}(s)) + \delta_c \cdot (U^+ - 1) \\
={} & \{\text{By the definition of } \delta_c\} \\
& G(\vec{v}(s)) + \varepsilon.
\end{aligned}$$

Therefore, $|G(s)-G(s_0)| \leq \varepsilon$, so $G(s)$ is continuous over the reals. □

Let $C_{\max}$ denote the largest C_i value in τ. We now show that $M(0) > 0$ and $M(C_{\max}) < 0$, completing the preconditions for the Intermediate Value Theorem.

Lemma 4.2.2. $M(0) > 0$

Proof. Let $1 \leq i \leq N$ be arbitrary. Then:

$$\begin{aligned}
&M(0)\\
=\ &\{\text{By (4.13) with } s=0\}\\
&G(0)\\
=\ &\{\text{By (4.12) and (4.2)}\}\\
&\sum_{U^{+}-1 \text{ largest}} g(\tau_i, v_i(0), p)\\
\geq\ &\{\text{Since, by (4.1), } g(\tau_i, v_i(0), p) \text{ cannot be negative}\}\\
&g(\tau_i, v_i(0), 0)\\
=\ &\{\text{By the definition of } g(\tau_i, v_i(0), 0) \text{ in (4.1)}\}\\
&\min\{C_i,\ \max\{0,\ v_i(0)+C_i\}\}\\
=\ &\{\text{By (4.10) and (4.11), with } s=0\}\\
&\min\left\{C_i,\ \max\left\{0,\ \frac{S(\vec{Y})+U(\tau)Y_i-C_i}{m}+C_i\right\}\right\}\\
=\ &\{\text{Simplifying}\}\\
&\min\left\{C_i,\ \max\left\{0,\ \frac{S(\vec{Y})+U(\tau)Y_i+(m-1)C_i}{m}\right\}\right\}\\
>\ &0.
\end{aligned}$$

□

Lemma 4.2.3. $M(C_{\max}) < 0$.

Proof. By (4.1), $g(\tau_i, v_i(C_{\max}), p) \leq C_i$ for any i and p. Therefore, for any i and p,

$$g(\tau_i, v_i(C_{\max}), p) \leq C_{\max}. \tag{4.14}$$

Therefore,

$$\begin{aligned}
&M(C_{\max}) \\
&= \{\text{By (4.13) with } s = C_{\max}\} \\
&\quad G(C_{\max}) - mC_{\max} \\
&= \{\text{By (4.12)}\} \\
&\quad G(\vec{v}(C_{\max})) - mC_{\max} \\
&= \{\text{By (4.2)}\} \\
&\quad \sum_{U^{+}-1 \text{ largest}} g(\tau_i, v_i(C_{\max}), p) - mC_{\max} \\
&\leq \{\text{By (4.14)}\} \\
&\quad (U^{+}-1)C_{\max} - mC_{\max} \\
&\leq \{\text{Since } U^{+} \leq m\} \\
&\quad -C_{\max} \\
&< 0.
\end{aligned}$$

□

Lemma 4.2.4. *There is an $s \in (0, C_{\max})$ such that $M(s) = 0$.*

Proof. By Lemma 4.2.1, Lemma 4.2.2, Lemma 4.2.3, and the Intermediate Value Theorem. □

We now verify that the s value of Lemma 4.2.4 is unique, using the following lemma.

Lemma 4.2.5. *$s_1 \neq s_2$ implies $M(s_1) \neq M(s_2)$*

Proof. Without loss of generality, assume $s_2 > s_1$ (otherwise, swap them). $\vec{v}(s_1)$ and $\vec{v}(s_2)$ are of the form of $\vec{x}$ and $\vec{z}$, respectively, with $k = n$ and $\delta = s_2 - s_1$, in Lemma 4.4. Therefore,

$$G(s_2) \leq G(s_1) + (s_2 - s_1)(U^{+} - 1). \tag{4.15}$$

Thus,

$$M(s_2) - M(s_1)$$

$$
\begin{aligned}
&= \{\text{By (4.13)}\} \\
&\quad G(s_2) - ms_2 - G(s_1) + ms_1 \\
&\leq \{\text{By (4.15)}\} \\
&\quad G(s_1)+(s_2-s_1)(U^+-1)-ms_2-G(s_1)+ms_1 \\
&= \{\text{Simplifying}\} \\
&\quad (s_2 - s_1)(U^+ - 1 - m) \\
&\leq \{\text{Since } U^+ \leq m\} \\
&\quad -1(s_2 - s_1) \\
&< 0.
\end{aligned}
$$

Therefore, $M(s_1) \neq M(s_2)$. □

Lemma 4.2.5 demonstrates that $s_1 \neq s_2$ and $M(s_1) = 0$ imply $M(s_2) \neq 0$, so the value of s characterized in Lemma 4.2.4 is unique. □

Lemma 4.2.4 also leads to Theorem 4.3, which provides a response-time bound that can be quickly calculated.

Theorem 4.3. *The response time of any job of any task τ_i cannot exceed $C_{\max} + \frac{S(\vec{Y})+U(\tau)Y_i-C_i}{m} + C_i$.*

Proof. Follows from Lemma 4.2.4, (4.10), and Theorem 4.1. □

4.4 Computation Algorithm

We now show how to compute the minimum compliant vector for a task system τ in time polynomial with respect to the size of τ and the number of processors. $G(s)$ as defined in (4.12) is a piecewise linear function; our algorithm works by tracing $G(s)$ until we find a fixed point $G(s) = ms$.

In order to assist the reader's understanding of this algorithm, we provide an example task system in Figure 4.2. Simple calculations reveal that, for this system, $S(\vec{Y}) = 0$ and $U(\tau) = 2$. Furthermore, in a two-CPU system, by Definition 4.1, we only need to consider $p = 0$. A graph of the relevant $g(\tau_i, v_i(s), 0)$ functions with respect to s is provided in Figure 4.3.

	C_i	T_i	Y_i
τ_1	6	10	10
τ_2	12	10	10
τ_3	4	20	20

Figure 4.2: Two-CPU task system example for Section 4.4. Deadlines are not included because they do not influence our response-time based computation.

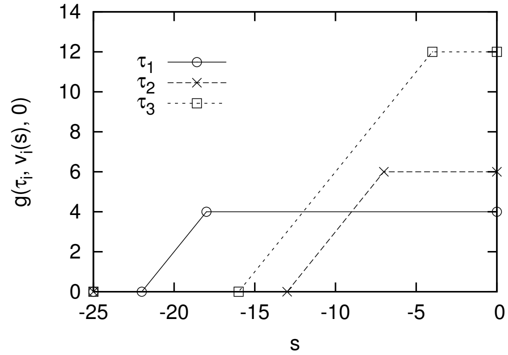


Figure 4.3: g functions for the system in Figure 4.2

We define the *slope* at point s of a piecewise linear function $f(s)$ to be $lim_{\varepsilon\to 0^+}\frac{f(s+\varepsilon)-f(s)}{\varepsilon}$. This definition differs from the common notion of derivative in that its limit is taken from the right; it is thus defined for all real s. For example, $g(\tau_1, v_1(s), 0)$ in Figure 4.3 has a slope of 1 at $s = -22$, but is not differentiable at $s = -22$.

For each value of s we will define $g(\tau_i, v_i(s), p)$ as being in one of three states, depending on the value of $v_i(s) + C_i - pT_i$:

- If $v_i(s) + C_i - pT_i < 0$, then $g(\tau_i, v_i(s), p)$ is in state 0, is equal to 0, and has a slope of 0. $g(\tau_1, v_1(s), 0)$ in Figure 4.3 is in state 0 in the interval $(-\infty, -22)$.

- If $0 \le v_i(s) + C_i - pT_i < C_i$, then $g(\tau_i, v_i(s), p)$ is in state 1, is equal to $v_i(s) + C_i - pT_i$, and has a slope of 1. $g(\tau_1, v_1(s), 0)$ in Figure 4.3 is in state 1 in the interval $[-22, -18)$.

- If $C_i \le v_i(s) + C_i - pT_i$, then $g(\tau_i, v_i(s), p)$ is in state 2, is equal to C_i, and has a slope of 0. $g(\tau_1, v_1(s), 0)$ in Figure 4.3 is in state 2 in the interval $[-18, \infty)$.

In order to analyze the piecewise linear function $G(s)$, we will need to determine where the slope changes. To do so, we need to determine which $g(\tau_i, v_i(s), p)$ components contribute to $G(s)$ for various intervals. For some intervals, the choice is arbitrary. For example, the task system in Figure 4.3 has only one $g(\tau_i, v_i(s), p)$ component contributing to $G(s)$, because $m-1 = 2-1 = 1$, but for $s < -22$, all $g(\tau_i, v_i(s), p)$ components equal zero. We provide a sufficient solution by arbitrarily tracking some valid set of $g(\tau_i, v_i(s), p)$ components.

We will create a set points of tuples, one for each possible change in the slope of $G(s)$. (Each will have an associated s value, but there could be multiple possible changes at the same s value.) Each tuple will identify a point where some $g(\tau_{i_0}, v_{i_0}(s), p_0)$ in state h_0 is replaced by some $g(\tau_{i_1}, v_{i_1}(s), p_1)$ in state h_1. Such a tuple will be of the form $(s, i_0, p_0, h_0, i_1, p_1, h_1)$. In some cases, more than one old component may be appropriate. To handle these cases efficiently, any of i_0, p_0, or h_0 may be set to $*$, which is defined as matching any value of the relevant parameter. For example, the tuple $(s, *, *, 0, i_1, p_1, 1)$ indicates that any arbitrary $g(\tau_{i_0}, v_{i_0}(s), p_0)$ in state 0 should be replaced by $g(\tau_{i_1}, v_{i_1}(s), p_1)$ in state 1.

The slope of $G(s)$ may change in any of the following cases:

1. Some $g(\tau_i, v_i(s), p)$ changes from state 0 to state 1. This occurs where $v_i(s) + C_i - pT_i = 0$. The resulting tuple will be $(s, *, *, 0, i, p, 1)$, as we can view $g(\tau_i, v_i(s), p_i)$ as replacing any $g(\tau_j, v_j(s), p_j)$ in state 0 in the system—they all have value 0. This change occurs exactly once per $g(\tau_i, v_i(s), p)$ and therefore $U^+ - 1$ times per task (once per value of p), for a total of $O(U^+ \cdot n)$ times for the system. In Figure 4.3, this state change occurs for $g(\tau_1, v_1(s), 0)$ at $s = -22$, for $g(\tau_2, v_2(s), 0)$ at $s = -13$, and for $g(\tau_3, v_3(s), 0)$ at $s = -16$.

2. Some $g(\tau_i, v_i(s), p)$ changes from state 1 to state 2. This occurs where $v_i(s) + C_i - pT_i = C_i$ (so $v_i(s) = pT_i$). The resulting tuple will be $(s, i, p, 1, i, p, 2)$. As above, this change occurs $O(U^+ \cdot n)$ times for the system. In Figure 4.3, this state change occurs for $g(\tau_1, v_1(s), 0)$ at $s = -18$, for $g(\tau_2, v_2(s), 0)$ at $s = -7$, and for $g(\tau_3, v_3(s), 0)$ at $s = -4$.

3. Some $g(\tau_i, v_i(s), p_i)$ is in state 1 and crosses C_j, and thus potentially crosses $g(\tau_j, v_j(s), p_j)$ (for some p_j) where the latter is in state 2. This occurs when $C_i > C_j$ and $v_i(s) + C_i - p_iT_i = C_j$. The resulting tuple will be $(s, j, *, 2, i, p, 1)$. This point may exist at most $n-1$ times per $g(\tau_i, v_i(s), p)$ (in the worst case, $g(\tau_i, v_i(s), p)$ crosses one $g(\tau_j, v_j(s), p_j)$ for each other τ_j), so occurs at most $O(U^+ \cdot n^2)$ times for the system. In Figure 4.3, this point does not occur for τ_1 (as C_1 is the smallest value in the system), occurs for $g(\tau_2, v_2(s), 0)$ with τ_1 at $s = -9$, and occurs for $g(\tau_3, v_3(s), 0)$ with τ_1 at $s = -12$ and with τ_2

```
slope := 0;
current := 0;
foreach $(s_1, i_1, p_1, h_1, i_2, p_2, h_2) \in$ points do
    if $(i_1, p_1, h_1)$ matches some $(i, p, h)$ in active then
        Replace $(i, p, h)$ in active with $(i_2, p_2, h_2)$;
        if $h_2 = 1$ then
            // Changing to state 1 means slope increases.
            slope := slope + 1;
        else
            // Must be changing away from state 1 or $(s_1, i_1, p_1, h_1, i_2, p_2, h_2)$ wouldn't
               be in points
            slope := slope − 1;
        $s_2$ := next $s$ value from points, or $C_{\max}$ if there is no such value;
        $s := \frac{\text{current} - \text{slope} \cdot s_1}{m - \text{slope}}$;
        if $s \in [s_1, s_2)$ then
            return $s$;
        current := current + slope $\cdot (s_2 - s_1)$;
```

Algorithm 4.1: Computation of minimum compliant vector.

at $s = -10$. (Although $g(\tau_3, v_3(s), 0)$ does not actually cross $g(\tau_2, v_2(s), 0)$ at $s = -10$, our algorithm nonetheless records the point where $g(\tau_3, v_3(s), 0)$ crosses C_2.)

In order to track $G(s)$, we order the tuples in points by s value, breaking ties in favor of tuples indicating a change in state for a particular $g(\tau_i, v_i(s), p)$ component. We create a list active containing tuples (i, p, h), each representing the corresponding $g(\tau_i, v_i(s), p)$ in state h that contributes its value to $G(s)$. For s smaller than the smallest in points, we may arbitrarily make $U^+ - 1$ choices of $g(\tau_i, v_i(s), p)$ components, each in state 0. Therefore, we initialize active to an arbitrary choice of $U^+ - 1$ tuples of the form $(i, p, 0)$.

The appropriate s value is computed using Algorithm 4.1, which works by tracing the piecewise linear function and checking for $G(s) = ms$ (as per (4.8)–(4.13)) in each segment.

As an example, suppose active is initialized to $\{(3, 0, 0)\}$, which represents $g(\tau_3, v_3(s), 0)$ in state 0. The first tuple in points is $(-22, *, 0, 0, 1, 0, 1)$, representing the leftmost slope change in Figure 4.3. This tuple will match the single tuple in active, so active will become $(1, 0, 1)$. slope is used to track the slope between s_1 and the next s value in points (which is called s_2). current is used to represent the correct value of $G(s_1)$. In this case, the immediate interval of interest is $-22 \le s < -18$. The new state h_2 is 1, so the slope (which was initially 0) will be incremented by 1, resulting in a new slope of 1. We now know the slope slope $= 1$ of $G(s)$ over $[-22, -18)$ and its value $G(s_1) =$ current $= 0$ at $s_1 = -22$. We therefore compute the point where

$G(s) = ms$ would hold, assuming a linear function that is equal to the correct piecewise linear function over the interval of interest. In this case, s is assigned the value $\frac{0-(-22)}{2-1} = 22$, which is not in $[-22, -18)$, so the desired value of s for the algorithm is not in the immediate interval of interest. We do not return, so we update the value $\mathsf{current}$ to match the value of $G(s_2)$ at the end of the immediate interval of interest (and thus in the next iteration the correct value of $G(s_1)$). In this case, $\mathsf{current}$ will be assigned to $0 + 1 \cdot 4 = 4$.

The set points is of size $O(U^+ \cdot n^2)$, and the set active is of size $O(U^+)$, so checking for matches will require $O(U^{+2}n^2)$ operations over the execution of the algorithm. Each match requires $O(1)$ time to process, so the complexity of Algorithm 4.1 is $O(U^{+2}n^2)$. Computing points requires $O(U^+ \cdot n^2)$ time, and sorting requires $O(U^+ \cdot n^2 \log(U^+ \cdot n))$ time, so the complexity of computing s is $O(U^+ \cdot n^2 \log(U^+ \cdot n) + U^{+2}n^2)$. Once an s value has been computed using Algorithm 4.1, the correct minimum compliant vector is simply $\vec{v}(s)$, which can be computed in $O(n)$ time.

4.5 Evaluation

As discussed in Section 4.3, we do not have an LP model for this type of system. Therefore, we do not have an efficient means of calculating the "best" choice of $\vec{Y}$. Therefore, to provide a comparison between systems with and without intra-task precedence constraints, we consider scheduling under G-EDF with implicit deadlines.

Under implicit deadlines, each $Y_i = T_i$. Therefore, for arbitrary τ_i,

$$\begin{aligned}
S_i(Y_i) &= \{\text{By the definition of } S_i(Y_i) \text{ in (3.8)}\} \\
&\quad C_i \cdot \max\left\{0,\ 1 - \frac{Y_i}{T_i}\right\} \\
&= \{\text{Because } Y_i = T_i\} \\
&\quad 0.
\end{aligned}$$

Thus, by the definition of $S(\vec{Y})$ in (4.3), $S(\vec{Y}) = 0$. Combining this result with Theorem 4.3 and the necessary condition that $U(\tau) \leq m$, the response time of any job of any task τ_i must be upper-bounded by $C_{\max} + Y_i + \frac{m-1}{m}C_i$. Therefore, the *lateness* of any job of τ_i must be no greater than $C_{\max} + \frac{m-1}{m}C_i$.

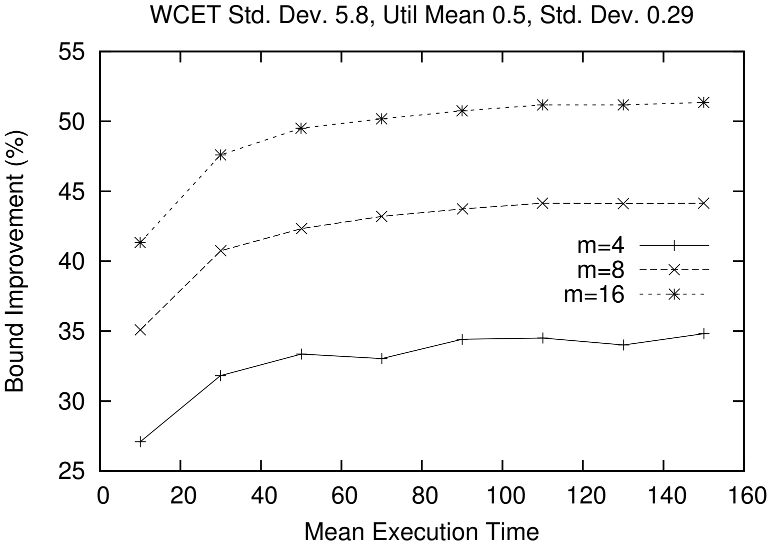
In order to evaluate the improvement to the bounds we obtain by eliminating implicit precedence constraints, we compared our results to those available using the EDF-CVA2 technique in Chapter 3. As shown in Chapter 3, this technique typically provides the best bounds for highly utilized systems scheduled using implicit-deadline G-EDF. (As discussed in Chapter 3, EDF-CVA can also analyze the same systems, but typically provides inferior bounds to EDF-CVA2 for highly utilized systems.)

Our experiments were intended to show how varying different task system parameters affected the improvements available by removing intra-task precedence constraints. All experiments were done with processor counts of 4, 8, and 16. We used uniform distributions for the task worst-case execution times and utilizations, described below. We determined the effects of varying each of four parameters: mean worst-case execution time ($\bar{C}$), standard deviation of worst-case execution time (C_σ), mean utilization ($\bar{U}$), and standard deviation of utilization (U_σ). We performed one experiment for each parameter. For mean x and standard deviation σ, values were chosen uniformly over $(x-\sigma\sqrt{3}, x+\sigma\sqrt{3})$.

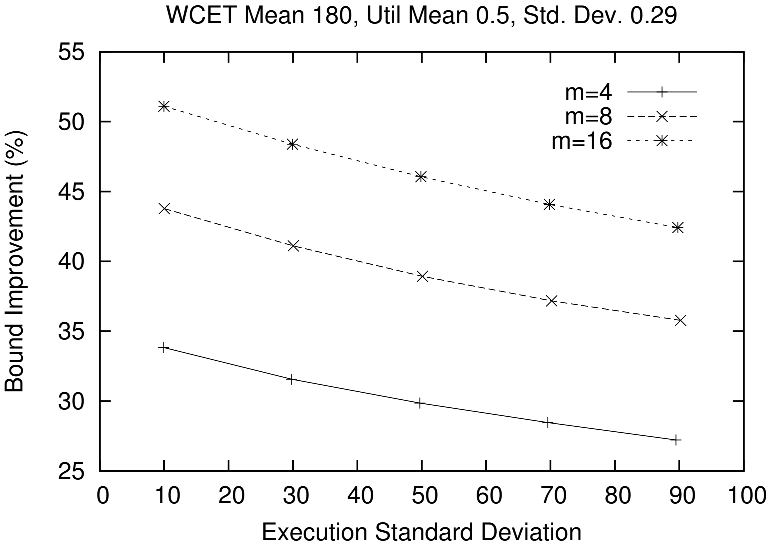
In each experiment, the processor count m and three of the four parameters above were fixed, and the remaining parameter was varied. For each value of the varied parameter, we generated 1,000 task sets. For each individual task set, we generated tasks until a task was generated that would cause $U(\tau)$ to exceed m, and that task was then discarded. For each task set we computed the mean lateness bound under EDF-CVA2, δ, and using Theorem 4.1, δ'. For each set of 1,000 task sets we computed $\bar{\delta}$ (the mean value of δ) and $\bar{\delta}'$ (the mean value of δ'). The *absolute improvement* for each set of sets is defined as $\bar{\delta}-\bar{\delta}'$, and the *relative improvement* for each set of sets is defined as $(\bar{\delta}-\bar{\delta}')/\bar{\delta}$.

Results of the two experiments varying WCET parameters are presented in Figure 4.4. In Figure 4.4(a), we use $C_\sigma = 5.8$, $\bar{U} = 0.5$, and $U_\sigma = 0.29$, and we depict the relative improvement with respect to $\bar{C}$. For larger $\bar{C}$, the relative improvement increases, but the dependence on $\bar{C}$ is small. This means that EDF-CVA2 is slightly more sensitive to $\bar{C}$ than the techniques proposed in this chapter. In Figure 4.4(b), we use $\bar{C} = 180$, $\bar{U} = 0.5$, and $U_\sigma = 0.29$, and we depict the relative improvement with respect to C_σ. Here we see a decreasing trend. When $\bar{C}$ is held constant but C_σ is increased, the largest few WCET values tend to increase. We see that the techniques proposed in this chapter are slightly more sensitive to this effect than EDF-CVA2.

Results of the two experiments varying utilization parameters are presented in Figure 4.5. In Figure 4.5(a), we use $\bar{C} = 10$, $C_\sigma = 2.9$, and $U_\sigma = 0.029$, and we depict the relative improvement with respect to $\bar{U}$. We see here that the relative improvement increases dramatically as mean utilizations are increased, indicating that EDF-CVA2 is significantly more sensitive to $\bar{U}$. This effect is not surprising, because no U_i term appears in Definition 4.1. In Figure 4.5(b), we use $\bar{C} = 10$, $C_\sigma = 2.9$, and $\bar{U} = 0.5$, and we depict the relative improvement with respect to U_σ. In an analogous fashion to the WCET experiments, increasing U_σ while



(a) Varying WCET mean.



(b) Varying WCET standard distribution.

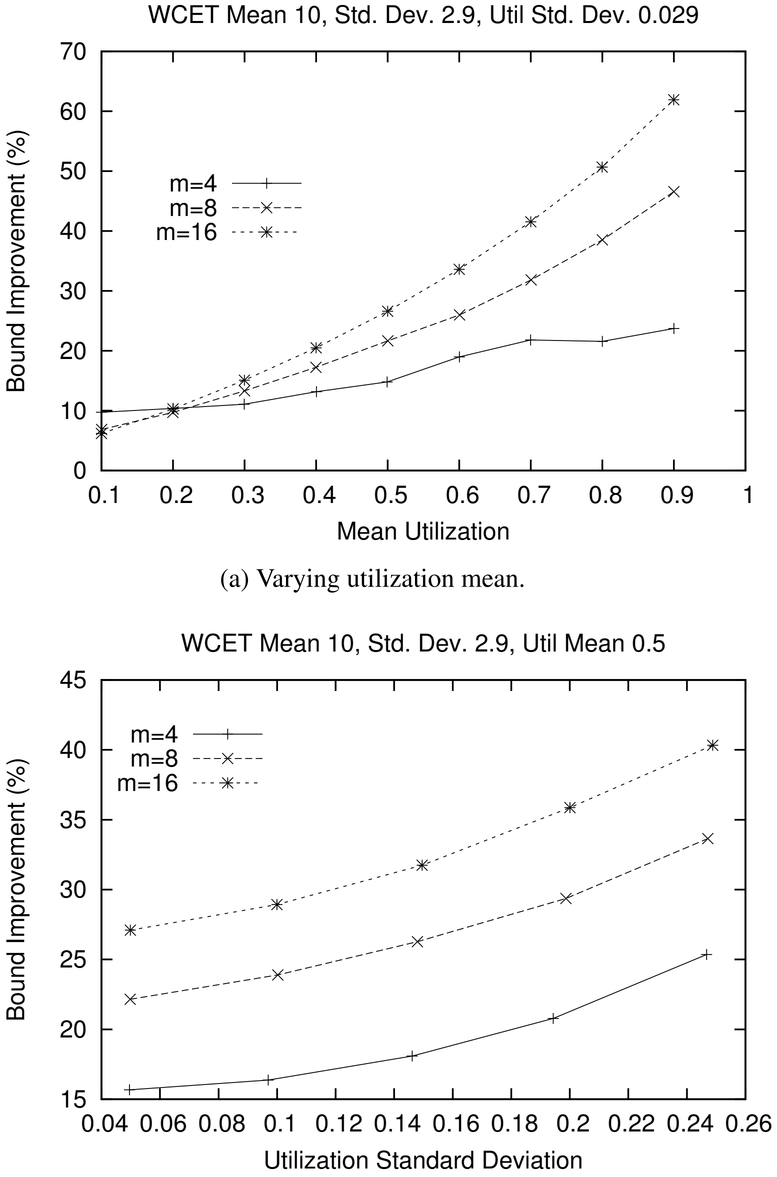
Figure 4.4: Results of experiments varying WCET parameters.

holding $\bar{U}$ constant resulted in increasing the largest few utilizations. In this case, EDF-CVA2 is sensitive to this effect, but the techniques proposed in this chapter do not depend on utilizations. Thus, the relative improvement grows with increasing U_{σ}.

Overall, we see that the relative improvement is quite substantial, particularly with large execution times, small variance in execution times, large utilizations, and large variance in utilizations. More significant improvement occurs with larger processor counts because the EDF-CVA2 bounds increase significantly with m, while our bounds are upper-bounded by $C_{\max} + \frac{m-1}{m} C_i$. This improvement is possible even when per-task utilization is restricted to be less than one to make our results comparable to prior work. We do not have results comparing our work to previous results when per-task utilization may exceed one, because prior work is not applicable in this case.

4.6 Conclusion

GEL scheduling has already proven useful for traditional SRT workloads in which jobs of the same task have implicit precedence constraints. Here we have demonstrated that GEL scheduling may be even more useful for SRT workloads in which jobs may be released as separate threads that can safely run concurrently. We have shown that doing so not only improves response times compared to prior work, but enables new workloads where a single task may overutilize a single processor.



(a) Varying utilization mean.

(b) Varying utilization standard deviation.

Figure 4.5: Results of experiments varying utilization parameters.

CHAPTER 5: JOB SPLITTING[1]

In this chapter, we discuss the technique of *job splitting* to reduce lateness bounds. Recall that in Chapter 3, lateness bounds are a function of job WCETs. If each job is split into a number of equally sized subjobs, then the WCET of each subjob is smaller that that of the original job, and the lateness bounds become smaller. However, job splitting increases the likelihood that the original job will be preempted/migrated frequently and thus can increase overheads that negatively impact schedulability. Also, as explained later, job splitting can cause problems for locking protocols. In this chapter, we examine the practical viability of job splitting for reducing tardiness bounds under GEL schedulers in light of such complications. We initially consider G-EDF, because it is the simplest GEL scheduler. However, all techniques are also applicable to other GEL schedulers such as G-FL. For experimental results, we consider G-FL due to its properties discussed in Chapter 3.

The remainder of this chapter is organized as follows. In Section 5.1, we illustrate our splitting technique with a motivating example. In Section 5.2, we describe the particulars of our task model and notation. Then, in Section 5.3, we describe the split G-EDF scheduling algorithm. We then discuss, in Section 5.4, the necessary modifications to the overhead accounting from Section 2.2 in order to account for splitting. In Section 5.5, we describe how to handle critical sections, and in Section 5.6, we discuss how the analysis from Chapter 3 can be modified to efficiently account for the changes made due to critical sections. We discuss overhead analysis in the presence of critical sections in Section 5.7. Then, in Section 5.8, we provide experimental results using G-FL, showing that our splitting scheme is effective. In Section 5.9, we conclude.

5.1 Basic Technique

In this section, we introduce our job splitting technique with a motivating example.

[1] This work appeared in preliminary form in the following paper:

Erickson, J. and Anderson, J. (2013). Reducing tardiness under global scheduling by splitting jobs. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems*, pages 14–24.

Motivating Example. For motivational purposes, we will repeatedly consider example schedules of a task system τ with three tasks, which we specify here using the notation (C_i, T_i): $\tau_1 = (4\text{ ms}, 6\text{ ms})$, $\tau_2 = (9\text{ ms}, 12\text{ ms})$, and $\tau_3 = (14\text{ ms}, 24\text{ ms})$. Each job of τ_1, τ_2, and τ_3 has a deadline at the earliest possible release time of its successor. An example G-EDF schedule for τ is given in Figure 5.1(b). Observe that τ_3 misses a deadline at time 24. (This system meets the constraints for bounded tardiness in Chapter 3.)

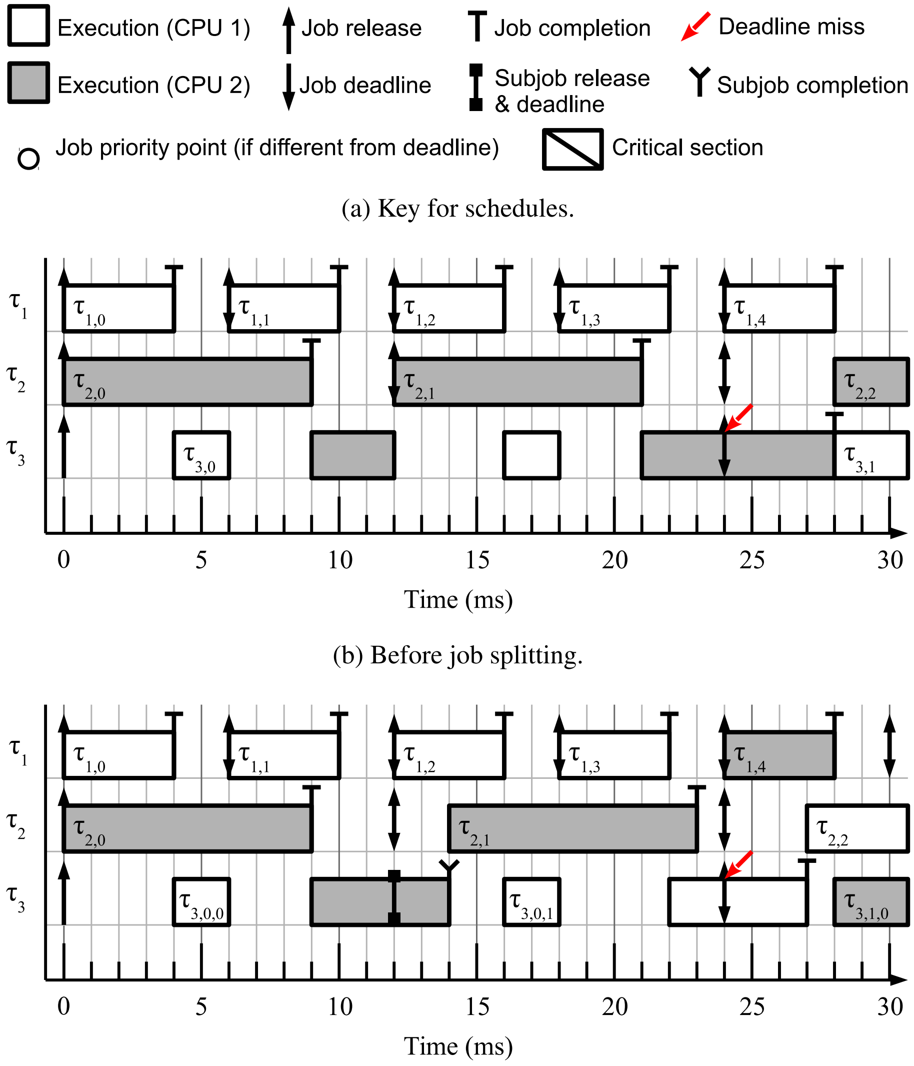
A Continuum of Schedulers. In the implementation of job splitting we propose, all job splitting is done through budget tracking in the OS. That is, job splitting does not require actually breaking the executable code that defines a task into multiple pieces. We define the *split factor* of a task as the number of subjobs into which each of its jobs is split. With any GEL scheduler, existing tardiness bounds can be driven arbitrarily close to zero by arbitrarily increasing such split factors. In the "limit," i.e., when each subjob becomes one time unit (or quantum) in length, a GEL algorithm becomes similar in nature to algorithms within the Pfair family of schedulers (Baruah et al., 1996), which are optimal for implicit-deadline sporadic systems. One can thus view task split factors as tuning parameters that can be set to select a desired scheduler within a continuum of schedulers to achieve desired tardiness bounds. If theoretical tardiness were the only issue, then split factors would naturally be set arbitrarily high, but this raises practical concerns, as discussed earlier.

Returning to our example task system, Figure 5.1(c) depicts a schedule for τ under G-EDF in which each job of τ_3 is split into two subjobs. Note that splitting is done in a way that preserves the task's original utilization. In this example, the tardiness of τ_3 is reduced by 1 ms, and no additional preemptions happen to be necessary.

5.2 Task Model

As in Chapter 3, we assume $U_i \leq 1$ for any i, $\sum_{\tau_i \in \tau} U_i \leq m$, and $n > m$. If $n \leq m$, then each task can be assigned its own processor, and each job of each τ_i will complete within C_i time units of its release. We assume that the OS enforces execution budgets, so that each job runs for at most C_i time units. In this chapter, we also assume implicit deadlines: for each τ_i, $D_i = T_i$.

To distinguish between a task before splitting (e.g., τ_3 in Figure 5.1(b)) and the same task after splitting (τ_3 in Figure 5.1(c)), we define τ_i^{base} as the *base task* before splitting and τ_i^{split} as the *split task* after splitting. To disambiguate between base and split tasks, we also use superscripts on parameters: C^{base}, C^{split}, U^{base}, etc. A job of a base task is called a *base job*, while a split task is instead composed of *subjobs* of base jobs. We



(a) Key for schedules.

(b) Before job splitting.

(c) Each job of τ_3 is split into two subjobs and jobs of the other tasks are not split.

Figure 5.1: G-EDF schedules showing the scheduling of $\tau_1 = (4, 6)$, $\tau_2 = (9, 12)$, and $\tau_3 = (14, 24)$ on two processors.

define the *split factor* of τ_i^{base}, denoted s_i, to be the number of subjobs per base job. In Figure 5.1(c), $s_3 = 2$. The subjobs of a base job $\tau_{i,k}^{base}$ are denoted $\tau_{i,k,0}, \tau_{i,k,1}, \ldots, \tau_{i,k,s_i-1}$. $\tau_{i,k,0}$ is its *initial subjob* (e.g., the first subjob $\tau_{3,0,0}$ of $\tau_{3,0}^{base}$ in Figure 5.1(c)) and τ_{i,k,s_i-1} is its *final subjob* (e.g., the second subjob $\tau_{3,0,1}$ of $\tau_{3,0}^{base}$ in Figure 5.1(c)). The longest time that any job of τ_i^{base} waits for or holds a single outermost lock is denoted C_i^{ext} (so named because it will be used to *ext*end execution budgets). Split tasks use a variant of the sporadic task model that is described in Sections 5.5 and 5.6, but the usual sporadic task model is assumed prior to Section 5.5.

The *max-lateness* bound for τ is the largest lateness bound for any $\tau_i \in \tau$. Similarly, the *max-tardiness* bound for τ is the largest tardiness bound for any $\tau_i \in \tau$.

When a non-final subjob completes, the resulting change in deadline is a *deadline move* (*DLM*). In Figure 5.1(c), a DLM occurs at time 14 for τ_3.

5.3 Split G-EDF Scheduling Algorithm

In this section, we describe the OS mechanisms necessary to implement job splitting under G-EDF. Although we will require the system designer to specify the split factor s_i for each job, we do not require the jobs to be split *a priori*. Instead, the OS will use the budget tracking schemes described in this section to perform DLMs at the appropriate times.

When certain events occur, the scheduler within the OS is called. We refer to this call as a *reschedule*. For example, a reschedule occurs whenever a job completes, so that another job can be selected for execution. In our implementation of splitting in LITMUS$^{\text{RT}}$, part of the scheduling process involves checking whether the currently executing job needs a DLM, and to perform the DLM if so.

In this section, we let $C_i^{split} = C_i^{base}/s_i$ and $T_i^{split} = T_i^{base}/s_i$. For example, in Figure 5.1(c), $C_3^{split} = 14/2 = 7$ and $T_3^{split} = 24/2 = 12$.

Like the response-time analysis in Chapter 4, the lateness analysis in Chapter 3 continues to hold if jobs become available for execution before their release times, as long as their deadlines are based on release times that follow the minimum separation constraint of the sporadic task model. The technique of allowing jobs to run before their release times is called *early releasing* (Anderson and Srinivasan, 2000). Allowing subjobs to be released early may prevent tasks from suspending unnecessarily and allows us to alter the deadline of a job $\tau_{i,k}$ only when it has executed for a multiple of C_i^{split}. With early releasing, we do not have to consider the wall clock time when determining a split job's deadline, because we can instead consider only its cumulative execution time. Additionally, as discussed in Section 5.4, early releasing prevents the same job from having to incur certain overheads multiple times.

We will track the budget of each $\tau_{i,k}$ in order to simulate the execution of $\tau_{i,k,0}, \tau_{i,k,1}, \ldots \tau_{i,k,s_i}$ with T_i^{split} time units between each pair of subjob releases and with each subjob executing for up to C_i^{split} time units. In order to do so, we define several functions below with respect to time. These functions are only defined for time t such that τ_i^{base} has a job that is ready for execution (it is released and its predecessor job has completed) but has not completed. We let $J_i(t)$ denote this job. For example, in Figure 5.1(c), $J_3(t)$ denotes $\tau_{3,0}^{base}$ for any $t \in [0,27)$, and $\tau_{3,1}^{base}$ after $t = 27$. Several of these functions are explicitly labelled as "ideal" functions that ignore critical sections—deviation from "ideal" behavior due to critical sections will be described in Section 5.5.

- The *current execution* $e_i(t)$ is the amount of execution that $J_i(t)$ has already completed before time t. In Figure 5.1(c), $e_3(4) = 0$ and $e_3(5) = 1$. Our definition of $e_i(t)$ allows us to keep track of how many subjobs of $J_i(t)$ have already completed.

- The *current release* $r_i(t)$ is the release time of $J_i(t)$. Note that $r_i(t)$ is the release time of the current base job, not the current subjob. In Figure 5.1(c), $r_3(4) = r_3(17) = 0$ and $r_3(29) = 24$.

- The *current offset* $o_i(t)$ is a parameter that will be used to adjust execution budgets in order to to properly handle overheads. In this section, we assume that $o_i(t) = 0$ for all t.

- The *naïve ideal subjob* $\lambda_i(t)$ is the index of the subjob of $J_i(t)$ that should be executing at time t, ignoring the effect of critical sections and acting as though $J_i(t)$ had executed for $e_i(t) + o_i(t)$ time units. In other words, it is the index of the subjob that should be executing based on the number of multiples of C_i^{split} that $J_i(t)$ has completed by time t, after the completed execution time has been adjusted by $o_i(t)$. It is defined as follows:

$$\lambda_i(t) = \left\lfloor \frac{s_i \cdot (e_i(t) + o_i(t))}{C_i^{base}} \right\rfloor. \tag{5.1}$$

Because $o_i(t) = 0$ for all t, in Figure 5.1(c), $\lambda_3(4) = 0$, $\lambda_3(17) = 1$, and $\lambda_3(29) = 0$. (Recall that subjobs are zero-indexed.) This quantity is "naïve" because if $o_i(t)$ is decreased, then $\lambda_i(t)$ may decrease, indicating (incorrectly) that a previously completed subjob should be running. Our algorithm should never actually simulate the execution of a completed subjob.

- The *ideal subjob* $j_i(t)$ is the index of the subjob of $J_i(t)$ that should be executing at time t, ignoring the effect of critical sections. It is simply the largest value of $\lambda_i(t_0)$ that has occurred at some time $t_0 \leq t$ such that $J_i(t_0) = J_i(t)$. In this section, because $o_i(t) = 0$ for all t, by (5.1), $\lambda_i(t)$ can decrease only at base job boundaries. Therefore, $j_i(t) = \lambda_i(t)$ for all i and all t. (Later, when we allow $o_i(t) \neq 0$, the two may differ.)

- The *ideal next DLM* $v_i(t)$ is the time for the next DLM after time t, ignoring the effect of critical sections and assuming that $J_i(t)$ is scheduled continuously from time t until $v_i(t)$. In other words, $v_i(t)$ is the time when the current ideal subjob should end assuming that it is not preempted. It is defined as follows:

$$v_i(t) = t + \frac{(j_i(t)+1)C_i^{base}}{s_i} - (e_i(t) + o_i(t)). \tag{5.2}$$

  In Figure 5.1(c), $v_3(4) = 11$. Observe that, because $\tau_{3,0,0}$ is actually preempted at time 6, the DLM actually does not occur until time 14.

- The *ideal subjob release* $\rho_i(t)$ is the release time for the current ideal subjob. It is defined as follows:

$$\rho_i(t) = r_i(t) + T_i^{split} j_i(t). \tag{5.3}$$

  (5.3) reflects that the subjobs are released every T_i^{split} time units, and the first subjob is released at the same time as the corresponding base job. In Figure 5.1(c), $\rho_3(4) = 0$. Although it does not occur in Figure 5.1(c), it is possible (due to early releasing) for the ideal subjob release to be after that subjob actually commences execution.

- The *ideal deadline* $d_i(t)$ is the deadline that should be active for $J_i(t)$ at time t, ignoring the effect of critical sections. In other words, it is the deadline of the ideal subjob $j_i(t)$. It is defined as follows:

$$d_i(t) = \rho_i(t) + T_i^{split}. \tag{5.4}$$

  (5.4) follows from the definition of G-EDF scheduling. In Figure 5.1(c), $d_3(4) = 12$.

- The *current deadline* $\delta_i(t)$ is the deadline that the scheduler actually uses for $J_i(t)$ at time t. This value is maintained by the budget tracking algorithm we describe in this section, rather than being merely a definition like the functions above. Because there are no critical sections in Figure 5.1(c) (as we are assuming in this section), $\delta_i(t)$ should be $d_i(t)$ for all i and all t. Therefore, $\delta_3(4)$ should be 12.

With these definitions in place, we define budget tracking rules in order to maintain the invariant $\delta_i(t) = d_i(t)$.

- **R1.** If a job of τ_i^{base} is released at time t, then $\delta_i(t)$ is assigned to $d_i(t)$.

  In Figure 5.1(c), applying this rule at time 0, we have $\delta_3(0) = 12$.

- **R2.** Whenever a non-final subjob of τ_i^{base} is scheduled at time t to run on a CPU, a *DLM timer* is assigned to force a reschedule on that CPU at time $v_i(t)$. Whenever τ_i^{base} is preempted, the DLM timer is cancelled.

  In the schedule depicted in Figure 5.1(c), the DLM timer for τ_3 is set at time 4 to fire at time $v_3(4) = 11$. However, the DLM timer is cancelled at time 6 when τ_3 is preempted. When τ_3 is selected for execution again at time 9, the DLM is set to fire at time 14. It does fire at that time and forces a reschedule. Because only the final subjob remains, the timer is not set at time 16.

- **R3.** If the scheduler is called at time t on a CPU that was running τ_i^{base}, then $\delta_i(t)$ is assigned the value $d_i(t)$, potentially causing a DLM.

  In Figure 5.1(c), the scheduler is called several times on a CPU that was running τ_3^{base}, including at times 6 and 14. At time 6, $d_3(t) = \delta_3(t)$ already held, so a DLM does not occur. However, a DLM occurs at time 14 because $d_3(t) > \delta_3(t)$ is established, causing $\delta_3(t)$ to be updated.

5.4 Overhead Analysis

We now describe how to implement job splitting in an efficient manner and how the overheads from our implementation differ from those in Section 2.2. Critical sections are not considered until Section 5.5. An illustration of overheads due to job splitting is given in Figure 5.2.

Care must be taken when performing budget enforcement (in order to end each subjob at the appropriate time) while accounting for overheads. In order to do so, we must distinguish between *actual budgets* used by

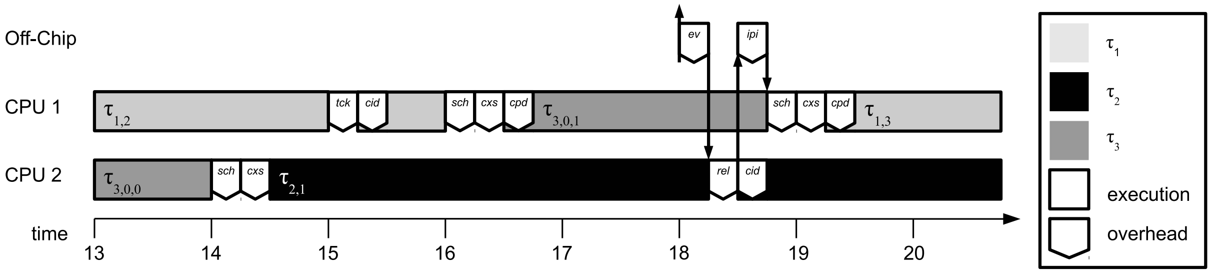


Figure 5.2: A subset of the schedule from Figure 5.1(c) with some overheads included.

the kernel and *analytical budgets* used in schedulability tests, because some overheads are charged to different tasks than those their execution affects. For example, the *cpd* overhead in Figure 5.2 before time 17 is incurred by $\tau_{3,0,1}$, but is charged to $\tau_{1,2}$. How to properly enforce budgets while accounting for overheads has not been previously described, to our knowledge. Henceforth, when we refer to "budget" without qualification, we are referring to the actual budget.

Our goal with budget enforcement is to simulate the execution of shorter subjobs in place of longer base jobs. As can be seen in Figure 5.2, most of the overheads considered above—*ev*, *rel*, *ipi*, *sch*, *cxs*, and *tck*—are not part of a job itself, so they should contribute only to the analytical job budget. However, the cache-related delays *cpd* and *cid* do affect the runtime of jobs. For the purpose of the analytical budget, *cpd* is charged to the job (like $\tau_{1,2}$) that relinquishes a processor to the preempted job, while *cid* is charged to the interrupt handler. However, in a real system these overheads would actually be incurred by the job that is preempted. Therefore, instead of assigning these overheads to the actual job budget ahead of time (as with the analytical budget), we wait until runtime to add each overhead to the budget for the job of τ_i^{base} *resuming after a preemption* at time t.

To aid in discussing the details of overhead analysis, we present in Section 5.4.1 below an example that illustrates most of the overheads that result from splitting and how we account for such overheads. Then, in Section 5.4.2, we discuss how to account for interrupt-related overheads. Finally, in Section 5.4.3, we discuss the specific mechanisms we use to manage subjob budgets.

5.4.1 Splitting Overhead Example

In Figure 5.3, we depict an example schedule of a system of three tasks on two processors, including relevant overheads. For simplicity, we do not consider interrupt-related overheads until Section 5.4.2 below, so we assume that a release master is used. Furthermore, for the purpose of illustration, we assume that each relevant overhead is exactly 1 ms.

We first consider Figure 5.3(b), which displays an actual schedule with overheads present. Each overhead is displayed with the task most directly affected. For example, each scheduling overhead is depicted with the task actually scheduled. We also assume that each time the scheduler selects a subjob (except for the very first time it selects each initial subjob), that subjob's budget is increased by Δ^{cpd}. For τ_1 at times 20 and 27, such a credit is not actually necessary, because τ_1 is not actually preempted. However, we allow this credit both to account for any loss of cache affinity that results from the execution of the scheduler, and to simplify the algorithm used to increase subjob budgets (as discussed in Section 5.4.3).

We first discuss some properties that are used in our overhead accounting. The scheduling algorithm with job splitting differs from normal GEL scheduling primarily due to the DLM timer, so we first consider the two different conditions that can apply when the DLM timer fires. In the first case, a subjob from a *different* task is selected.

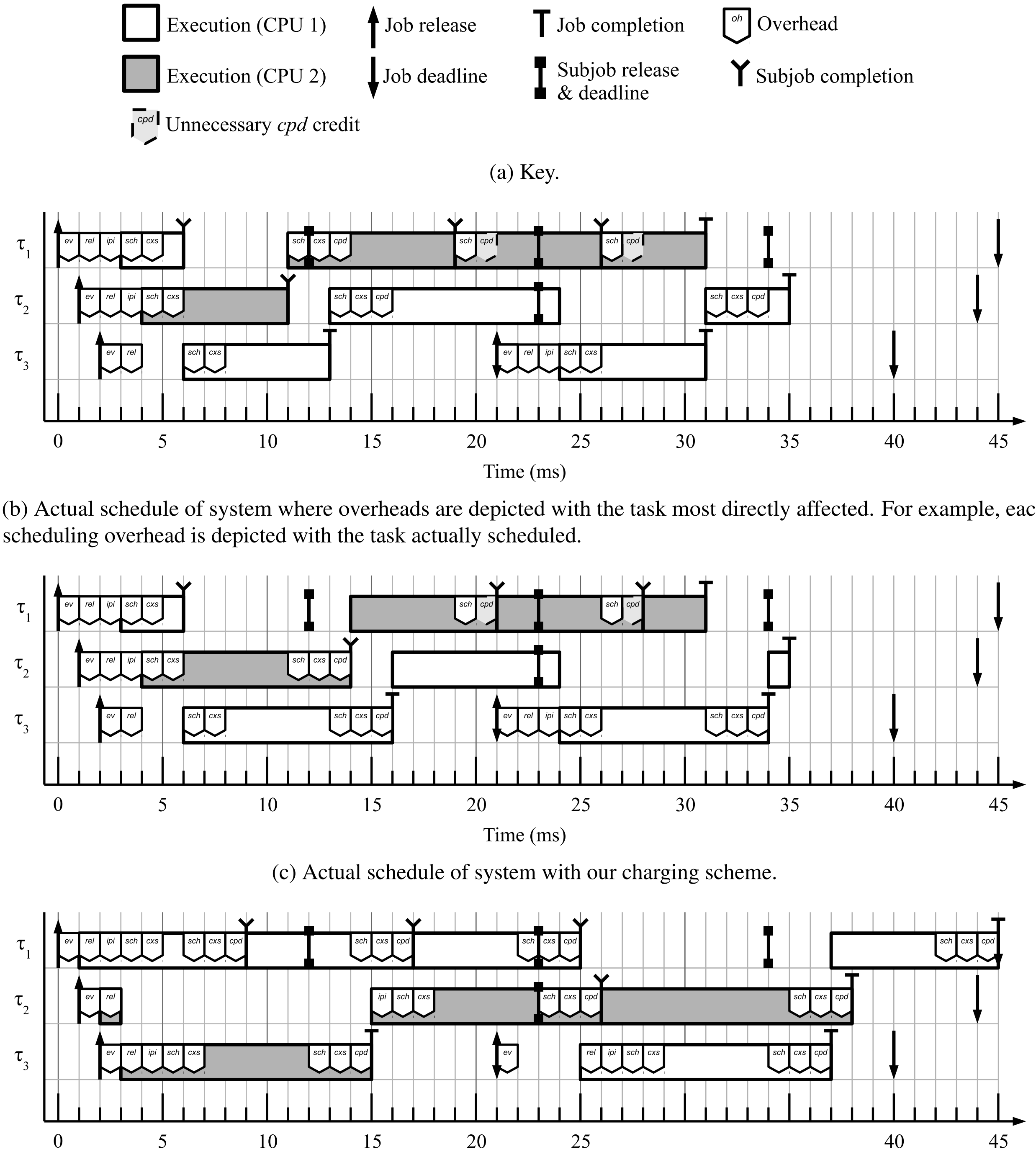
Property 5.1. *If a non-final subjob $\tau_{i,k,v}$ completes and $\tau_{j,\ell,w}$ ($j \neq i$) is selected for execution as a result, then from the perspective of the overheads incurred by τ_i, this situation can be accounted for like a preemption upon receipt of an IPI.*

To see why Property 5.1 holds, compare in Figure 5.3(b) the completion of the first subjob of τ_2 at time 11 and the preemption of τ_2 by τ_3 at time 24. In both cases, τ_2 is effectively preempted when the scheduler runs and selects a different task. At time 11, the scheduler runs on CPU 2 due to the DLM timer and selects τ_1 for execution, whereas at time 24, the scheduler runs on CPU 1 and selects τ_3 due to the IPI sent on behalf of τ_3. Furthermore, when the second subjob of τ_1 commences execution at time 13, it incurs the same overheads as when the preempted second subjob resumes execution at time 31.

We next consider the case that the next subjob from the same task is selected.

Property 5.2. *If non-final $\tau_{i,k,v}$ completes and $\tau_{i,k,v+1}$ is selected for execution, then the overhead is no greater than if $\tau_{i,k}$ were preempted and immediately resumed.*

To see why this property holds, consider in Figure 5.3(b) τ_1 at both time 19 and time 26. In each case, the scheduler is run due to the DLM timer, but τ_1 is again selected. The only overhead actually incurred is the execution of the scheduler itself, although (as discussed above) the scheduler credits τ_1 with an extra Δ^{cpd}



(a) Key.

(b) Actual schedule of system where overheads are depicted with the task most directly affected. For example, each scheduling overhead is depicted with the task actually scheduled.

(c) Actual schedule of system with our charging scheme.

(d) Schedule of system with all charged overheads.

Figure 5.3: Example task system with overheads, running on two processors plus a release master. In (b) and (c), an overhead is shown on top of an execution box if and only if it runs on that CPU. *rel* runs on the release master and *ev* and *ipi* do not have an associated CPU. τ_1 has a period of 45 ms and an actual budget of 1 ms for its initial subjobs and 5 ms for all other subjobs. τ_2 has a period of 43 ms and an actual budget of 5 ms for its initial subjobs and 9 ms for the rest of its subjobs. τ_3 has a period of 19 ms and an actual budget of 5 ms.

units of execution. By comparison, after τ_1 is actually preempted at time 6, when it resumes at time 11, it must incur *sch*, *cxs*, and *cpd* overheads.

Brandenburg (2011) accounts for the overheads incurred by a job $\tau_{i,k}$ resuming from a preemption (*sch*, *cxs*, and *cpd*, as seen in Figure 5.3(b) for τ_1 at time 11 and for τ_2 at times 13 and 31) by charging those overheads to the job whose completion allowed $\tau_{i,k}$ to run. We charge those overheads to the *subjob* whose completion allowed $\tau_{i,k}$ to run. In Figure 5.3(c), we depict the same schedule as in Figure 5.3(b), but with overheads charged using this scheme. The next two properties establish that this charging scheme is valid.

Property 5.3. *If $\tau_{i,k,v}$ completes and $\tau_{j,\ell,w}$ ($j \neq i$) is selected for execution as a result, then from the perspective of the overheads incurred by τ_j, treating this situation as a normal job completion suffices.*

To see that Property 5.3 holds, compare in Figure 5.3(b) the resumption of τ_1 at time 11, which occurs due to the completion of the first subjob of τ_2, and the resumption of τ_2 at time 13, which occurs due to the completion of the first job of τ_3. In each case, the same overheads (*sch*, *cxs*, *cpd*) are incurred by the subjob that resumes.

Property 5.4. *If $\tau_{i,k}$ is preempted, then $\tau_{i,k}$ resumes upon completion of some $\tau_{j,\ell,w}$ ($j \neq i$).*

Property 5.4 holds because our algorithm is *subjob*-level static priority; the priority of a subjob does not change during its execution. It can be seen in Figure 5.3(b) every time a job resumes, i.e., for τ_1 at time 11 and for τ_2 at times 13 and 31.

We now provide properties that specify how we charge different overheads to different subjob *analytical* budgets, as depicted in Figure 5.3(c) (which displays the actual schedule with each overhead charged to the correct job) and Figure 5.3(d) (which depicts a pessimistic schedule where each overhead that we charge to a subjob is incurred by that subjob). We also discuss how our accounting compares to that of Brandenburg (2011).

Our first overhead charging property follows from Properties 5.3 and 5.4 and the discussion above.

Property 5.5. *In order to account for the cost incurred by resuming previously preempted jobs, each subjob's analytical budget should be charged sch, cxs, and cpd overheads.*

These overheads are charged to *all* subjobs, both initial and non-initial. As can be seen in Figure 5.3(d), when not accounting for interrupts, these are the *only* overheads charged to non-initial subjobs. We next describe the overheads charged to initial subjobs.

Property 5.6. *In order to account for delays when a job is first released, its task's minimum separation time must be reduced by subtracting Δ^{ev} from it. This is done* before *splitting the job.*

This transformation is identical to that performed in (Brandenburg, 2011). *ev* is accounted for by subtracting from the period because, in a real system, Δ^{ev} is merely an upper bound on event latency. If one job experiences Δ^{ev} units of event latency, but its successor experiences a much shorter event latency, then the separation time of two jobs can be up to Δ^{ev} units shorter than expected. Because this accounting is for the delay due to *initially* starting a job, it does not apply to all subjobs. For example, in Figure 5.3(c), τ_1 incurs an *ev* overhead only at time 1.

In the absence of a release master, the interrupt-related overhead analysis in Section 5.4.2 accounts for the *rel* charge, so it does not need to be explicitly accounted for with each job. Therefore, in the absence of a release master, we define

$$c^{rm} \triangleq 0, \tag{5.5}$$

and in the presence of a release master, we define

$$c^{rm} \triangleq \Delta^{rel}. \tag{5.6}$$

We use these definitions when accounting for release overheads.

Property 5.7. *c^{rm} and ipi overheads should be charged to the analytical budgets of initial subjobs, but not those of non-initial subjobs.*

These overheads are only incurred once per base job, as in (Brandenburg, 2011). For example, in Figure 5.3(c), τ_1 incurs these overheads only between time 1 and 3. Furthermore, they are incurred before the job begins execution, so are charged to the initial subjob.

Property 5.8. *Each initial subjob's analytical budget should be charged sch and cxs overheads for its own scheduling, in addition to the charges referenced in Property 5.5.*

For example, these charges can be seen in Figure 5.3(c) for τ_1 at times 3–5.

Properties 5.5–5.8 are sufficient to account for all non-interrupt-related overheads. In the next subsection, we discuss accounting for interrupts.

5.4.2 Analysis Including Interrupt Overheads

Our method for accounting to interrupt-related overheads is identical to that proposed by Brandenburg (2011). We briefly review this accounting method here. We will then provide specific expressions for subjob budgets. In this section, we do not consider IPI-related or DLM-related overheads as "interrupt-related," but instead refer only to those overheads caused by timer ticks or (in the absence of a release master) job release interrupts.

We define a *segment* of a subjob as a contiguous interval of execution of a subjob, the overheads charged to it, and interrupt-related overheads.[2] However, *we only include overheads that run on the same CPU as the job*. For example, in Figure 5.3(c), there are no interrupt-related overheads, so τ_1 has segments of $[3\text{ ms}, 6\text{ ms})$, $[14\text{ ms}, 21\text{ ms})$, $[21\text{ ms}, 28\text{ ms})$, and $[28\text{ ms}, 31\text{ ms})$, τ_2 has segments of $[4\text{ ms}, 14\text{ ms})$, $[16\text{ ms}, 24\text{ ms})$, and $[34\text{ ms}, 35\text{ ms})$, and τ_3 has segments of $[6\text{ ms}, 16\text{ ms})$ and $[24\text{ ms}, 34\text{ ms})$.

Brandenburg (2011) provided separate analysis for the presence and absence of a release master. In this section, in order to avoid repetitive consideration of cases, we define new notation that will allow us to model either system with the same analysis. Brandenburg demonstrated that, for a segment that actually executes the subjob and non-interrupt-related overheads for C_i^{seg} time units, the segment itself (including interrupt-related overheads) can be as long as $C_i^{seg}/q + c^{pre}$ time units, where in the absence of a release master,

$$q \triangleq 1 - U_0^{tck} - \sum_{1 \le j \le n} U_j^{irq}, \tag{5.7}$$

and c^{pre} is defined in (2.4), and in the presence of a release master,

$$q \triangleq 1 - U_0^{tck}, \tag{5.8}$$

and c^{pre} is defined in (2.7).

This result leads immediately to the following additional properties for the accounting.

Property 5.9. *In subjob analytical budgets, terms that reflect job execution, Δ^{sch}, Δ^{cxs}, and Δ^{cpd} must be divided by q wherever they appear.*

[2] Brandenburg (2011) referred to this concept as a "subjob," but that term has a different definition in this chapter.

These are the terms that are included in the segment execution term C_i^{seg}, because (as can be seen in Figure 5.3(c)) these are the terms that are incurred on the CPU of the job.

Property 5.10. *A charge of c^{pre} must be included in the analytical budget of each subjob to account for a resuming previously preempted job that runs afterward.*

Property 5.10 is similar in nature to Property 5.5, but accounts for the increased exposure to interrupts that the previously preempted job incurs.

Property 5.11. *An additional charge of c^{pre} (beyond that required by Property 5.10) must be included in the analytical budget of each initial subjob.*

Property 5.11 is similar in nature to Property 5.8 and accounts for the first segment of the initial subjob.

The lateness analysis we will use is based on the *worst-case* analytical budget of any subjob. However, by Properties 5.7, 5.8, and 5.11, the analytical budgets of initial subjobs must include more overhead charges than those of non-initial subjobs. Therefore, we will assign actual budgets to initial subjobs that are *smaller* than for non-initial subjobs in order to compensate. As can be seen in Figure 5.3(d), it is then possible to ensure that all subjobs have identical analytical budgets, spreading out the overhead charges as evenly as possible. For simplicity, we first define the actual (analytical) budget for subjob $\tau_{i,k,v}$ as $C_{i,v}^{split}$ ($C_{i,v}^{split\prime}$). We initially provide expressions for analytical budgets *in terms of the to-be-determined actual budgets*. We then provide expressions for actual budgets that effectively spread out the overheads that appear in our expressions.

Throughout this section, we denote the pre-inflation terms

$$\begin{aligned} C_i^{split} &\triangleq \frac{C_i^{base}}{s_i}, \\ T_i^{split} &\triangleq \frac{T_i^{base}}{s_i}, \end{aligned} \tag{5.9}$$

and we add a prime to any variable to indicate the post-inflation version. (Subjob analytical budgets can be viewed as post-inflation actual budgets.)

By Property 5.6, we charge Δ^{ev} to the base task period before splitting and define

$$T_i^{split\prime} \triangleq \frac{T_i^{base} - \Delta^{ev}}{s_i}. \tag{5.10}$$

By Properties 5.5 and 5.7–5.11, for initial subjobs we define

$$C_{i,0}^{split\prime} \triangleq \frac{C_{i,0}^{split} + 2\cdot(\Delta^{sch} + \Delta^{cxs}) + \Delta^{cpd}}{q} + 2\cdot c^{pre} + \Delta^{ipi} + c^{rm}. \tag{5.11}$$

By Properties 5.5, 5.9, and 5.10, for $v \neq 0$, we define

$$C_{i,v}^{split\prime} \triangleq \frac{C_{i,v}^{split} + \Delta^{sch} + \Delta^{cxs} + \Delta^{cpd}}{q} + c^{pre}. \tag{5.12}$$

We would like to define actual subjob budgets in such a way as to average (5.11) and (5.12), replacing each coefficient of two in (5.11) with a coefficient of $1+\frac{1}{s_i}$ and adding a coefficient of $\frac{1}{s_i}$ to each of Δ^{ipi} and c^{rm}. We will show that the following allocation of actual budgets achieves this effect.

$$C_{i,v}^{split} \triangleq \begin{cases} C_i^{split} - \frac{s_i-1}{s_i}\cdot\left(\Delta^{sch} + \Delta^{cxs} + q\cdot(c^{pre} + \Delta^{ipi} + c^{rm})\right) & \text{if } v = 0, \\ C_i^{split} + \frac{1}{s_i}\cdot\left(\Delta^{sch} + \Delta^{cxs} + q\cdot(c^{pre} + \Delta^{ipi} + c^{rm})\right) & \text{if } v > 0. \end{cases} \tag{5.13}$$

(If $C_{i,0}^{split} < 0$, then we consider the system unschedulable.)

We first show that the total budget allocated for each job is its WCET. We have

$$\begin{aligned}
\sum_{v=0}^{s_i-1} C_{i,v}^{split} &= \{\text{By (5.13)}\} \\
&\quad C_i^{split} - \frac{s_i-1}{s_i}\cdot\left(\Delta^{sch} + \Delta^{cxs} + q\cdot(c^{pre} + \Delta^{ipi} + c^{rm})\right) \\
&\quad + \sum_{v=1}^{s_i-1}\left(C_i^{split} + \frac{1}{s_i}\cdot\left(\Delta^{sch} + \Delta^{cxs} + q\cdot(c^{pre} + \Delta^{ipi} + c^{rm})\right)\right) \\
&= \{\text{Rewriting, because the summation has } s_i - 1 \text{ terms}\} \\
&\quad s_i \cdot C_i^{split} \\
&= \{\text{By the definition of } C_i^{split} \text{ in (5.9)}\} \\
&\quad C_i.
\end{aligned}$$

We next show that all subjobs have the same *analytical* budget. First,

$$C_{i,0}^{split\prime} = \{\text{By (5.11)}\}$$

$$
\begin{aligned}
&\frac{C^{split}_{i,0} + 2\cdot(\Delta^{sch}+\Delta^{cxs}) + \Delta^{cpd}}{q} + 2\cdot c^{pre} + \Delta^{ipi} + c^{rm} \\
=\ &\{\text{By (5.13) with } v = 0\} \\
&\frac{C^{split}_{i} - \frac{s_i-1}{s_i}\left(\Delta^{sch}+\Delta^{cxs}+q\cdot\left(c^{pre}+\Delta^{ipi}+c^{rm}\right)\right) + 2\cdot(\Delta^{sch}+\Delta^{cxs}) + \Delta^{cpd}}{q} + 2\cdot c^{pre} + \Delta^{ipi} + c^{rm} \\
=\ &\{\text{Rewriting}\} \\
&\frac{C^{split}_{i} + \left(1+\frac{1}{s_i}\right)\cdot(\Delta^{sch}+\Delta^{cxs}) + \Delta^{cpd}}{q} + \left(1+\frac{1}{s_i}\right)\cdot c^{pre} + \frac{1}{s_i}\cdot(\Delta^{ipi}+c^{rm}),
\end{aligned}
\tag{5.14}
$$

and for $v \neq 0$,

$$
\begin{aligned}
C^{split\prime}_{i,v} =\ &\{\text{By (5.12)}\} \\
&\frac{C^{split}_{i,v} + \Delta^{sch} + \Delta^{cxs} + \Delta^{cpd}}{q} + c^{pre} \\
=\ &\{\text{By (5.13)}\} \\
&\frac{C^{split}_{i} + \frac{1}{s_i}\cdot\left(\Delta^{sch}+\Delta^{cxs}+q\cdot\left(c^{pre}+\Delta^{ipi}+c^{rm}\right)\right) + \Delta^{sch} + \Delta^{cxs} + \Delta^{cpd}}{q} + c^{pre} \\
=\ &\{\text{Rewriting}\} \\
&\frac{C^{split}_{i} + \left(1+\frac{1}{s_i}\right)\cdot\left(\Delta^{sch}+\Delta^{cxs}\right) + \Delta^{cpd}}{q} + \left(1+\frac{1}{s_i}\right)\cdot c^{pre} + \frac{1}{s_i}\cdot(\Delta^{ipi}+c^{rm}) \\
=\ &\{\text{By (5.14)}\} \\
&C^{split\prime}_{i,0},
\end{aligned}
$$

as desired. We therefore define

$$
\begin{aligned}
C^{split\prime}_{i} \triangleq\ &C^{split\prime}_{i,0} \\
=\ &\{\text{By (5.14)}\} \\
&\frac{C^{split}_{i} + \left(1+\frac{1}{s_i}\right)\cdot\left(\Delta^{sch}+\Delta^{cxs}\right) + \Delta^{cpd}}{q} + \left(1+\frac{1}{s_i}\right)\cdot c^{pre} + \frac{1}{s_i}\cdot(\Delta^{ipi}+c^{rm}).
\end{aligned}
\tag{5.15}
$$

5.4.3 Budget Accounting Mechanisms

In the last subsection, we defined the actual subjob budgets required to ensure the best lateness bounds. Additionally, at the beginning of Section 5.4, we discussed the requirement to add Δ^{cpd} or Δ^{cid} to the actual budget of each subjob whenever it resumes from a preemption or after an interrupt, respectively. In this subsection, we describe the actual mechanisms required to account for these budgets. These mechanisms are based on the $o_i(t)$ term that was defined in Section 5.3.

Whenever a job resumes after a preemption or interrupt, we need to increase the budget of the current subjob by Δ^{cpd} or Δ^{cid} units, respectively. We allocate the budget to the *current* subjob in order to mimic what would happen if the subjob were a "real" job, in that its execution would increase. We define the following budget accounting rules:

- **B1.** Every time τ_i is selected for execution, we subtract Δ^{cpd} from $o_i(t)$.
- **B2.** Every time τ_i resumes execution after an interrupt, we subtract Δ^{cid} from $o_i(t)$.
- **B3.** When τ_i is ready for execution after a change due to Rule B1 or Rule B2 (after any necessary context switch), we update the DLM timer as in Rule R2 in Section 5.3.

When they take effect, these rules will cause the current subjob to be treated as though it has used less execution time than it actually has, so it will be allowed to run for longer before the next split. For example, in Figure 5.3(b), $o_1(t)$ will be reduced by 1 ms by the scheduler executing at each of times 11, 19, and 26. (As discussed at the beginning of Section 5.4, this action is not actually necessary at times 19 and 26.)

Because the initial cache load from a base job is typically included in the job's execution time, we will initialize $o_i(t)$ on each job release to ensure that the job does not erroneously receive credit when it first begins.

Additionally, we must account for the fact that, by (5.13), the actual budgets for initial and non-initial subjobs differ. The basic idea for doing so (ignoring Rules B1–B3, which we account for later) is depicted in Figure 5.4, which we now describe. By (5.1), each base job's actual execution is divided into s_i identically sized subjobs. In Figure 5.4, the base job is divided into three identically sized subjobs. Because there may be multiple non-initial subjobs, if $s_i > 1$, then *for the purposes of the algorithm in Section 5.3*, we redefine

$$C_i^{base} \triangleq s_i \cdot C_{i,1}^{split}.$$

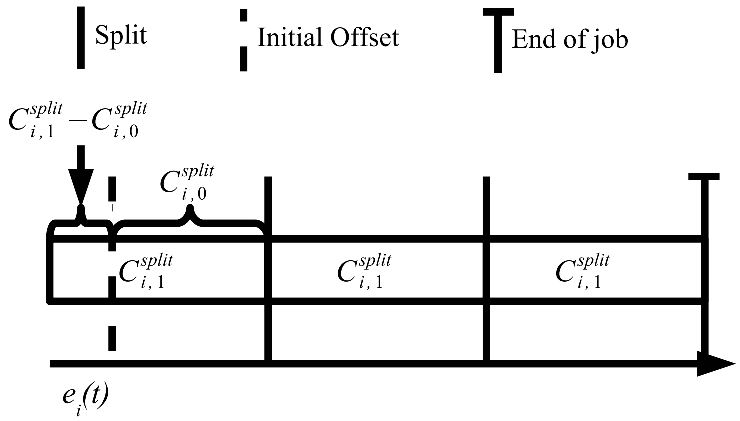


Figure 5.4: Illustration of initial offset $o_i(t)$ for a job with $s_i > 0$. Observe that the axis is with respect to the job's execution time, rather than the wall-clock time.

This ensures that all subjobs *except* the initial subjob receive the correct actual budgets. In order to achieve the correct budget for the initial subjob, we add $C_{i,1}^{split} - C_{i,0}^{split}$ to the initial $o_i(t)$, which ensures that the initial subjob receives $C_{i,0}^{split}$ units of actual budget, as desired. It is as if the job had already executed up to the dotted vertical line in Figure 5.4.

In order to apply this technique in the presence of Rules B1–B3, if a base job $\tau_{i,k}$ becomes eligible at time t (either because it is released or because its predecessor has completed), we define

$$o_i(t) = \begin{cases} \Delta^{cpd} & \text{if } s_i = 1, \\ \Delta^{cpd} + C_{i,1}^{split} - C_{i,0}^{split} & \text{if } s_i > 1. \end{cases}$$

To see how our mechanism works, consider the job of τ_1 in Figure 5.3(b). Because we do not consider interrupt-related overheads ($U_0^{tck} = 0$ and $C_0^{tck} = 0$) but do consider a release master, $q = 1$ by (5.8), $c^{pre} = 0$ by (2.7), and $c^{rm} = \Delta^{rel}$ by (5.6). Recall that C_1^{base} is the maximum execution of each job of τ_1 *before* accounting for overheads. For the base job of τ_1 in Figure 5.3(b), it bounds the execution of the job that either does not involve an overhead, or that involves an *unnecessary cpd* credit. Observe that this occurs for 16 ms in total, so we use $C_1^{base} = 16$ ms. Therefore,

$$C_1^{split} = \{\text{By (5.9)}\}$$
$$\frac{C_1^{base}}{s_i}$$

$$
\begin{aligned}
&= \{\text{Substituting actual values}\}\\
&\quad \frac{16\text{ ms}}{4}\\
&= \{\text{Simplifying}\}\\
&\quad 4\text{ ms}. \qquad (5.16)
\end{aligned}
$$

We have

$$
\begin{aligned}
C_{1,0}^{split} &= \{\text{By (5.13)}\}\\
&\quad C_{1}^{split} - \frac{s_1 - 1}{s_1} \cdot \left(\Delta^{sch} + \Delta^{cxs} + q \cdot (c^{pre} + \Delta^{ipi} + c^{rm})\right)\\
&= \{\text{Substituting } s_1 = 4,\ q = 1,\ c^{pre} = 0, \text{ 1 ms overheads, and (5.16)}\}\\
&\quad 4\text{ ms} - \frac{3}{4} \cdot (2\text{ ms} + 1 \cdot 2\text{ ms})\\
&= \{\text{Simplifying}\}\\
&\quad 1\text{ ms}, \qquad (5.17)
\end{aligned}
$$

and for $v \neq 0$,

$$
\begin{aligned}
C_{1,v}^{split} &= \{\text{By (5.13)}\}\\
&\quad C_{1}^{split} - \frac{1}{s_1} \cdot \left(\Delta^{sch} + \Delta^{cxs} + q \cdot (c^{pre} + \Delta^{ipi} + c^{rm})\right)\\
&= \{\text{Substituting } s_1 = 4,\ q = 1,\ c^{pre} = 0, \text{ 1 ms overheads, and (5.16)}\}\\
&\quad 4\text{ ms} + \frac{1}{4} \cdot (2\text{ ms} + 1 \cdot 2\text{ ms})\\
&= \{\text{Simplifying}\}\\
&\quad 5\text{ ms}. \qquad (5.18)
\end{aligned}
$$

Therefore, for the purposes of the algorithm in Section 5.3, we use $C_1^{base} = s_1 \cdot C_{1,1}^{split} = 20$ ms, and we use $o_1(t) = \Delta^{cpd} + C_{1,1}^{split} - C_{1,0}^{split} = 5$ ms initially. The scheduler that runs at time 3 in Figure 5.3(b) will then, by Rule B1, subtract $\Delta^{cpd} = 1$ ms from $o_i(t)$, so that $o_1(t) = 4$ ms for $t \in [4\text{ ms}, 12\text{ ms})$ (until the end of the next scheduler invocation that selects τ_1). At time 5, by Rule B3, the DLM timer needs to be set properly according to the definition of $v_1(5)$ in (5.2). Because the job has not yet actually begun execution at time 5,

$e_1(5) = 0$. We thus have

$$\begin{aligned}
\lambda_1(5) &= \{\text{By the definition of } \lambda_1(5) \text{ in (5.1)}\} \\
&\quad \left\lfloor \frac{s_1 \cdot (e_1(5) + o_1(5))}{C_i^{base}} \right\rfloor \\
&= \{\text{Substituting } s_1 = 4,\ e_1(5) = 0 \text{ ms},\ o_1(5) = 4 \text{ ms, and } C_i^{base} = 20 \text{ ms}\} \\
&\quad \left\lfloor \frac{4 \cdot (0 \text{ ms} + 4 \text{ ms})}{20 \text{ ms}} \right\rfloor \\
&= \{\text{Simplifying}\} \\
&\quad 0.
\end{aligned}$$

We have not yet observed a higher value of $\lambda_1(t)$ within the same job, so $j_1(5) = 0$ as well. Therefore,

$$\begin{aligned}
v_1(5) &= \{\text{By the definition of } v_1(5) \text{ in (5.2)}\} \\
&\quad 5 + \frac{(j_1(5) + 1)C_i^{base}}{s_1} - (e_1(5) + o_1(5)) \\
&= \{\text{Substituting } j_1(5) = 0,\ C_i^{base} = 20 \text{ ms},\ s_1 = 4,\ e_1(5) = 0 \text{ ms, and } o_1(5) = 4 \text{ ms}\} \\
&\quad 5 + \frac{(0 + 1) \cdot 20 \text{ ms}}{4} - (0 \text{ ms} + 4 \text{ ms}) \\
&= \{\text{Simplifying}\} \\
&\quad 6.
\end{aligned}$$

Thus, the DLM timer is scheduled to fire at time 6. At that time, because the job has executed for an additional 1 ms, $e_i(t) = 1$ ms. Therefore,

$$\begin{aligned}
\lambda_1(6) &= \{\text{By the definition of } \lambda_1(6) \text{ in (5.1)}\} \\
&\quad \left\lfloor \frac{s_1 \cdot (e_1(6) + o_1(6))}{C_i^{base}} \right\rfloor \\
&= \{\text{Substituting } s_1 = 4,\ e_1(6) = 1 \text{ ms},\ o_1(6) = 4 \text{ ms, and } C_i^{base} = 20 \text{ ms.}\} \\
&\quad \left\lfloor \frac{4 \cdot (1 \text{ ms} + 4 \text{ ms})}{20 \text{ ms}} \right\rfloor \\
&= \{\text{Simplifying}\} \\
&\quad 1. \qquad (5.19)
\end{aligned}$$

Therefore, the initial subjob has ended at time 6.

Then, when the scheduler runs again at time 11, due to Rule B1, it subtracts 1 ms from $o_1(t)$, so that $o_1(t) = 3$ ms for $t \in [12\text{ ms}, 20\text{ ms})$. At time 13, by Rule B3, the DLM timer needs to be set properly according to the definition of $v_1(6)$ in (5.2). Because the job has not executed since time 6, when $e_1(6) = 1$ ms, we also have $e_1(13) = 1$ ms. We thus have

$$\begin{aligned}
\lambda_1(13) &= \{\text{By the definition of } \lambda_1(13) \text{ in (5.1)}\} \\
&\quad \left\lfloor \frac{s_1 \cdot (e_1(13) + o_1(13))}{C_i^{base}} \right\rfloor \\
&= \{\text{Substituting } s_1 = 4,\ e_1(13) = 1 \text{ ms},\ o_1(5) = 3 \text{ ms, and } C_i^{base} = 20 \text{ ms}\} \\
&\quad \left\lfloor \frac{4 \cdot (1 \text{ ms} + 3 \text{ ms})}{20 \text{ ms}} \right\rfloor \\
&= \{\text{Simplifying}\} \\
&\quad 0.
\end{aligned}$$

In this case, $\lambda_1(13)$ is *not* the largest value of $\lambda_1(t)$ that has been observed while this job has been ready for execution and incomplete. In particular, we demonstrated in (5.19) that $\lambda_1(6) = 1$. Therefore, $j_1(13) = 1$. Thus,

$$\begin{aligned}
v_1(13) &= \{\text{By the definition of } v_1(13) \text{ in (5.2)}\} \\
&\quad 13 + \frac{(j_1(13)+1)C_i^{base}}{s_1} - (e_1(13) + o_1(13)) \\
&= \{\text{Substituting } j_1(13) = 1,\ C_i^{base} = 20 \text{ ms},\ s_1 = 4,\ e_1(13) = 1 \text{ ms, and } o_1(13) = 3 \text{ ms}\} \\
&\quad 13 + \frac{(1+1) \cdot 20 \text{ ms}}{4} - (1 \text{ ms} + 3 \text{ ms}) \\
&= \{\text{Simplifying}\} \\
&\quad 19.
\end{aligned}$$

Thus, the DLM timer is scheduled to fire at time 19. At that time, because the job has executed for an additional 6 ms since time 6 (of which 1 ms was due to the *cpd* credit), $e_i(t) = 7$ ms. Thus,

$$\lambda_1(19) = \{\text{By the definition of } \lambda_1(19) \text{ in (5.1)}\}$$

$$\left\lfloor \frac{s_1 \cdot (e_1(19) + o_1(19))}{C_i^{base}} \right\rfloor$$

$$= \{\text{Substituting } s_1 = 4,\ e_1(19) = 7 \text{ ms},\ o_1(19) = 3 \text{ ms, and } C_i^{base} = 20 \text{ ms.}\}$$

$$\left\lfloor \frac{4 \cdot (7 \text{ ms} + 3 \text{ ms})}{20 \text{ ms}} \right\rfloor$$

$$= \{\text{Simplifying}\}$$

$$2.$$

Therefore, the second subjob has ended at time 19. Thus, its 5 ms budget has been correctly inflated to account for the extra *cpd* charge, and it has run for 6 ms.

5.5 Handling Critical Sections

One of the advantages of GEL schedulers is that they are *job-level static priority* (*JLSP*) algorithms, which is important for locking protocols (Brandenburg, 2011). However, when splitting is introduced, a GEL algorithm is no longer truly JLSP. If a subjob ends while waiting for or holding a lock, then the priority of the underlying job is changed, potentially violating the assumptions of locking protocols. Furthermore, if a locking protocol operates nonpreemptively, then it is not possible to split a job while it is waiting for or holding a critical section. Fortunately, we can solve both problems by simply extending subjob budgets for as long as a resource request is active. A similar technique was proposed previously by Ghazalie and Baker (1995) for aperiodic servers.

In order to support the necessary budget extensions, we use a more complicated set of rules than those described in Section 5.3. To illustrate the behavior of our modified algorithm, we present in Figure 5.5 a modification of the schedule from Figure 5.1(c) with the addition of critical sections. Our new rules allow the budget for a subjob to be extended when its DLM is delayed. Furthermore, because this delay does not change the expressions for $j_i(t)$, $v_i(t)$, $\rho_i(t)$, or $d_i(t)$, the next subjob implicitly has its budget shortened. Essentially, we are only allowing each DLM to "lag" behind the ideal DLM by at most C_i^{ext} units of the corresponding base job's execution. (Recall that C_i^{ext} was defined in Section 5.2 to be the total amount of time that a job could be waiting for or holding a critical section.) It is even possible for a subjob to be implicitly skipped by this mechanism if $C_i^{ext} > C_i^{split}$.

- **R1.** If a job of τ_i^{base} is released at time t, then $\delta_i(t)$ is assigned to $d_i(t)$.

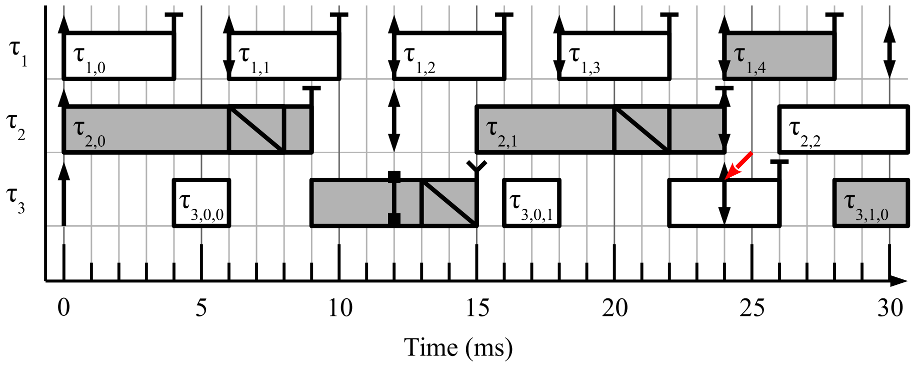


Figure 5.5: G-EDF schedule of the same system as Figure 5.1, with the same key, where τ_3 is split into two subjobs, the other tasks are not split, and critical sections are present.

This rule is identical to Rule R1 from Section 5.3.

- **R2.** Whenever a non-final subjob of τ_i^{base} is scheduled at time t to run on a CPU, a *DLM timer* is assigned to force a reschedule on that CPU at time $v_i(t)$. Whenever τ_i^{base} is preempted, or τ_i^{base} requests a resource, the DLM timer is cancelled.

  In the schedule in Figure 5.5, the DLM timer for τ_3 is set at time 9 to fire at time 14, but is cancelled at time 13 when τ_3^{base} requests a resource. Because only a final subjob remains after time 15, however, the timer will not be set again.

- **R3.** Whenever a critical section ends, if $d_i(t) > \delta_i(t)$, then a reschedule is forced.

  Observe that for $t \in [14, 15)$ in Figure 5.5, the current subjob of $\tau_{3,0}$ ($\tau_{3,0,0}$) is an earlier subjob than that indexed by $j_3(t)$. Thus, when the critical section ends, a DLM should occur. Triggering a reschedule will cause the needed DLM.

- **R4.** If the scheduler is called at time t on a CPU that was running τ_i^{base}, and τ_i^{base} is neither waiting for nor holding a resource, then $\delta_i(t)$ is assigned the value $d_i(t)$, potentially causing a DLM.

  This rule is nearly identical to Rule R3 in Section 5.3 and functions the same way, except in the case that the scheduler is invoked due to a job release from another task while τ_i^{base} is waiting for or holding a resource. However, if a DLM does occur, then the scheduler could have been invoked either due to Rule R2 or Rule R3 as modified above. In Figure 5.5 it is invoked due to Rule R3 at time 15.

We let C_i^{split} denote the ideal WCET of a subjob, ignoring critical sections. When we account for critical sections, a single subjob of a job from τ_i can run for as long as $C_i^{split} + C_i^{ext}$. Nonetheless, τ_i's processor share over the long term is not affected, because the total execution of all subjobs of a base job must equal the execution of that base job. The lateness analysis from Chapter 3 is described in the next section in a modified form accounting for critical sections. The resulting lateness bounds are increased by approximately C_i^{ext} with no utilization loss.

5.6 G-FL Lateness Bounds

When we account for critical sections, a single subjob of $\tau_{i,k}$ can run for as long as $C_i^{split} + C_i^{ext}$ time units. Nonetheless, τ_i's processor share over the long term is not affected, because the total execution of all subjobs must be the execution of the base job. Intuitively, $\delta_i(t)$ can lag behind $d_i(t)$ with respect to $e_i(t)$ (rather than t) for the bounded amount of at most C_i^{ext} units of execution of the base job.

Using $C_i^{split} + C_i^{ext}$ in the sporadic task model would be overly pessimistic, because it would unnecessarily increase the long-term utilization used in the analysis. Therefore, we extend the sporadic task model to reduce this pessimism. We use the function DBF($\tau_i^{split}, Y_i^{split}, \Delta$), defined as the maximum demand of jobs from τ_i that have both release times and absolute PPs in any interval of length Δ.

We restate (3.8) for split tasks:

$$S_i^{split}(Y_i^{split}) \triangleq C_i^{split} \cdot \max\left\{0, 1 - \frac{Y_i^{split}}{T_i^{split}}\right\}. \tag{5.20}$$

We now extend Lemmas 3.1 and 3.2 to our task model for split jobs where each split can be delayed at most C_i^{ext} time units. For G-FL, we use $Y_i^{split} = T_i^{split} - \frac{m-1}{m}(C_i^{split} + C_i^{ext})$. We still define $U_i^{split} = C_i^{split}/T_i^{split}$ and $S_i^{split}(Y_i^{split})$ using (5.20), but allow a job to run for as long as $C_i^{split} + C_i^{ext}$ time units. We first provide an extension of Lemma 3.1 to our slightly more general task model.

Lemma 5.1. *If $\Delta \geq Y_i$, then*

$$DBF(\tau_i^{split}, Y_i^{split}, \Delta) \leq U_i^{split}\Delta + C_i \cdot \left(1 - \frac{Y_i}{T_i}\right) + C_i^{ext}.$$

Proof. Let I be a worst-case interval of length Δ, i.e., such that jobs of τ_i^{split} with releases and absolute PPs in I demand DBF($\tau_i^{split}, Y_i^{split}, \Delta$) units of execution. If we assume $C_i^{ext} = 0$, then by Lemma 3.1, there are at most $U_i^{split}t + C_i \cdot \left(1 - \frac{Y_i}{T_i}\right)$ units of execution demanded in I. Allowing $C_i^{ext} > 0$ essentially allows a subjob to "steal" budget from its successor, which increases the demand in the interval if the stolen budget is from outside the interval. For example, when analyzing τ_3^{split} within $I = [0, 13)$ in Figure 5.5, only $\tau_{3,0,0}$ has both a release and an absolute PP in I, but 1 ms of budget from $\tau_{3,0,1}$ is stolen, and thus contributes to the demand from jobs of τ_3^{split} in I. By our definition of worst-case demand, the most demand that can enter I would be C_i^{ext} units of demand added to the last subjob within I. The lemma follows. □

We next provide an extension of Lemma 3.2 to our task model.

Lemma 5.2. *$DBF(\tau_i^{split}, Y_i^{split}, \Delta) \le U_i^{split}\Delta + S_i^{split}(Y_i^{split}) + C_i^{ext}$.*

Proof. Using identical reasoning to Lemma 5.1, but using Lemma 3.2 in place of Lemma 3.1. □

The numerator in (3.13) measures the competing work remaining at the PP of a job under analysis. When applied to τ_i^{split}, it is based on analyzing DBF($\tau_i^{split}, Y_i^{split}, \Delta$) over a particular interval for each τ_i^{split} using Lemmas 3.1 and 3.2. By comparing Lemmas 3.1 and 3.2 to Lemmas 5.1 and 5.2, we see that adding $\sum_{\tau_j \in \tau} C_j^{ext}$ to the numerator in (3.13) accounts for the additional demand. In addition, because a job of τ_i^{split} can run for $C_i^{split} + C_i^{ext}$ time units, each C_i or C_j should be replaced with $C_i^{split} + C_i^{ext}$ or $C_j^{split} + C_j^{ext}$, respectively. We now have:

$$G(\vec{x}, \vec{Y}^{split}) = \sum_{m-1 \text{ largest}} (x_j U_j^{split} + C_j^{split} + C_j^{ext} - S_j^{split}(Y_j^{split}))$$

$$x_i = \frac{G(\vec{x}, \vec{Y}^{split}) + S(\vec{Y}^{split}) + \sum_{\tau_j \in \tau} C_j^{ext} - (C_i^{split} + C_i^{ext})}{m} \tag{5.21}$$

and, for each task τ_i^{split}, a lateness bound of $x_i + C_i^{split} + C_i^{ext} - T_i^{split} + Y_i^{split}$. Observe that while we needed to inflate C_i^{split} values, we did not need to inflate U_i^{split} values. Because utilization values rather than execution-time values are used to determine whether tardiness bounds exist, no capacity loss occurs.

5.7 Locking Overheads

In order to account for overheads from locking, we used the same strategy as Brandenburg (2011). As Brandenburg did, we assumed that a release master is used. In this chapter, we consider only the non-preemptive mutex queue spinlock locking protocol, as discussed in Sections 2.2.2 and 2.2.3. Because determining which subjob will access a critical section may not be possible, we pessimistically assumed that every subjob accesses every critical section. This behavior is not actually possible, but our assumption provides a safe upper bound.

We used the same locking and overhead analysis procedure discussed in Section 2.2.3. The analysis of $lower(\tau_i)$ discussed in that section assumes that jobs do not begin execution before their release times, which is not true under our splitting mechanism, so we pessimistically assumed that *every* task in τ_i's cluster, other than τ_i itself, is in $lower(\tau_i)$. Furthermore, we defined G-FL priorities based on *inflated* values of $C_i^{split\prime}$, which may change after the locking analysis, so the analysis of GEL schedulers provided in Section 2.2.3 also does not apply to unsplit task systems, and we used the same pessimistic choice of $lower(\tau_i)$.

During the locking analysis phase, we use the following formula to compute the maximum amount of time that τ_j spends waiting for a lock on resource ψ_q due to jobs in cluster v.

$$spin''_j(\tau_j, v, \psi_q) = \begin{cases} total(c_v, tifs(\theta_v, \psi_q, R_j, 1)) & \text{if cluster } v \text{ is not } \tau_j\text{'s cluster,} \\ total(c_v - 1, tifs(\theta_v \setminus \{\tau_j\}, \psi_q, R_j, 1)) & \text{if cluster } v \text{ is } \tau_j\text{'s cluster.} \end{cases} \tag{5.22}$$

(5.22) is very similar to (2.9), which computes the same quantity *assuming that some job of* τ_i *is pi-blocked by* τ_j. Thus, (2.9) was derived assuming that τ_i is not currently running, so the second case within (2.9) does not consider the possibility that τ_j is waiting for a request of ψ_q from τ_i. For example, consider the situation depicted in Figure 5.6, where τ_2 begins spinning at time 6 for a lock held by τ_1. This situation *cannot* cause pi-blocking for τ_1 within [5 ms, 9 ms), because in it, τ_1 is already running. A similar situation would occur if τ_1 were also spinning, contending for the same lock as τ_2, but in that case, τ_1 would be s-blocked rather than pi-blocked. By contrast, we do not exclude from θ_v such a τ_i, as we want to know the absolute maximum time that τ_j can be blocked waiting for ψ_q, and the situation depicted in Figure 5.6 may be the case causing such maximal blocking. To obtain the required bound for a given ψ_q, we simply sum $spin''(\tau_j, v, \psi_q)$ over all clusters and add $L_{j,q}$. To obtain C_j^{ext}, we simply select the largest such bound for any resource.

During the phase analyzing non-locking-related overheads, we use (5.10) and (5.15) with (5.6) and (5.8) to inflate parameters. In addition, a subjob of $\tau_{i,k}$ may use up to C_i^{ext} units of extra actual budget while waiting for or holding a lock. Because actual budgets are divided by q in (5.11) and (5.12), we use $C_i^{ext\prime} = C_i^{ext}/q$ in place of C_i^{ext} for our final response-time bound computation.

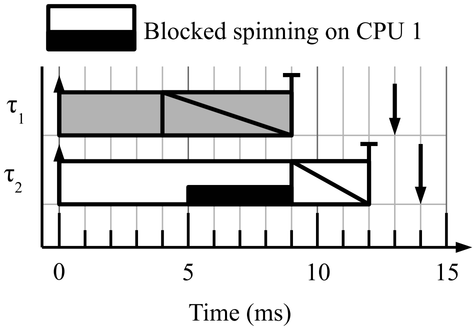


Figure 5.6: Scenario that would not be considered in the analysis for τ_1 in Section 2.2.2 but that we must consider. Other than the additional symbol noted at the top, this figure has the same key as Figure 5.1.

5.8 Experiments

As stated at the beginning of this chapter, we discussed G-EDF in prior sections only for ease of exposition. In this section, we instead consider G-FL (based on *inflated* values $C_i^{split\prime}$), which has better maximum lateness bounds. Brandenburg (2011) showed that, when the effects of overheads are considered, the best scheduling strategy for bounded tardiness is *clustered* scheduling, where CPUs are grouped along cache boundaries and a global scheduling algorithm is used within each cluster. We thus consider *clustered fair lateness* (*C-FL*), where G-FL is used within each cluster.

Our experiments consisted of two phases. In the first phase, we measured overheads from an actual implementation of C-FL. In the second phase, we used these measurements as part of an overhead-aware schedulability study. This two-phase methodology has been used in several prior studies (e.g., see (Brandenburg, 2011)).

First Phase: Overhead Measurement. We performed our overhead measurements on the system depicted in Figure 5.7, which is a 24-core Xeon L7455 (2.13GHz) system with 64GB of RAM. On that system, pairs of cores share an L2 cache and the cores on each six-core chip share an L3 cache. We considered four variants of C-FL. Under C-FL-L2, CPUs are grouped at the level of L2 caches, and all processors both receive interrupts and execute tasks. Under C-FL-L2-RM, CPUs are still grouped at the level of L2 caches, but one CPU operates as a release master. C-FL-L3 and C-FL-L3-RM are analogous to C-FL-L2 and C-FL-L2-RM, respectively, but group CPUs according to L3 caches rather than L2 caches.

For the first phase of our experiments, we implemented all four C-FL variants with splitting and measured relevant overheads in a manner similar to the overhead measurements by Brandenburg (2011). Recall that

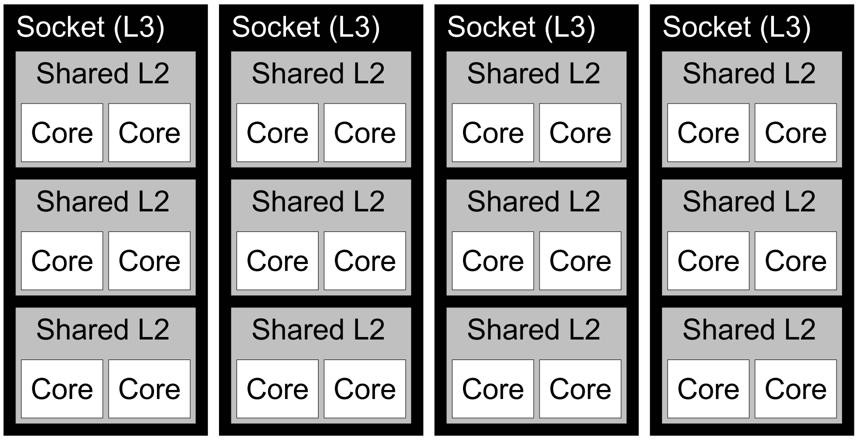


Figure 5.7: Architecture of machine used for overhead experiments.

the relevant overheads are described in Section 2.2, and our modifications to the accounting method are described in Section 5.4. Because the techniques used for adjusting offsets require overhead numbers, our implementation did not adjust offset parameters.

Second Phase: Schedulability Study. For the second phase of our experiments, we performed a schedulability study considering the same 24-core platform considered in the first phase.

We assessed schedulability trends by generating implicit-deadline task sets based on the experimental design from Brandenburg (2011). We generated task utilizations using either a uniform, a bimodal, or an exponential distribution. For task sets with uniformly distributed utilizations, we used either a *light* distribution with values randomly chosen from $[0.001, 0.1]$, a *medium* distribution with values randomly chosen from $[0.1, 0.4]$, or a *heavy* distribution with values randomly chosen from $[0.5, 0.9]$. For tasks sets with bimodally distributed utilizations, values were chosen uniformly from either $[0.001, 0.5]$ or $[0.5, 0.9]$, with respective probabilities of $8/9$ and $1/9$ for *light* distributions, $6/9$ and $3/9$ for *medium* distributions, and $4/9$ and $5/9$ for *heavy* distributions. For task sets with exponentially distributed utilizations, we used exponential distributions with a mean of 0.10 for *light* distributions, 0.25 for *medium* distributions, or 0.50 for *heavy* distributions. Utilizations were drawn until one was generated between 0 and 1. We generated integral task periods using a uniform distribution from $[3ms, 33ms]$ for *short* periods, $[10ms, 100ms]$ for *moderate* periods, or $[50ms, 250ms]$ for *long* periods.

When assessing schedulability in the presence of locking, critical sections were chosen uniformly from either $[1\mu s, 15\mu s]$ for *short* critical sections, $[1\mu s, 100\mu s]$ for *medium* critical sections, or $[5\mu s, 1280\mu s]$

for *long* critical sections. We varied the lock contention via two parameters, n_r, which is the number of resources, and p_{acc}, which is the probability that any task accesses a given resource. We performed tests with $n_r = 6$ and $n_r = 12$, and with $p_{acc} = 0.1$ and $p_{acc} = 0.25$. For a task using a given resource, we generated the number of accesses uniformly from the set $\{1,2,3,4,5\}$. These parameter choices are a subset of those used by Brandenburg (2011) because, unlike Brandenburg, we opted to perform experiments on a larger variety of working set sizes (see below) to facilitate better comparisons with experiments without locking. A prior implementation study (Brandenburg and Anderson, 2007) demonstrated that for typical soft real-time applications, the majority of critical sections are less than $10\mu s$. Therefore, the short critical section distribution is likely to be the most common in many settings.

For each tested set of distribution parameters, we generated 100 task sets for each utilization cap of the form $\frac{24i}{20}$ where i is an integer in $[1,20]$, in order to get a reasonable spread of utilization caps. Tasks were generated until one was created that would cause the system to exceed the given utilization cap. That task was then discarded. We tested each task set for each choice of working set size (WSS) within $\{2^2KB, 2^3KB, \ldots, 2^{11}KB\}$. The WSS for a base job is defined as its total memory footprint, and the WSS for each subjob was assumed to be the same as that of its base job. For tests not involving locking, we considered each combination of task set and WSS under each of the four C-FL variants. For tests involving locking, we instead considered the two C-FL variants using a release master, and we assumed that the mutex queue spinlock locking protocol described in Section 2.2.2 was used.

For each combination of task set and parameters, we first assigned the tasks to clusters using a worst-fit decreasing heuristic: we ordered tasks by decreasing utilization, and we placed each task in order on the CPU with the most remaining capacity. This heuristic was intended to maximize the available utilization for overheads. We then evaluated tardiness without splitting by first inflating task parameters to account for overheads and locking using the methods from Brandenburg (2011) (as reviewed in Section 2.2), and then using the inflated task system in the lateness bound computation method in Chapter 3. We ignored task sets that were either not schedulable under C-FL (without splitting) or resulted in zero tardiness, because our goal was to show improvements upon previously available schedulers. The number of ignored task sets varied greatly with respect to the utilization distribution, period distribution, and utilization cap. For example, very large utilization caps usually led to unschedulable systems once overheads were accounted for, even without splitting.

```
   Input: Task system $\tau$, cluster $c$, old maximum lateness old_max_late
   Output: The new maximum lateness after splitting, or $\perp$ if no improvement is possible
 1 split_candidates := get_split_candidates($\tau, c$)
 2 for $\tau_i \in$ split_candidates do
 3     if not split_saturated$_i$ then
 4         $s_i := s_i + 1$
 5         max_late := comp_max_lateness($\tau$)
 6         if max_late $= \perp$ then
 7             $s_i := s_i - 1$
 8             split_saturated$_i$ := true
 9         else if max_late < old_max_late then
10             return max_late
11         else
12             $s_i := s_i - 1$
13 return $\perp$
```

Algorithm 5.1: Function `try_one_split`

For each considered task set, we applied task splitting using the heuristic described below. When computing tardiness bounds after splitting, we first inflated the task parameters for overheads and locking as before, but the method for inflating parameters was modified according to Sections 5.4, 5.5, and 5.7. We compared the maximum tardiness bounds for the inflated task sets before and after splitting. (Because $s_i = 1$ is allowed by our algorithm, every considered task set is scheduable under C-FL with splitting by definition.)

Heuristic for Determining s_i. In order to use splitting to reduce tardiness bounds, it is necessary to determine appropriate s_i values. To do so, we used a simple heuristic algorithm, described in Algorithms 5.1 and 5.2. The function `comp_max_lateness()` accounts for overheads and returns the maximum lateness for the entire task system if it is schedulable and $\perp$ if it is not. It uses the methods described in Sections 5.4, 5.6, and (for experiments involving locking) 5.5 and 5.7. The function `get_split_candidates()` returns a list of tasks in a cluster ordered by contribution to the lateness bounds. Tasks that contribute to $G(\vec{x}, \vec{Y})$ (see Section 5.6) are ordered by decreasing contribution to $G(\vec{x}, \vec{Y})$, and the remaining tasks are ordered by decreasing $S_i^{split}(Y_i^{split})$.

The function `try_one_split` (Algorithm 5.1) attempts to reduce the lateness bound for the system by increasing a single s_i by 1. It may make multiple attempts before finding a splitting that actually reduces the maximum lateness bound for the system. For each task τ_i attempted, the algorithm first increases s_i in Line 4. If this increase improves the maximum lateness bound, the function returns in Line 10, so no other s_i values will be affected. Otherwise, if the increase either results in an unschedulable system or simply no improvement to the maximum lateness bound, it is reverted (Lines 7 and 12). To improve efficiency,

```
   Input: Task system $\tau$
   Output: The new maximum lateness after splitting
 1 max_late := comp_max_lateness($\tau$)
 2 keep_splitting := true
 3 while keep_splitting do
 4     keep_splitting := false
 5     clust_list := clusters sorted by decreasing maximum lateness
 6     for $c \in$ clust_list do
 7         new_late := try_one_split($\tau, c,$ max_late)
 8         if new_late $\neq \perp$ then
 9             max_late := new_late
10             keep_splitting := true
11 return max_late
```

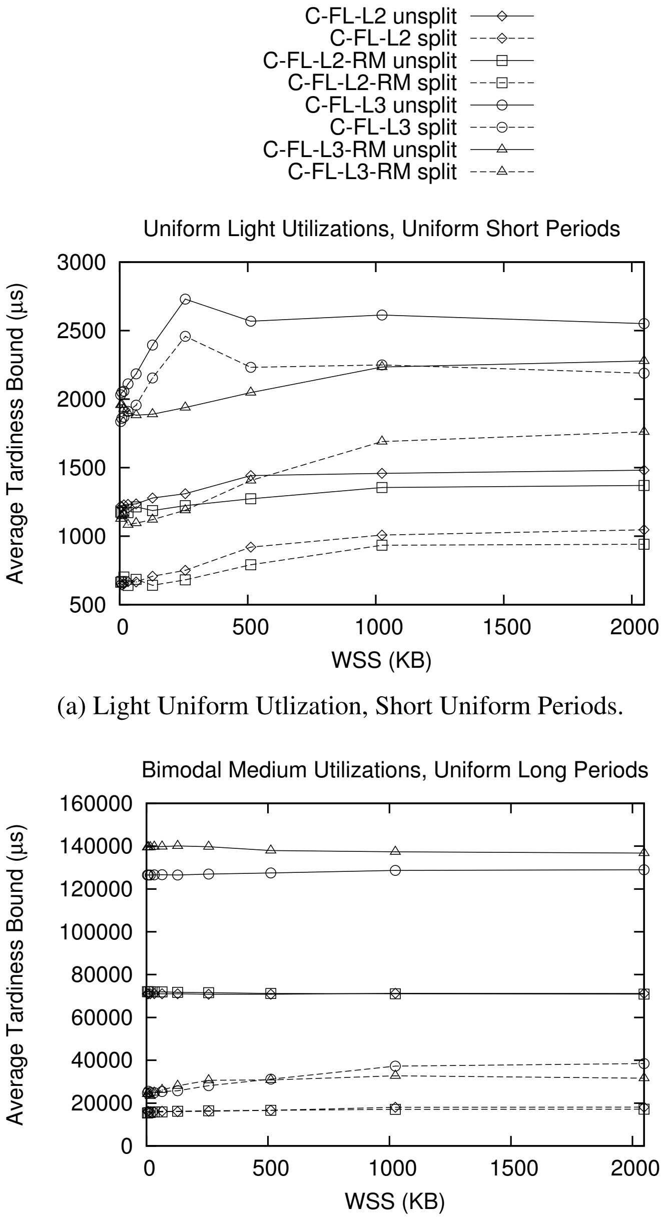
Algorithm 5.2: Splitting heuristic

try_one_split uses the per-task variable split_saturated$_i$, which is initially assumed to be **false**, but which is set to **true** in Line 8, when increasing s_i by 1 would result in an unschedulable system. The value of split_saturated$_i$ is assumed to persist between calls to try_one_split. The purpose of accounting for split_saturated$_i$ is to avoid repeatedly trying to further split τ_i.

Algorithm 5.2 computes values for s_i for all τ_i. It consists of a loop (Lines 3–10) that is run repeatedly until the maximum lateness bound has stopped improving. Within this loop, the algorithm calls try_one_split on each cluster, ordered by the largest to the smallest maximum lateness bound. The purpose of calling try_one_split on multiple clusters is that in a system with locking, reducing the lateness bound in one cluster may reduce the lateness bounds for other clusters. Thus, even if there is no task to split in the cluster with the largest maximum lateness bound, it may be possible to reduce the lateness bound for that cluster by reducing the lateness bounds in a different cluster.

Results. Examples of results without locking are depicted in Figure 5.8, where each graph plots average maximum tardiness bounds as a function of WSS.[3] Observe that improvements over 25% are common, and can be nearly 100% in some cases. Because task systems with higher WSSs are more likely to be unschedulable even without splitting, higher WSSs often represent significantly smaller groups of task sets and are skewed towards task sets with smaller utilizations. This can cause a nonincreasing trend in the tardiness bounds with increased WSS for C-FL, but our purpose is to compare the effect of splitting when bounded tardiness is already achievable by C-FL. An overall trend from our experiments is that splitting

[3] Additional results can be found at `http://cs.unc.edu/~anderson`. In total, our experiments resulted in several hundred graphs.



(a) Light Uniform Utlization, Short Uniform Periods.

(b) Medium Bimodal Utilization, Long Uniform Periods.

Figure 5.8: Schedulability results with respect to WSS in the absence of locking.

provides more benefit when jobs are longer (larger utilizations and longer periods). This phenomenon occurs because the additional overheads from splitting are proportional to the split factor rather than job length, so the additional overheads are relatively smaller in comparison to longer jobs.

Figure 5.9 has the same key as Figure 5.8, but plots the average maximum tardiness bound as a function of the system utilization cap rather than WSS. Observe that the bounds with splitting (dashed lines) tend to grow more slowly than the bounds without splitting (solid lines) until they grow drastically before all tested

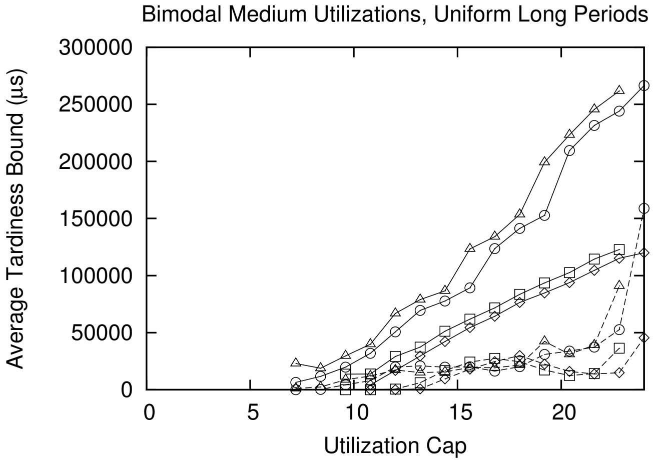


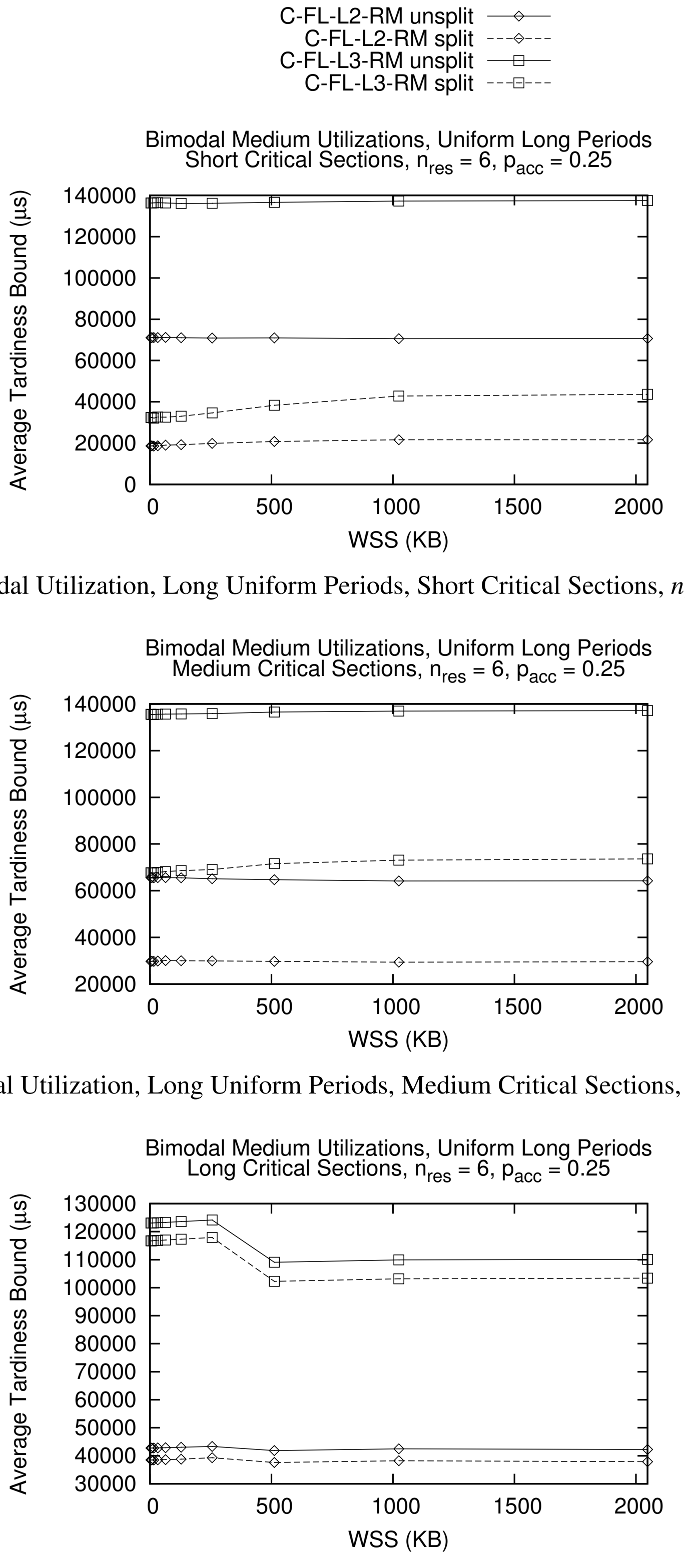
Figure 5.9: Medium Bimodal Utilization, Long Uniform Periods, WSS = 128KB. This graph is with respect to utilization cap instead of WSS.

task sets are unschedulable. This phenomenon occurs because the overheads from splitting use some of the system's remaining utilization, and when very little utilization is available tasks cannot be split as finely.

Figure 5.10 has a distinct key from that of Figure 5.8 and depicts the behavior of the system in the presence of locks. Observe that significant gains from splitting are available most of the time despite this pessimism. In several cases, the improvement is on a similar order of magnitude to the gains without locking. However, with long locks, the gains are much smaller. This phenomenon occurs because when the critical sections are long, the extra blocking time from multiple subjobs can quickly overutilize the system. These results demonstrate that when critical sections are not too long, job splitting is likely to be a useful technique even in the presence of locks.

5.9 Conclusion

Lateness bounds established previously for GEL schedulers can be lowered in theory by splitting jobs. However, such splitting can increase overheads and create problems for locking protocols. In this chapter, we showed how to incorporate splitting-related costs into overhead analysis and how to address locking-related concerns. We then applied these results in a schedulablity study in which real measured overheads were considered. This study suggests that job splitting can viably lower lateness bounds in practice.



(a) Medium Bimodal Utilization, Long Uniform Periods, Short Critical Sections, $n_r = 6$, $p_{acc} = 0.25$.

(b) Medium Bimodal Utilization, Long Uniform Periods, Medium Critical Sections, $n_r = 6$, $p_{acc} = 0.25$.

(c) Medium Bimodal Utilization, Medium Uniform Periods, Long Critical Sections, $n_r = 6$, $p_{acc} = 0.25$.

Figure 5.10: Results with respect to WSS in systems with locking.

CHAPTER 6: DISSIPATION BOUNDS

In this chapter, we provide response-time bounds for tasks at level C in MC^2, which was described in Section 1.2. Recall that "restricted supply," indicating that some processors are not fully available to tasks at level C, can be handled using the work of Leontyev and Anderson (2010), as discussed in Section 2.1. MC^2 statically prioritizes tasks at levels A and B above those at level C, so execution at levels A and B can be considered as restricted supply when analyzing level C. We continue to use this general strategy, but reduce pessimism. In addition, our analysis is sufficiently general to account for level-A, -B, and -C jobs that overrun their level-C PWCETs.

When any job at or above level C overruns its level-C PWCET, the system at level C may be *overloaded*. As noted above, this can compromise level-C guarantees. Using the normal MC^2 framework, a task may have its per-job response times permanently increased as a result of even a single overload event, and multiple overload events could cause such increases to build up over time. For example, if a system is fully utilized, then there is no "slack" with which to recover from overload. Therefore, we must alter scheduling decisions to attempt to recover from transient overload conditions. We do so by scaling task inter-release times and modifying scheduling priorities. We also provide *dissipation bounds* on the time required for response-time bounds to settle back to normal.

The remainder of this chapter is organized as follows. In Section 6.1, we discuss the general task and supply model used in this chapter. In Section 6.2, we provide our response-time bounds with full generality. Then, in Section 6.3, we provide dissipation bounds. In Section 6.4, we discuss the implementation of a system with recovery, and in Section 6.5 we provide experimental results examining dissipation times on an actual implementation. Finally, in Section 6.6, we conclude.

6.1 System Model

In this chapter, we consider a generalized version of GEL scheduling, called *GEL with virtual time* (*GEL-v*) scheduling, and a generalized version of the sporadic task model, called the *sporadic with virtual time and overload* (*SVO*) *model*. We assume that time is continuous.

In our analysis, we consider only the system at level C. In other words, we model level-A and -B tasks as supply that is unavailable to level C, rather than as explicit tasks. Thus, we consider a system $\tau = \{\tau_0, \tau_1, \dots, \tau_{n-1}\}$ of n *level-C* tasks running on $m > 1$ processors $P = \{P_0, P_1, \dots, P_{m-1}\}$. We assume that $\min_{\tau_i \in \tau} r_{i,0} = 0$. Unlike in Chapters 3–5, we explicitly assume that jobs are *not* allowed to begin execution before their release times. While this restriction could likely be lifted, doing so increases proof complexity and is left as future work. The time when an arbitrary $\tau_{i,k}$ actually completes is denoted $t^c_{i,k}$. We define the following quantities that pertain to the execution time of $\tau_{i,k}$.

Definition 6.1. $e_{i,k}$ is the actual execution time of $\tau_{i,k}$.

Definition 6.2. $e^c_{i,k}(t)$ (*c*ompleted) is the amount of execution that $\tau_{i,k}$ completes before time t.

Definition 6.3. $e^r_{i,k}(t)$ (*r*emaining) is the amount of execution that $\tau_{i,k}$ compeletes after time t.

These quantities are related by the following property, which follows immediately from Definitions 6.1–6.3.

Property 6.1. *For arbitrary $\tau_{i,k}$ and time t, $e^c_{i,k}(t) + e^r_{i,k}(t) = e_{i,k}$.*

We also define what it means for $\tau_{i,k}$ to be "pending".

Definition 6.4. $\tau_{i,k}$ is defined to be *pending* at time t if $r_{i,k} \le t \le t^c_{i,k}$.

Both our scheduling scheme and our task model are modified compared to that discussed in Section 1.1.1. Under GEL scheduling and the conventional sporadic task model discussed in Section 1.1.1, each task is characterized by a per-job worst-case execution time (WCET) $C_i > 0$, a minimum separation $T_i > 0$ between releases, and a relative PP $Y_i \ge 0$. Using the above notation, the system is subject to the following constraints for every $\tau_{i,k}$:

$$e_{i,k} \le C_i, \tag{6.1}$$

$$r_{i,k+1} \ge r_{i,k} + T_i, \tag{6.2}$$

$$y_{i,k} = r_{i,k} + Y_i. \tag{6.3}$$

Under the SVO model, we no longer assume a particular WCET (thus allowing overload from level-C tasks).[1] Therefore, (6.1) is no longer required to hold.

[1] MC^2 optionally supports the use of execution budgets, which would prevent these particular overloads. However, our analysis is general enough to apply even when execution budgets are *not* used.

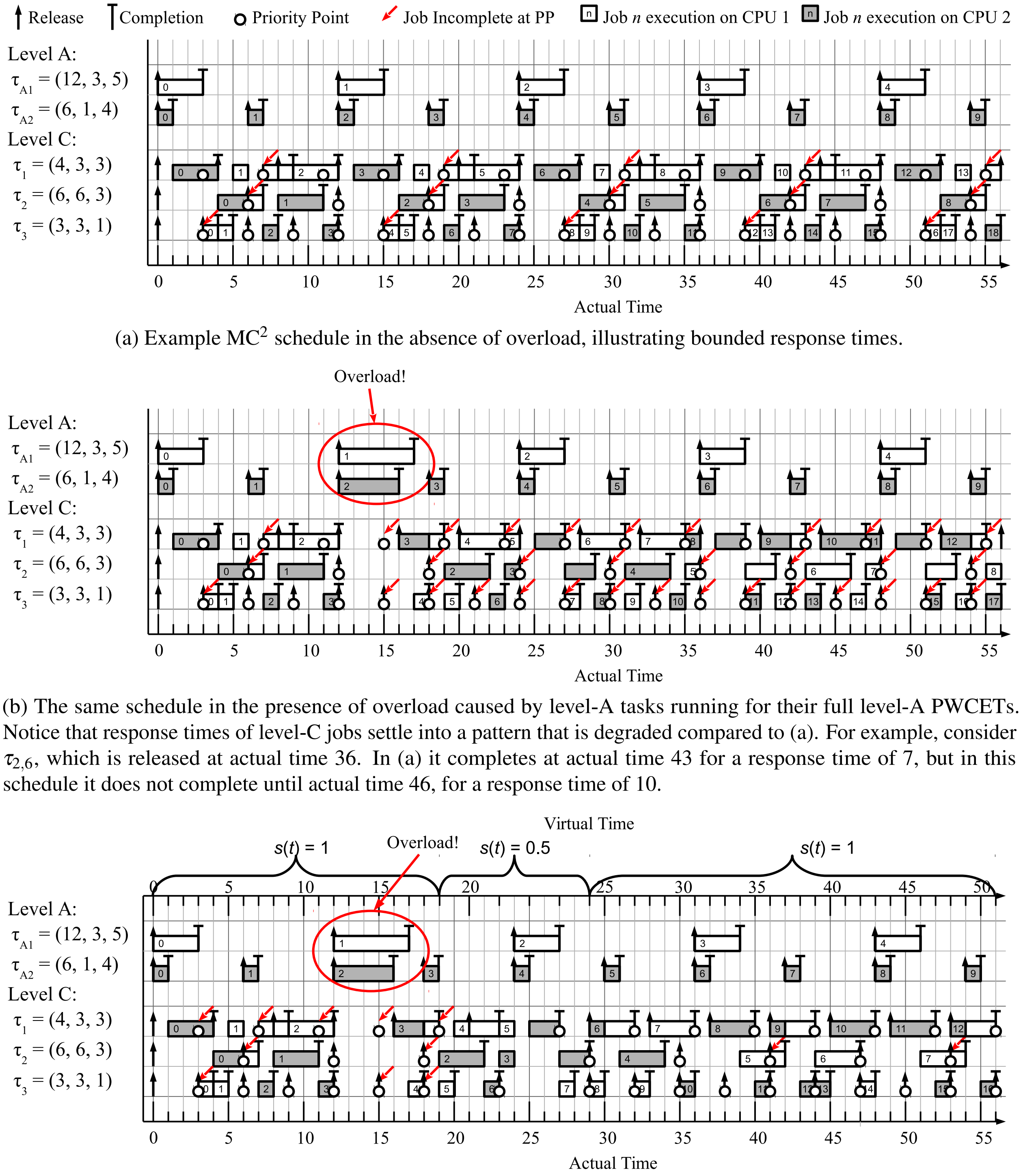
Under GEL-v scheduling and the SVO model, we use a notion of *virtual time* (as in (Stoica et al., 1996)), and we define the minimum separation time and relative PP of a task with respect to virtual time after one of its job releases instead of actual time. The purpose of virtual time is depicted in Figure 6.1, which we now describe.

In Figure 6.1, we depict a system that only has level-A and -C tasks, with one level-A task per CPU. For level-A tasks, we use the notation (T_i, C_i^C, C_i^A), where T_i is task τ_i's period, C_i^C is its level-C PWCET, and C_i^A is its level-A PWCET. For level-C tasks, we use the notation (T_i, Y_i, C_i), where all parameters are defined below. Using the analysis provided in this chapter, response times for all jobs can be shown to be bounded in the absence of overload. However, even in the absence of overload, as depicted in Figure 6.1(a), some jobs complete shortly after their PPs or after successor jobs are released.

The particular example in Figure 6.1 fully utilizes all processors. In the situation depicted in Figure 6.1(b), both level-A tasks released at time 12 run for their full level-A PWCETs. Therefore, from the perspective of level C, an overload occurs.[2] Because the system is fully utilized, there is no "slack" that allows for recovery from overload, and response times are permanently increased. In a system with large utilization, response times could take significant time to settle back to normal, even if they eventually will.

Once an overload occurs, the system can respond by altering virtual time for level C, as depicted in Figure 6.1(c). Virtual time is based on a global speed function $s(t)$. During normal operation of the system, $s(t)$ is always 1. This means that actual time and virtual time progress at the same rate. However, after an overload occurs, the scheduler may choose to select $0 < s(t) < 1$, at which point virtual time progresses more slowly than actual time. In Figure 6.1(c), the system chooses to use $s(t) = 0.5$ for $t \in [19, 29)$. As a result, virtual time progresses more slowly in this interval, and new releases of jobs are delayed. This allows the system to recover from the overload, so at actual time 29, $s(t)$ returns to 1. Observe that job response times are significantly increased after actual time 12 when the overload occurs, but after actual time 29, they are similar to before the overload. In fact, the depicted arrival pattern of level A happens to result in better response times after recovery than before the overload, although this is not guaranteed under a sporadic release pattern.

[2] A similar overload could occur if a level-C task exceeds its level-C PWCET. However, as discussed in Footnote 1, MC^2 optionally supports the use of execution budgets in order to prevent such an occurrence. While the use of execution budgets would prevent level-A and -B tasks from overrunning their level-A and -B PWCETs, respectively, they can still overrun their level-C PWCETs. Thus, we have chosen examples that provide overload even when execution budgets are used.



(a) Example MC^2 schedule in the absence of overload, illustrating bounded response times.

(b) The same schedule in the presence of overload caused by level-A tasks running for their full level-A PWCETs. Notice that response times of level-C jobs settle into a pattern that is degraded compared to (a). For example, consider $\tau_{2,6}$, which is released at actual time 36. In (a) it completes at actual time 43 for a response time of 7, but in this schedule it does not complete until actual time 46, for a response time of 10.

(c) The same schedule in the presence of overload and the recovery techniques described in this section. Notice that response times of level-C jobs settle into a pattern that is more like (a) than (b). For example, consider again $\tau_{2,6}$, which now is not released until actual time 41 and completes at actual time 47 for a response time of 6 ms. This is more similar to (a) than to (b).

Figure 6.1: Example MC^2 task system, without and with overload.

An actual time t is converted to a virtual time using

$$v(t) \triangleq \int_0^t s(t)\,dt. \tag{6.4}$$

For example, in Figure 6.1(c), $v(25) = \int_0^{25} s(t)\,dt = \int_0^{19} 1\,dt + \int_{19}^{25} 0.5\,dt = 19 + 3 = 22$. This definition leads to the following property:

Property 6.2. *For arbitrary time instants t_0 and t_1,*

$$v(t_1) - v(t_0) = \int_{t_0}^{t_1} s(t)\,dt.$$

Unless otherwise noted, all instants (e.g., t, $r_{i,k}$, etc.) are specified in actual time, and all variables *except* T_i, Y_i, and U_i^v (all defined below) refer to quantities of actual time.

Under the SVO model, (6.2) generalizes to

$$v(r_{i,k+1}) \geq v(r_{i,k}) + T_i, \tag{6.5}$$

and under GEL-v scheduling, (6.3) generalizes to

$$v(y_{i,k}) = v(r_{i,k}) + Y_i. \tag{6.6}$$

For example, in Figure 6.1(c), $\tau_{1,0}$ is released at actual time 0, has its PP three units of (both actual and virtual) time later at actual time 3, and $\tau_{1,1}$ can be released four units of (both actual and virtual) time later at time 4. However, $\tau_{1,5}$ of the same task is released at actual time 21, shortly after the virtual clock slows down. Therefore, its PP is at actual time 27, which is three units of *virtual* time after its release, and the release of $\tau_{1,6}$ can be no sooner than actual time 29, which is four units of *virtual* time after the release of $\tau_{1,5}$. However, the execution time of $\tau_{1,5}$ is not affected by the slower virtual clock.

In light of (6.5), we denote as $b_{i,k}$ (*b*oundary) the earliest actual time that $\tau_{i,k+1}$ could be released, based on the release time of $\tau_{i,k}$. $b_{i,k}$ is indexed using k because it depends on the actual release time of $\tau_{i,k}$, not the actual release time of $\tau_{i,k+1}$.

Definition 6.5. $b_{i,k}$ is the actual time such that $v(b_{i,k}) = v(r_{i,k}) + T_i$.

Although it is possible to analyze systems where some $Y_i > T_i$, doing so increases proof complexity without providing any benefit to response-time or dissipation bounds. Therefore, we assume that for all i,

$$Y_i \leq T_i. \tag{6.7}$$

In our analysis, we will frequently refer to the total work that a task produces from jobs that have both releases and PPs in a certain interval. We therefore define a function for this quantity.

Definition 6.6.

$$D_i^e(t_0, t_1) \triangleq \sum_{\tau_{i,k} \in \omega} e_{i,k}$$

(*D*emand), where ω is the set of jobs with $t_0 \leq r_{i,k} \leq y_{i,k} \leq t_1$.

In order to model processor supply, we use a "service function" as in (Chakraborty et al., 2003; Leontyev and Anderson, 2010), and as reviewed in Section 2.1.

Definition 6.7. $\beta_p(t_0, t_1)$ is the total number of units of time during which P_p is available to level C within $[t_0, t_1)$.

We further characterize processor supply in two parts. First, we assign to each processor P_p a *nominal utilization* $\widehat{u_p}$, representing how much of its time we expect to be available to level C in the long term. Within an arbitrary $[t_0, t_1)$, we expect $\beta_p(t_0, t_1) \approx \widehat{u_p}(t_1 - t_0)$. For example, in Figure 6.1(c), P_0 is available whenever τ_{A1} is not running, and the utilization of τ_{A1} at level C is $\frac{3}{12}$, so we choose $\widehat{u_0} = 1 - \frac{3}{12} = \frac{3}{4}$. Similarly, P_1 is available whenever τ_{A2} is not running, so we choose $\widehat{u_1} = 1 - \frac{1}{6} = \frac{5}{6}$.

Over some intervals $[t_0, t_1)$, a CPU is available for less time than indicated by nominal utilization alone. For example, in Figure 6.1(c) over $[0, 3)$, P_0 is not available at all to level C. Thus, for our second characterization, we define a *supply restriction overload* function,

$$o_p(t_0, t_1) \triangleq \max\{0, \widehat{u_p} \cdot (t_1 - t_0) - \beta_p(t_0, t_1)\}. \tag{6.8}$$

This implies that

$$\beta_p(t_0, t_1) \geq \widehat{u_p} \cdot (t_1 - t_0) - o_p(t_0, t_1). \tag{6.9}$$

For example, consider $[0, 3)$ in Figure 6.1(c). By naïvely using nominal utilization, we would expect level C to receive $\widehat{u_0} \cdot (t_1 - t_0) = \frac{3}{4} \cdot (3 - 0) = \frac{9}{4}$ units of service on P_0, but it actually receives 0, so $o_0(t_0, t_1) = \frac{9}{4}$. In the absence of overload, there must be some constant σ_p such that, for all intervals $[t_0, t_1)$, $o_p(t_0, t_1) \leq \widehat{u_p}\sigma_p$, and our model reduces exactly to that used by Leontyev and Anderson (2010) under this assumption. However, our more general model can be used to account for arbitrary overloads, by allowing $o_p(t_0, t_1) > \sigma_p$ when overload occurs within $[t_0, t_1)$. We also define

$$u_{tot} \triangleq \sum_{P_p \in P} \widehat{u_p}, \tag{6.10}$$

which (when supply restriction overload is bounded) represents the total processing capacity available to the system at level C.

We assume that the eventual supply available to the system is infinite. This assumption leads to the following property.

Property 6.3. *Any particular $\tau_{i,k}$ eventually completes.*

If the eventual supply is finite, then no overload recovery is possible. Similarly, we assume that $u_{tot} > 0$, for otherwise, the system cannot be guaranteed to schedule *any* system *even in the absence of unexpected overload*.

6.2 Response-Time Analysis

In this section, we provide a *general* method for analyzing response times of a system at level C, under GEL-v scheduling and with most of the generality of the SVO model. Because we make few assumptions about overload, this method does not provide response-time bounds that apply to all jobs. In fact, it applies even in situations where such bounds *do not exist*. However, we will use these results, with additional assumptions, in Section 6.3 to provide dissipation bounds and long-term response-time bounds in the absence of overload.

Under GEL scheduling applied to ordinary sporadic task systems, we proved in Chapter 3 that $t^c_{i,k} \leq y_{i,k} + x_i + C_i$, where x_i is a per-task constant. That proof works by analyzing the behavior of each $\tau_{i,k}$ after $y_{i,k}$, because no job with higher priority can be released after $y_{i,k}$. In the presence of overload a single per-task x_i may not exist. Furthermore, even in cases where such an x_i does exist, it must pessimistically bound all job releases, preventing any analysis of dissipation bounds. Therefore, we instead define a function of time $x_i(t) \geq 0$ so that $t^c_{i,k} \leq y_{i,k} + x_i(y_{i,k}) + e_{i,k}$. (We use $e_{i,k}$ in place of C_i because our analysis no longer assumes that $e_{i,k} \leq C_i$ holds.) In our analysis, it is convenient to define $x_i(t)$ over all nonnegative real numbers. Furthermore, it will be convenient to treat $x_i(t)$ as merely a safe upper bound. Therefore, we use the following definition.

Definition 6.8. $x_i(t)$ is x-sufficient if $x_i(t) \geq 0$, and for all $\tau_{i,k}$ with $y_{i,k} \leq t$,

$$t^c_{i,k} \leq t + x_i(t) + e_{i,k}.$$

Throughout our analysis both here and in Section 6.3, we will frequently use the following property, which follows immediately from Definition 6.8.

Property 6.4. *If $c_1 \geq c_0$ and $x_i(t_a) = c_0$ is x-sufficient, then $x_i(t_a) = c_1$ is x-sufficient.*

In the remainder of this section, we will provide an x-sufficient value for $x_i(t_a)$ for each τ_i and each time t_a (under *a*nalysis). We will exhaustively consider the cases depicted in Figure 6.2 for each t_a, in approximate order of increasing complexity. Note that Cases D and E reference terminology that will be defined later in this section.

We first consider Case A, which provides the value of $x_i(t_a)$ when $t_a < y_{i,0}$. This case is simple.

Theorem 6.1. *If $t_a < y_{i,0}$, then $x_i(t_a) = 0$ is x-sufficient.*

Proof. This theorem results from the definition of x-sufficient in Definition 6.8. If $t_a < y_{i,0}$, then the condition in Definition 6.8 holds vacuously, because there are no jobs $\tau_{i,k}$ with $y_{i,k} \leq t_a$. □

We now consider Case B, in which $t_a = y_{i,k}$ for some k and $t^c_{i,k} \leq y_{i,k} + e_{i,k}$. This case is similarly simple, and we analyze it separately from the other cases with $t_a = y_{i,k}$ in order to simplify later proofs.

Theorem 6.2. *If $t_a = y_{i,k}$ for some k and $t^c_{i,k} \leq y_{i,k} + e_{i,k}$, then $x_i(t_a) = 0$ is x-sufficient.*

Proof. This theorem follows immediately from the definition of x-sufficient in Definition 6.8. □

We next consider Case C, in which t_a lies between two consecutive PPs. In this case, our bound depends on having an x-sufficient value at the last PP before t_a of a job in τ_i. This can be computed using Case B, D, or E.

Theorem 6.3. *If $t_a \in (y_{i,k}, y_{i,k+1})$ and $x_i(y_{i,k})$ is x-sufficient, then $x_i(t_a) = \max\{0, x_i(y_{i,k}) - (t_a - y_{i,k})\}$ is x-sufficient.*

A. $t_a < y_{i,0}$ (Theorem 6.1).

B. $t_a = y_{i,k}$ for some k and $t^c_{i,k} \leq y_{i,k} + e_{i,k}$ (Theorem 6.2).

C. $t_a \in (y_{i,k}, y_{i,k+1})$ for some k (Theorem 6.3).

D. $t_a = y_{i,k}$ for some k, $t^c_{i,k} > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is f-dominant for L (Theorem 6.4).

E. $t_a = y_{i,k}$ for some k, $t^c_{i,k} > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is m-dominant for L (Theorem 6.5).

Figure 6.2: Cases for which values of $x_i(t_a)$ are provided.

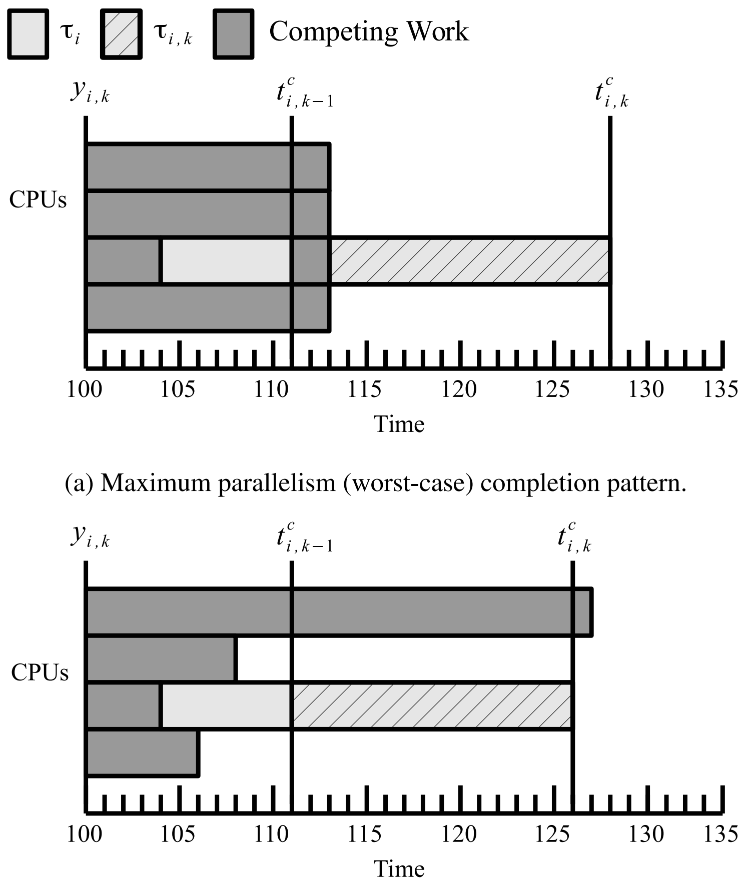
Proof. This theorem results from the definition of x-sufficient in Definition 6.8. If $t_a \in (y_{i,k}, y_{i,k+1})$, then there are no jobs of τ_i with PPs in $(y_{i,k}, t_a)$, so $t^c_{i,k}$ is the latest completion of any job of τ_i with a PP before t_a. We have

$$\begin{aligned} t^c_{i,k} &\leq \{\text{By the definition of } x\text{-sufficient in Definition 6.8}\} \\ &\quad y_{i,k} + x_i(y_{i,k}) + e_{i,k} \\ &= \{\text{Rearranging}\} \\ &\quad t_a + x_i(y_{i,k}) - (t_a - y_{i,k}) + e_{i,k}. \end{aligned}$$

Therefore, combined with the requirement from Definition 6.8 that $x_i(t_a) \geq 0$, $x_i(t_a) = \max\{0,\ x_i(y_{i,k}) - (t_a - y_{i,k})\}$ is x-sufficient. □

We will next consider Cases D and E. In both of these cases, $t_a = y_{i,k}$ for some k. Before providing proofs, we will first provide a basic explanation for why the presence of supply restriction adds complexity to response-time analysis, and how we account for such complexity. This discussion motivates the structure of our proofs, and also motivates the separate consideration of Case D and Case E.

After $y_{i,k}$, $\tau_{i,k}$ can be delayed for two reasons: all processors can be occupied by either other work and/or supply restriction, or some predecessor job of $\tau_{i,k}$ within τ_i can be incomplete. We define work from $\tau_{j,\ell}$ as *competing* with $\tau_{i,k}$ if $y_{j,\ell} \leq y_{i,k}$ and $j \neq i$, and supply restriction as *competing* if it occurs before $t^c_{i,k}$. Note that, in order to account for work or supply restriction that indirectly affects $\tau_{i,k}$ by pushing other work forward in time, we do not require that work or supply restriction occur after $r_{i,k}$ in order to say that it is "competing" with $\tau_{i,k}$.



(a) Maximum parallelism (worst-case) completion pattern.

(b) Alternative completion pattern.

Figure 6.3: Example completion patterns for competing work in the absence of supply restriction.

We first review the basic structure of previous analysis from Chapter 3 in the *absence* of supply restriction. Such analysis considers competing work remaining at $y_{i,k}$. Some example patterns for the completion of competing work are depicted in Figure 6.3. Figure 6.3(a) depicts the worst-case delay *due to competing work rather than a predecessor*, when all processors are occupied until $\tau_{i,k}$ can begin execution. Figure 6.3(b) depicts an alternative completion pattern for the same amount of work. Observe that before $t^c_{i,k-1}$, there are idle CPUs. Thus, this example depicts the situation where $\tau_{i,k}$ is delayed due to its predecessor.

If some processor is idle, then there must be fewer than m tasks with remaining work. Thus, in the absence of supply restriction, τ_i will run continuously until $\tau_{i,k}$ completes. This is why the worst-case completion pattern is the one with maximum parallelism, as depicted in Figure 6.3(a). For fixed $t^c_{i,k-1}$, any other completion pattern might allow $\tau_{i,k}$ to complete earlier (as happens in Figure 6.3(b)) or else does not change the completion time of $\tau_{i,k}$ (if the delay due to an incomplete predecessor already dominated). To summarize, in the absence of overload, either the delay due to an incomplete predecessor dominates, as in Figure 6.3(b), or the delay due to competing work dominates, as in Figure 6.3(a).

We now consider the effects of introducing supply restriction. Figure 6.4 depicts similar completion patterns as Figure 6.3. As before, $\tau_{i,k}$ can be delayed either because all processors are occupied by competing work or supply restriction, or because some predecessor of $\tau_{i,k}$ within τ_i is incomplete. However, as can be seen by comparing Figure 6.4(a) and Figure 6.4(b), having all competing work complete with maximum parallelism is no longer the worst case. This phenomenon occurs because supply restriction can now prevent the execution of τ_i even after some processor has become idle, by reducing the number of available processors below the number of tasks with remaining work. This increases the complexity of determining the interaction between delays caused by competing work and delays caused by an incomplete predecessor, as the simple dominance that occurred in the absence of supply restriction may not occur.

To determine an upper bound on $t^c_{i,k}$, we add to $y_{i,k}$ the sum of the lengths of three types of sub-intervals within $[y_{i,k}, t^c_{i,k})$, as depicted in Figure 6.4(b).

1. Sub-intervals during which τ_i does not run because all m processors are occupied by competing work or supply restriction.

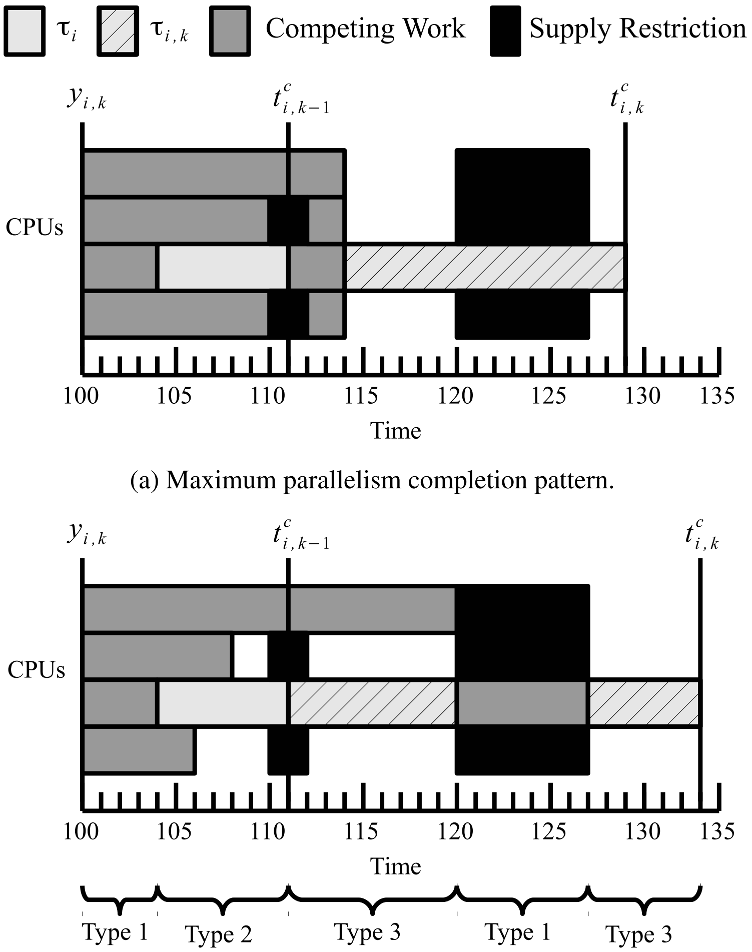
2. Sub-intervals during which jobs of τ_i before $\tau_{i,k}$ execute.

3. Sub-intervals during which $\tau_{i,k}$ executes.

We will bound the total length of sub-intervals of Type 1 by bounding the total amount of competing work and supply restriction. We will now define the total length of sub-intervals of Type 2 as $e^p_{i,k}$; the total length of sub-intervals of Type 3 is simply $e^r_{i,k}(y_{i,k})$.

Definition 6.9. $e^p_{i,k}$ is the work remaining after $y_{i,k}$ due to jobs of τ_i prior to $\tau_{i,k}$.

Let c denote a bound on the total amount of competing work after $y_{i,k}$ and competing supply restriction after $y_{i,k}$. The specific value of c will be derived in Lemma 6.6 below, but its exact expression is not relevant for the purposes of this introductory discussion. The total length of Type 1 sub-intervals of $[y_{i,k}, t^c_{i,k})$, where τ_i is not running, can be upper bounded by dividing c by m. However, this bound may be unnecessarily pessimistic, because some of the competing work and supply restriction may actually run concurrently with τ_i. For example, in Figure 6.4(b), some competing work and supply restriction runs within $[104, 120)$ even though this interval is composed of a sub-interval of Type 2 and a sub-interval of Type 3.



(a) Maximum parallelism completion pattern.

(b) Alternative completion pattern.

Figure 6.4: Example completion patterns for competing work in the presence of supply restriction.

We now informally describe an optimization that allows us to reduce some of this pessimism. We will simplify our informal analysis by assuming that $k > 0$ and $t^c_{i,k-1} > y_{i,k}$. We will later discuss how to relax this assumption. Let v be an arbitrary integer with $0 \le v < m$. We consider two cases.

Few Tasks Case. If there is some time within $[y_{i,k}, t^c_{i,k-1})$ such that at most v processors are occupied by work or supply restriction, then there are at most v tasks that have work remaining, or more CPUs would be occupied. Thus, in this case there are at most v tasks with competing work remaining after $t^c_{i,k-1}$, and $\tau_{i,k}$ can execute after $t^c_{i,k-1}$ whenever there are at least v processors available to level C. For example, in Figure 6.4(b), if $v = 2$, then because only v processors are occupied at time $109 \in [y_{i,k}, t^c_{i,k-1})$, there are only v tasks with remaining work at this time, and $\tau_{i,k}$ can run after $t^c_{i,k-1}$ whenever at least $v = 2$ processors are available to level C. Therefore, rather than summing the lengths of the intervals of each type, we can compute an upper bound on the time it takes for there to be $e_{i,k}$ time units with at least v processors available after $t^c_{i,k-1}$.

Many Tasks Case. If there are at least $v+1$ tasks with work remaining throughout $[y_{i,k}, t^c_{i,k-1})$, then at least v processors are occupied with competing work and/or supply restriction in all sub-intervals of Type 2 (in which jobs of τ_i prior to $\tau_{i,k}$ are running). For example, in Figure 6.4(b), this case holds with $v = 1$, because there is some other task executing on the first processor until $t^c_{i,k-1}$. By the definition of $e^p_{i,k}$ in Definition 6.9, the total length of Type 2 intervals is $e^p_{i,k}$. Thus, at least $v \cdot e^p_{i,k}$ units of competing work and supply restriction actually run in intervals of Type 2, so we can subtract $v \cdot e^p_{i,k}$ from the bound c to upper bound the amount of work and supply restriction running in Type 1 intervals.

As mentioned above, this informal analysis assumes that $\tau_{i,k-1}$ exists and that $t^c_{i,k-1} > y_{i,k}$. If this is not the case, then by the definition of $e^p_{i,k}$ in Definition 6.9, $e^p_{i,k} = 0$. Thus, the analysis from the Many Tasks Case will subtract $v \cdot e^p_{i,k} = 0$ units of competing work and supply restriction from the bound c for Type 1 intervals. Therefore, it is safe to use the analysis from the Many Tasks Case when $k = 0$ or $t^c_{i,k-1} \leq y_{i,k}$. Observe that for *any* v with $0 \leq v < m$, one of these two cases must hold. We therefore define a pair of properties, one for each case.

Definition 6.10. Let L be an arbitrary integer with $0 \leq L < m$. If $k > 0$, $t^c_{i,k-1} > y_{i,k}$, and there are at most $m - L - 1$ tasks that have work remaining at $t^c_{i,k-1}$, then $\tau_{i,k}$ is *f-dominant for L* (*f*ew tasks case).

$\tau_{i,k}$ is f-dominant for L if the Few Tasks Case applies.

Definition 6.11. Let L be an arbitrary integer with $0 \leq L < m$. If $\tau_{i,k}$ is not f-dominant for L, then $\tau_{i,k}$ is *m-dominant for L* (*m*any tasks case).

$\tau_{i,k}$ is m-dominant for L if the Many Tasks Case applies.

In Section 6.2.1 below, we will consider Case D, in which $\tau_{i,k}$ is f-dominant for L. Then, in Section 6.2.2, we will consider Case E, in which $\tau_{i,k}$ is m-dominant for L.

6.2.1 Case D: $t_a = y_{i,k}$ for some k and $\tau_{i,k}$ is f-dominant for L.

In this case, we use the Few Tasks Case with $v = m - L - 1$. Recall from the above discussion that in this case, $\tau_{i,k}$ runs after $t^c_{i,k-1}$ whenever there are at least v processors available to level C. Lemma 6.2 below provides a bound on $t^c_{i,k}$ in this case. Lemma 6.1 is used to prove Lemma 6.2.

Lemma 6.1. *For any integer* $0 \leq v \leq m$*, in any time interval* $[t_0, t_1)$ *with* $t_1 \geq t_0$*, there are at least*

$$(t_1 - t_0) - \sum_{P_p \in \zeta} ((1 - \widehat{u_p}) \cdot (t_1 - t_0) + o_p(t_0, t_1))$$

units of time during which at least v processors are available to level C, where ζ *is the set of v processors that minimizes the sum.*

Proof. We prove this lemma by induction. Without loss of generality, we fix t_0 and t_1 and assume that P is ordered by increasing $(1 - \widehat{u_p})(t_1 - t_0) + o_p(t_0, t_1)$. We prove the stronger condition that within $[t_0, t_1)$, there are $(t_1 - t_0) - \sum_{p=0}^{v-1}((1 - \widehat{u_p})(t_1 - t_0) + o_p(t_0, t_1))$ units of time during which the specific processors P_0 through P_{v-1} are available to level C.

As the base case, we consider $v = 0$. During any time interval, it is vacuously true that all processors in the empty set are available to level C, so there are $t_1 - t_0$ such units of time in $[t_0, t_1)$ and the lemma holds.

For the inductive case, assume that there are

$$(t_1 - t_0) - \sum_{p=0}^{v-1} ((1 - \widehat{u_p}) \cdot (t_1 - t_0) + o_p(t_0, t_1)) \tag{6.11}$$

units of time in $[t_0, t_1)$ during which processors P_0 through P_{v-1} are available to level C.

By the definition of $\beta_v(t_0, t_1)$ in Definition 6.7, P_v is unavailable to level C in $[t_0, t_1)$ for

$$\begin{aligned}
&(t_1 - t_0) - \beta_v(t_0, t_1) \\
\leq\; &\{\text{By (6.9)}\} \\
&(t_1 - t_0) - (\widehat{u_v} \cdot (t_1 - t_0) - o_v(t_0, t_1)) \\
=\; &\{\text{Rearranging}\} \\
&(1 - \widehat{u_v}) \cdot (t_1 - t_0) + o_v(t_0, t_1)
\end{aligned} \tag{6.12}$$

units of time.

In the worst case, as depicted in Figure 6.5, either all of P_v's unavailable time occurs when processors P_0 through P_{v-1} are all available, or P_v is unavailable during all times when P_0 through P_{v-1} are available. In either case, the lemma holds by subtracting (6.12) from (6.11). □

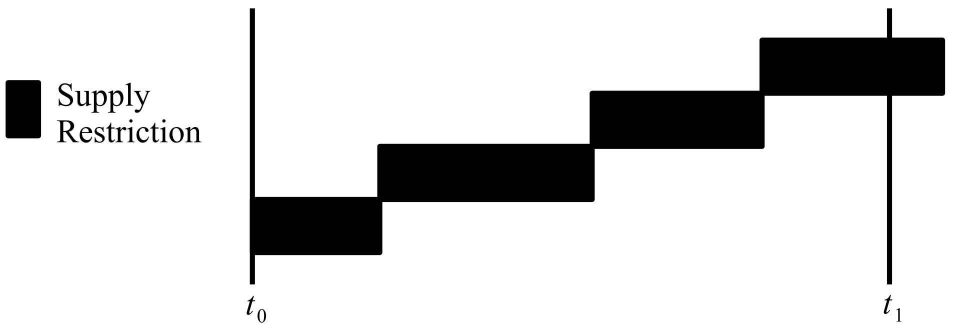


Figure 6.5: Worst-case pattern of supply restriction for the first four processors, in order to minimize the amount of time that all four processors are available. In this case, the four processors are never all available at the same time within $[t_0, t_1)$.

We now use Lemma 6.1 to prove the next lemma, which bounds $t^c_{i,k}$ in the Few Tasks Case. We let v be an integer with $0 \le v < m$, and we let

$$A_{i,k}(v) \triangleq \begin{cases} \frac{e_{i,k}+\sum_{P_p\in\Theta} o_p(t^c_{i,k-1},t^c_{i,k})}{1-v+\sum_{P_p\in\Theta}\widehat{u_p}} & \text{if } 1-v+\sum_{P_p\in\Theta}\widehat{u_p} > 0, \\ \infty & \text{otherwise,} \end{cases} \tag{6.13}$$

where Θ is the set of v processors that minimizes $A_{i,k}(v)$.

Lemma 6.2. *$A_{i,k}(v) > 0$ holds. Moreover, if $\tau_{i,k}$ can run after $t^c_{i,k-1}$ whenever there are at least v processors available to level C, then $t^c_{i,k} \le t^c_{i,k-1} + A_{i,k}(v)$.*

Proof. If $A_{i,k}(v)$ is infinite, then the lemma must hold by Property 6.3. Thus, we assume that $A_{i,k}(v)$ is finite.

To see that $A_{i,k}(v) > 0$, observe that because $A_{i,k}(v)$ is finite, by (6.13), $1-v+\sum_{P_p\in\Theta}\widehat{u_p} > 0$. Furthermore, by Definition 6.1, $e_{i,k} > 0$, and by (6.8), $o_p(t^c_{i,k-1},t^c_{i,k}) \ge 0$. Therefore, by (6.13), $A_{i,k}(v) > 0$.

To prove the second part of the lemma, we use proof by contradiction. Suppose $t^c_{i,k} > t^c_{i,k-1} + A_{i,k}(v)$. By Lemma 6.1, the number of time units in $[t^c_{i,k-1}, t^c_{i,k})$ with v processors available to level C is at least

$$\begin{aligned} &(t^c_{i,k} - t^c_{i,k-1}) - \sum_{P_p\in\zeta}\left((1-\widehat{u_p})(t^c_{i,k} - t^c_{i,k-1}) + o_p(t^c_{i,k-1}, t^c_{i,k})\right) \\ &\ge \{\text{By the definition of } \zeta \text{ in Lemma 6.1, because } \Theta \text{ has } v \text{ processors}\} \\ &(t^c_{i,k} - t^c_{i,k-1}) - \sum_{P_p\in\Theta}\left((1-\widehat{u_p})(t^c_{i,k} - t^c_{i,k-1}) + o_p(t^c_{i,k-1}, t^c_{i,k})\right) \end{aligned}$$

$$
\begin{aligned}
&= \{\text{Rearranging, and because } \Theta \text{ has } v \text{ processors}\} \\
&\quad \left(1 - v + \sum_{P_p \in \Theta} \widehat{u_p}\right)(t^c_{i,k} - t^c_{i,k-1}) - \sum_{P_p \in \Theta} o_p(t^c_{i,k-1}, t^c_{i,k}) \\
&> \{\text{Because } t^c_{i,k} > t^c_{i,k-1} + A_{i,k}(v) \text{ and } 1 - v + \textstyle\sum_{P_p \in \Theta} \widehat{u_p} > 0\} \\
&\quad \left(1 - v + \sum_{P_p \in \Theta} \widehat{u_p}\right) A_{i,k}(v) - \sum_{P_p \in \Theta} o_p(t^c_{i,k-1}, t^c_{i,k}) \\
&= \{\text{By (6.13), because } A_{i,k}(v) \text{ is finite}\} \\
&\quad \left(1 - v + \sum_{P_p \in \Theta} \widehat{u_p}\right) \cdot \frac{e_{i,k} + \sum_{P_p \in \Theta} o_p(t^c_{i,k-1}, t^c_{i,k})}{1 - v + \sum_{P_p \in \Theta} \widehat{u_p}} - \sum_{P_p \in \Theta} o_p(t^c_{i,k-1}, t^c_{i,k}) \\
&= \{\text{Rearranging}\} \\
&\quad e_{i,k}. \qquad (6.14)
\end{aligned}
$$

However, because $\tau_{i,k}$ can run after $t^c_{i,k-1}$ whenever there are at least v processors available to level C, $\tau_{i,k}$ must have executed for longer than $e_{i,k}$ units. This is a contradiction. □

We now use this result to provide a bound on $x_i(t_a)$ to handle Case D.

Theorem 6.4. *If $t_a = y_{i,k}$ for some k, $t^c_{i,k} > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is f-dominant for L, then $x_i(t_a) = x^f_{i,k}$ is x-sufficient, where*

$$x^f_{i,k} \triangleq t^c_{i,k-1} - y_{i,k} + A_{i,k}(m - L - 1) - e_{i,k} \qquad (6.15)$$

(few tasks).

Proof. By the definition of f-dominant for L in Definition 6.10, because no new competing work is released after $y_{i,k}$, throughout $[t^c_{i,k-1}, t^c_{i,k})$, there are at most $m - L - 1$ tasks that have remaining competing work. Therefore, whenever at least $m - L - 1$ processors are available to level C within $[t^c_{i,k-1}, t^c_{i,k})$, $\tau_{i,k}$ is running. Thus,

$$
\begin{aligned}
t^c_{i,k} &\le \{\text{By Lemma 6.2}\} \\
&\quad t^c_{i,k-1} + A_{i,k}(m - L - 1) \\
&= \{\text{Rewriting}\} \\
&\quad y_{i,k} + (t^c_{i,k-1} - y_{i,k} + A_{i,k}(m - L - 1) - e_{i,k}) + e_{i,k}
\end{aligned}
$$

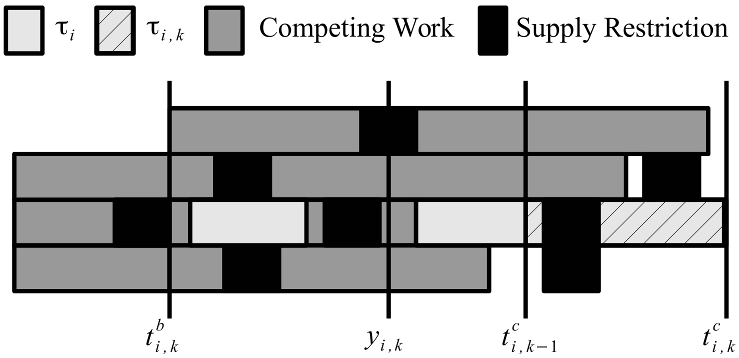


Figure 6.6: Example depicting $t^b_{i,k}$ when $m = 4$.

$$= \{\text{By the definition of } x^f_{i,k} \text{ in (6.15)}\}$$
$$y_{i,k} + x^f_{i,k} + e_{i,k}. \tag{6.16}$$

Because $t^c_{i,k} > y_{i,k} + e_{i,k}$ by the statement of the lemma, by (6.16), $x^f_{i,k} > 0$. Thus, by (6.16) and the definition of x-sufficient in Definition 6.8, $x_i(y_{i,k}) = x^f_{i,k}$ is x-sufficient. Because $t_a = y_{i,k}$, the theorem follows. □

6.2.2 Case E: $t_a = y_{i,k}$ for some k and $\tau_{i,k}$ is m-dominant for L

The basic structure of our analysis of Case E is fundamentally similar to the analysis in Chapter 3. We will analyze the lateness of an arbitrary job $\tau_{i,k}$, ignoring all jobs that have PPs after $y_{i,k}$. We define an interval as *busy* if, for the entire interval, all processors are either unavailable or are executing tasks with PPs not after $y_{i,k}$. As depicted in Figure 6.6, we define a time $t^b_{i,k}$.

Definition 6.12. $t^b_{i,k}$ the earliest time such that $t^b_{i,k} \leq y_{i,k}$ and $[t^b_{i,k}, y_{i,k})$ is busy.

Such a time must exist, because $[y_{i,k}, y_{i,k})$ (which is an empty interval) is vacuously busy. We separately upper bound work (Lemma 6.4) and competing supply restriction (Lemma 6.5) after $t^b_{i,k}$, and then use those results to determine a full response-time bound.

We will first upper bound all remaining work at $t^b_{i,k}$, including both competing work and work due to τ_i. In order to do so, we first prove a lemma that bounds the amount of work after arbitrary time $t_0 \leq y_{i,k}$ contributed by a task τ_j with a pending job at t_0. This will allow us to bound the work by tasks that have pending jobs at $t^b_{i,k}$. (We use t_0 instead of $t^b_{i,k}$ because the same lemma will also be used in Section 6.3 with a different choice of t_0.)

Lemma 6.3. *If there is a pending job of τ_j at* arbitrary *time $t_0 \leq y_{i,k}$, then denote as $\tau_{j,\ell}$ the earliest job of τ_j pending at t_0. The total remaining work at time t_0 for jobs of τ_j with PPs not later than $y_{i,k}$ is $e^r_{j,\ell}(t_0) + D^e_j(b_{j,\ell}, y_{i,k})$.*

Proof. By the definition of $b_{j,\ell}$ in Definition 6.5 and the definition of T_i in (6.5), any job of τ_j after $\tau_{j,\ell}$ must be released no sooner than $b_{j,\ell}$. Therefore, by Definition 6.6, the total work from such jobs with PPs not later than $y_{i,k}$ is $D^e_j(b_{j,\ell}, y_{i,k})$. Adding $e^r_{j,\ell}(t_0)$ for the remaining work due to $\tau_{j,\ell}$ yields the lemma. □

We now bound all remaining work at $t^b_{i,k}$, including that due to τ_i.

Lemma 6.4. *The remaining work at $t^b_{i,k}$ for jobs with PPs not later than $y_{i,k}$ is*

$$W_{i,k} \triangleq \sum_{\tau_{j,\ell} \in \theta_{i,k}} \left(e^r_{j,\ell}(t^b_{i,k}) + D^e_j(b_{j,\ell}, y_{i,k})\right) + \sum_{\tau_j \in \overline{\theta_{i,k}}} D^e_j(t^b_{i,k}, y_{i,k}). \tag{6.17}$$

where $\theta_{i,k}$ is the set of jobs $\tau_{j,\ell}$ such that $\tau_{j,\ell}$ is the earliest pending job of τ_j at $t^b_{i,k}$, $r_{j,\ell} < t^b_{i,k}$, and $y_{j,\ell} \leq y_{i,k}$, and $\overline{\theta_{i,k}}$ is the set of tasks that do not have jobs in $\theta_{i,k}$.

Proof. As before, we ignore any jobs that have PPs after $y_{i,k}$. We bound the work remaining for each task τ_j at $t^b_{i,k}$, depending on whether it has a pending job at $t^b_{i,k}$ with a release before $t^b_{i,k}$.

Case 1: τ_j has no Pending Job at $t^b_{i,k}$ with a Release Before $t^b_{i,k}$. In this case, all relevant work remaining for τ_j at $t^b_{i,k}$ comes from jobs $\tau_{j,\ell}$ with $t^b_{i,k} \leq r_{j,\ell} \leq y_{j,\ell} \leq y_{i,k}$. Thus, by the definition of $D^e_j(t^b_{i,k}, y_{i,k})$ in Definition 6.6, there are

$$D^e_j(t^b_{i,k}, y_{i,k}) \tag{6.18}$$

units of such work. Furthermore, such a task is in $\overline{\theta_{i,k}}$ by the definition of $\overline{\theta_{i,k}}$ in the statement of the lemma.

Case 2: τ_j has a Pending Job at $t^b_{i,k}$ with a Release Before $t^b_{i,k}$. Denote as $\tau_{j,\ell}$ the earliest pending job of τ_j at $t^b_{i,k}$. By Lemma 6.3 (with $t_0 = t^b_{i,k}$), the remaining work for τ_j at $t^b_{i,k}$ is

$$e^r_{j,\ell}(t^b_{i,k}) + D^e_j(b_{j,\ell}, y_{i,k}). \tag{6.19}$$

Furthermore, $\tau_{j,\ell}$ is in $\theta_{i,k}$ by the definition for $\theta_{i,k}$ in the statement of the lemma.

Summing over all tasks, using (6.18) or (6.19) as appropriate, yields $W_{i,k}$ by (6.17). □

We next consider supply restriction, accounting for it as if it were competing work. There is one significant difference between supply restriction and competing work. Under GEL-v scheduling, once a job has reached its PP, no new competing work can arrive. However, new supply restriction can continue to be encountered until the job completes. By Property 6.3 and the definition of x-sufficient in Definition 6.8, we have that each $\tau_{i,k}$ completes by time $y_{i,k} + x_i(y_{i,k}) + e_{i,k}$ for some x-sufficient $x_i(y_{i,k})$. We reference such a $x_i(y_{i,k})$ in the following lemma, and we instantiate it to a specific value in Theorem 6.5 below.

Lemma 6.5. *For arbitrary job $\tau_{i,k}$ and x-sufficient $x_i(y_{i,k})$, at most*

$$(m - u_{tot})((y_{i,k} - t^b_{i,k}) + x_i(y_{i,k}) + e_{i,k}) + O_{i,k}$$

units of competing supply restriction exist after $t^b_{i,k}$, where

$$O_{i,k} \triangleq \sum_{P_p \in P} o_p(t^b_{i,k}, t^c_{i,k}). \tag{6.20}$$

(Recall that we describe supply restriction as "competing" if it occurs before $t^c_{i,k}$.)

Proof. By the definition of $\beta_p(t^b_{i,k}, t^c_{i,k})$ in Definition 6.7, the amount of time that P_p is *not* available to level C over $[t^b_{i,k}, t^c_{i,k})$ is

$$\begin{aligned}
&(t^c_{i,k} - t^b_{i,k}) - \beta_p(t^b_{i,k}, t^c_{i,k})\\
\leq\ &\{\text{By (6.9)}\}\\
&(t^c_{i,k} - t^b_{i,k}) - \widehat{u_p} \cdot (t^c_{i,k} - t^b_{i,k}) + o_p(t^b_{i,k}, t^c_{i,k}).\\
=\ &\{\text{Rearranging}\}\\
&(1 - \widehat{u_p}) \cdot (t^c_{i,k} - t^b_{i,k}) + o_p(t^b_{i,k}, t^c_{i,k}).
\end{aligned}$$

This quantity upper bounds the competing supply restriction on P_p after $t^b_{i,k}$. Summing over all processors, the total amount of competing supply restriction on all processors is at most

$$\begin{aligned}
&\sum_{P_p \in P} ((1 - \widehat{u_p}) \cdot (t^c_{i,k} - t^b_{i,k}) + o_p(t^b_{i,k}, t^c_{i,k}))\\
=\ &\{\text{Rearranging}\}
\end{aligned}$$

$$\left(\sum_{P_p \in P} 1 - \sum_{P_p \in P} \widehat{u_p}\right) \cdot (t^c_{i,k} - t^b_{i,k}) + \sum_{P_p \in P} o_p(t^b_{i,k}, t^c_{i,k})$$

$=$ {Because there are m processors in P, by the definition of u_{tot} in (6.10), and by the definition of $O_{i,k}$ in (6.20)}

$$(m - u_{tot}) \cdot (t^c_{i,k} - t^b_{i,k}) + O_{i,k}$$

$\leq$ {By the definition of x-sufficient in Definition 6.8}

$$(m - u_{tot}) \cdot (y_{i,k} + x_i(y_{i,k}) + e_{i,k} - t^b_{i,k}) + O_{i,k}$$

$=$ {Rearranging}

$$(m - u_{tot}) \cdot ((y_{i,k} - t^b_{i,k}) + x_i(y_{i,k}) + e_{i,k}) + O_{i,k}. \qquad \square$$

We now compute a lateness bound that accounts for both work and competing supply restriction. As discussed earlier, we will analyze the behavior of the system after $y_{i,k}$, when new job arrivals cannot preempt $\tau_{i,k}$.

We will now consider how to bound the total length of sub-intervals of Type 1 as described earlier, during which τ_i does not execute because all processors are occupied by competing work or supply restriction. We will do so by bounding the total amount of competing work and supply restriction after $y_{i,k}$. Recall that in Lemmas 6.4–6.5, competing work and supply restriction were determined after $t^b_{i,k}$ rather than after $y_{i,k}$. The following property will allow us to transition to reasoning after $y_{i,k}$. It holds by the definition of $t^b_{i,k}$ in Definition 6.12.

Property 6.5. *$m \cdot (y_{i,k} - t^b_{i,k})$ units of work and/or supply restriction complete in $[t^b_{i,k}, y_{i,k})$.*

We now bound the amount of competing work and supply restriction in $[y_{i,k}, t^c_{i,k})$.

Lemma 6.6. *For arbitrary $\tau_{i,k}$ and x-sufficient $x_i(y_{i,k})$, at most*

$$W_{i,k} - R_{i,k} + (m - u_{tot})(x_i(y_{i,k}) + e_{i,k}) + O_{i,k} - e^r_{i,k}(y_{i,k}) - e^p_{i,k}$$

units of competing work and supply restriction remain at $y_{i,k}$, where

$$R_{i,k} \triangleq u_{tot}(y_{i,k} - t^b_{i,k}). \tag{6.21}$$

Proof. By Lemma 6.4, the total amount of remaining work at $t^b_{i,k}$ for jobs $\tau_{j,\ell}$ with $y_{j,\ell} \leq y_{i,k}$ is $W_{i,k}$. Adding this to the bound on competing supply restriction in Lemma 6.5, there are at most

$$W_{i,k} + (m - u_{tot})((y_{i,k} - t^b_{i,k}) + x_i(y_{i,k}) + e_{i,k}) + O_{i,k}$$

units of work and supply restriction remaining after $t^b_{i,k}$. Of this work and supply restriction, by Property 6.5, the amount remaining at $y_{i,k}$ is at most

$$\begin{aligned}
&W_{i,k} + (m - u_{tot})((y_{i,k} - t^b_{i,k}) + x_i(y_{i,k}) + e_{i,k}) + O_{i,k} - m \cdot (y_{i,k} - t^b_{i,k}) \\
=&\ \{\text{Rearranging}\} \\
&W_{i,k} - u_{tot} \cdot (y_{i,k} - t^b_{i,k}) + (m - u_{tot}) \cdot (x_i(y_{i,k}) + e_{i,k}) + O_{i,k} \\
=&\ \{\text{By the definition of } R_{i,k} \text{ in (6.21)}\} \\
&W_{i,k} - R_{i,k} + (m - u_{tot}) \cdot (x_i(y_{i,k}) + e_{i,k}) + O_{i,k}. \qquad (6.22)
\end{aligned}$$

Of this remaining work and supply restriction, by the definition of $e^p_{i,k}$ in Definition 6.9, $e^p_{i,k}$ units are due to jobs of τ_i prior to $\tau_{i,k}$, and by the definition of $e^r_{i,k}(y_{i,k})$ in Definition 6.3, $e^r_{i,k}(y_{i,k})$ units are due to $\tau_{i,k}$ itself. The lemma follows immediately. □

Next, we prove a result that formalizes our initial discussion of the Many Tasks Case, allowing us to provide a lower bound on competing work and supply restriction within Type 2 sub-intervals.

Lemma 6.7. *If $\tau_{i,k}$ is m-dominant for L, then whenever jobs of τ_i prior to $\tau_{i,k}$ are running after $y_{i,k}$, at least $m - L - 1$ processors are occupied by competing work or supply restriction.*

Proof. If $k = 0$ or $t^c_{i,k-1} \leq y_{i,k}$, then the lemma is vacuously true, as no jobs of τ_i prior to $\tau_{i,k}$ execute after $y_{i,k}$. Therefore, we assume that $t^c_{i,k-1} > y_{i,k}$.

By the definition of m-dominant for L in Definition 6.11, there are at least $m - L$ tasks that have work remaining at $t^c_{i,k-1}$. Therefore, over $[y_{i,k}, t^c_{i,k-1})$, there are always at least $m - L - 1$ tasks *other* than τ_i eligible for execution. All of these tasks must be executing at all times over $[y_{i,k}, t^c_{i,k-1})$, except when all m processors are occupied are occupied by work or supply restriction. Because $L \geq 0$ by Definition 6.11, the lemma follows. □

We now bound the completion time of $\tau_{i,k}$.

Lemma 6.8. *If $\tau_{i,k}$ is m-dominant for L, $t^c_{i,k} > y_{i,k} + e_{i,k}$, and $x_i(y_{i,k})$ is x-sufficient, then*

$$t^c_{i,k} \leq y_{i,k} + \frac{W_{i,k} - R_{i,k} + (m - u_{tot})(x_i(y_{i,k}) + e_{i,k}) + O_{i,k} - e_{i,k} + Le^p_{i,k}}{m} + e_{i,k}.$$

Proof. By Lemma 6.7, during any instant within any sub-interval of Type 2 (as depicted in Figure 6.4(b)), there are at least $m - L - 1$ processors executing competing work or supply restriction. Recall that, by the definition of "Type 2" and the definition of $e^p_{i,k}$ in Definition 6.9, the total length of such sub-intervals is $e^p_{i,k}$.

By Lemma 6.6 there can be at most

$$\begin{aligned} c &\triangleq W_{i,k} - R_{i,k} + (m - u_{tot})(x_i(y_{i,k}) + e_{i,k}) + O_{i,k} - e^r_{i,k}(y_{i,k}) - e^p_{i,k} - (m - L - 1)e^p_{i,k} \\ &= \{\text{Rearranging}\} \\ &\quad W_{i,k} - R_{i,k} + (m - u_{tot})(x_i(y_{i,k}) + e_{i,k}) + O_{i,k} - e^r_{i,k}(y_{i,k}) + (L - m)e^p_{i,k} \end{aligned} \tag{6.23}$$

units of computing work and supply restriction after $y_{i,k}$ that do not run concurrently with jobs of τ_i prior to $\tau_{i,k}$. This bound includes all work and/or supply restriction in sub-intervals of Type 1. All m processors are occupied by work or supply restriction in such sub-intervals, so the total length of such sub-intervals is at most c/m.

Recall that the total length of Type 3 sub-intervals (in which $\tau_{i,k}$ runs) is $e^r_{i,k}(y_{i,k})$. Therefore,

$$\begin{aligned} t^c_{i,k} &\leq \{\text{Adding the total length of each type of sub-interval to } y_{i,k}\} \\ &\quad y_{i,k} + \frac{c}{m} + e^p_{i,k} + e^r_{i,k}(y_{i,k}) \\ &= \{\text{By (6.23)}\} \\ &\quad y_{i,k} + \frac{W_{i,k} - R_{i,k} + (m - u_{tot})(x_i(y_{i,k}) + e_{i,k}) + O_{i,k} - e^r_{i,k}(y_{i,k}) + (L - m)e^p_{i,k}}{m} + e^p_{i,k} + e^r_{i,k}(y_{i,k}) \\ &= \{\text{Rearranging}\} \\ &\quad y_{i,k} + \frac{W_{i,k} - R_{i,k} + (m - u_{tot})(x_i(y_{i,k}) + e_{i,k}) + O_{i,k} + Le^p_{i,k}}{m} + \frac{m-1}{m} \cdot e^r_{i,k}(y_{i,k}) \\ &\leq \{\text{Because } e^r_{i,k}(y_{i,k}) \leq e_{i,k} \text{ and } m > 1\} \\ &\quad y_{i,k} + \frac{W_{i,k} - R_{i,k} + (m - u_{tot})(x_i(y_{i,k}) + e_{i,k}) + O_{i,k} + Le^p_{i,k}}{m} + \frac{m-1}{m} \cdot e_{i,k} \end{aligned}$$

$$= \{\text{Rearranging}\}$$

$$y_{i,k} + \frac{W_{i,k} - R_{i,k} + (m - u_{tot})(x_i\left(y_{i,k}\right) + e_{i,k}) + O_{i,k} - e_{i,k} + Le^p_{i,k}}{m} + e_{i,k}. \qquad \square$$

The next theorem provides the actual bound on $x_i\left(t_a\right)$.

Theorem 6.5. *If $t_a = y_{i,k}$ for some k, $t^c_{i,k} > y_{i,k} + e_{i,k}$ and $\tau_{i,k}$ is m-dominant for L, then $x_i\left(t_a\right) = x^m_{i,k}$ is x-sufficient, where*

$$x^m_{i,k} \triangleq \frac{W_{i,k} - R_{i,k} + (m - u_{tot} - 1)e_{i,k} + O_{i,k} + Le^p_{i,k}}{u_{tot}} \tag{6.24}$$

(many tasks).

Proof. We let

$$x^t_{i,k} \triangleq t^c_{i,k} - e_{i,k} - y_{i,k} \tag{6.25}$$

($t$ight). Rearranging,

$$t^c_{i,k} = y_{i,k} + x^t_{i,k} + e_{i,k}. \tag{6.26}$$

Because $t^c_{i,k} > y_{i,k} + e_{i,k}$, by (6.25)–(6.26) and the definition of x-sufficient in Definition 6.8, $x_i\left(y_{i,k}\right) = x^t_{i,k}$ is x-sufficient.

Therefore, by Lemma 6.8 with $x_i\left(y_{i,k}\right) = x^t_{i,k}$ and (6.26),

$$x^t_{i,k} \le \frac{W_{i,k} - R_{i,k} + (m - u_{tot}) \cdot (x^t_{i,k} + e_{i,k}) + O_{i,k} - e_{i,k} + L \cdot e^p_{i,k}}{m}.$$

We solve for $x^t_{i,k}$. First, we will add $\frac{u_{tot} - m}{m} \cdot x^t_{i,k}$ to both sides, which yields

$$\frac{u_{tot}}{m} \cdot x^t_{i,k} \le \frac{W_{i,k} - R_{i,k} + (m - u_{tot}) \cdot e_{i,k} + O_{i,k} - e_{i,k} + L \cdot e^p_{i,k}}{m}.$$

We then multiply both sides by $\frac{u_{tot}}{m}$. Because $u_{tot} > 0$ and $m > 0$,

$$x^t_{i,k} \le \frac{W_{i,k} - R_{i,k} + (m - u_{tot}) \cdot e_{i,k} + O_{i,k} - e_{i,k} + L \cdot e^p_{i,k}}{u_{tot}}$$

$$= \{\text{Rearranging}\}$$

$$\frac{W_{i,k} - R_{i,k} + (m - u_{tot} - 1) \cdot e_{i,k} + O_{i,k} + L \cdot e^p_{i,k}}{u_{tot}}$$

$$= \{\text{By the definition of } x^m_{i,k} \text{ in (6.24)}\}$$

$$x^m_{i,k}.$$

Because $x_i(y_{i,k}) = x^t_{i,k}$ is x-sufficient, by Property 6.4 with $c_0 = x^t_{i,k}$ and $c_1 = x^m_{i,k}$, $x_i(y_{i,k}) = x^m_{i,k}$ is x-sufficient. Because $t_a = y_{i,k}$, the theorem follows. □

6.3 Dissipation Bounds

The response-time analysis provided in Section 6.2 is very general, in order to provide an accurate characterization of behavior in overload situations. In particular, it can even be used to analyze the behavior of systems where no per-task bound on response times exists. In this section, we consider systems that have per-task response time bounds in the absence of overload. In other words, each task has some constant $x^s_i(1)$ such that, if $s(t) = 1$ for all t, then $x_i(t) = x^s_i(1)$ is x-sufficient for all τ_i and time t. (The reason for the "1" argument will be described later.) In this section, we analyze a system where an overload actually does occur, but the overload is transient. This situation is similar to that depicted in Figure 6.1(c), where both τ_{A1} and τ_{A2} have jobs starting at actual time 12 that run for longer than their level-C PWCETs, but no later jobs that do so.

If we could determine choices of $x^s_i(1)$ that are as tight as possible, it is likely that $x_i(t) = x^s_i(1)$ would no longer be x-sufficient for $t > 12$. However, due to the pessimism of the analysis here, where $x^s_i(1)$ is not tight, $x_i(t) = x^s_i(1)$ may actually remain x-sufficient after this time. For illustration purposes, when discussing Figure 6.1(c), we consider a system under *ideal analysis*, where $x^s_i(1)$ is as tight as possible. We would like to return the system to a state where $x_i(t) = x^s_i(1)$ is x-sufficient for all τ_i and all t greater than some t_n (*n*ormal operation). In this section, we demonstrate a method to provide such a guarantee, *provided that we use the (not tight) analytically-derived $x^s_i(1)$ described herein.*

In Figure 6.7, we depict many details of our analysis of dissipation bounds. The first of these details is depicted at the top of the figure: the three intervals into which we divide time. The first is the *overload* interval, $[0, t_r)$, from the beginning of the schedule until after a transient overload has passed. If we had ideal analysis, this interval would occur from actual time 0 to actual time 19 in Figure 6.1(c). We make very few assumptions about the behavior of the system in $[0, t_r)$, primarily using the analysis from Section 6.2. This allows us to account for any overload condition allowed by our general model. The second considered interval is the *recovery* interval, $[t_r, t_n)$, during which the virtual time clock operates at a slower rate in order

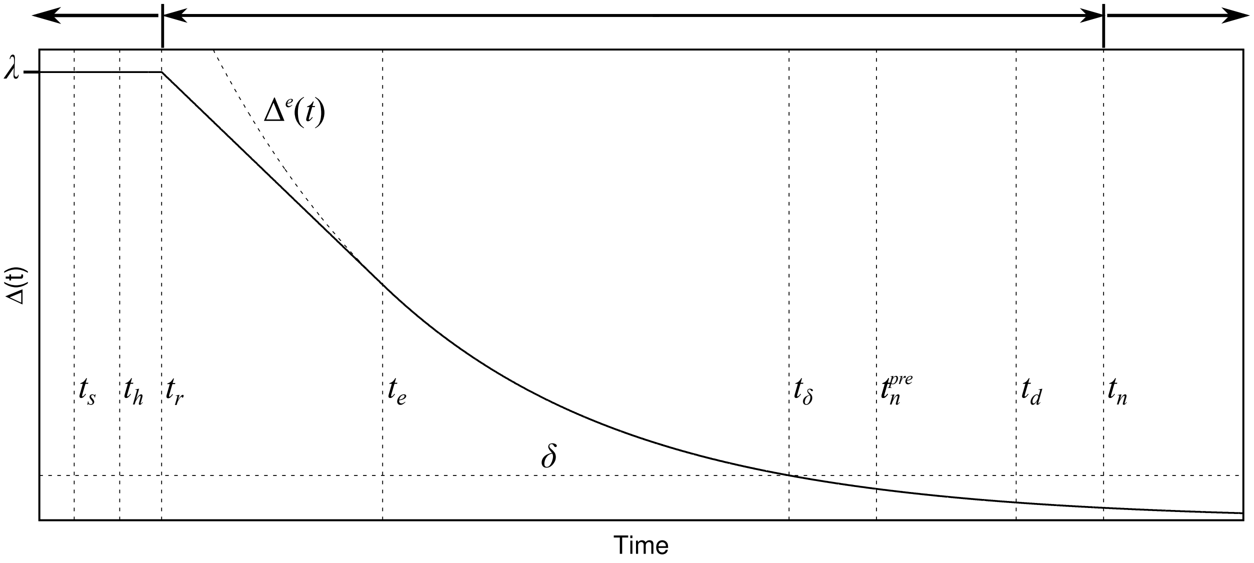


Figure 6.7: Graph of $\Delta(t)$, marked with various terms used in its definition and analysis.

to recover from the overload. Under ideal analysis, this would occur from actual time 19 to actual time 24 in Figure 6.1(c). The final interval we consider is the *normal* interval, $[t_n, \infty)$, when the system again operates normally. The virtual time clock executes at full speed during the normal interval. Under ideal analysis, this would occur from actual time 24 onwards in Figure 6.1(c).

In order to specify the boundaries between these intervals, we define several variables. t_s is defined to be the time at which the virtual clock actually slows. t_n will be defined as the time when the virtual clock can be returned to a normal speed. We note that the virtual clock can be returned to a normal speed at a *later* time without compromising correctness. We assume that the virtual clock is slowed to a constant speed s_r from t_s to t_n, as specified in the following property.

Property 6.6. *For all* $t \in [t_s, t_n)$, $s(t) = s_r < 1$.

In Figure 6.1(c), $s_r = 0.5$. Similarly, the following property describes the behavior of the virtual clock after the system has returned to normal.

Property 6.7. *If* $t \in [t_n, \infty)$, *then* $s(t) = 1$.

Because the speed of the virtual clock is determined by the OS, it is always possible to ensure that both properties hold.

Although the virtual clock is actually slowed at time t_s, for our analysis within $[t_r, t_n)$, it will often be convenient to assume that that the virtual clock has been operating at a constant rate for a period of time. Furthermore, we will also need to assume that overload does not occur in the recovery interval in order to make guarantees, even though unexpected overload could continue to occur even after t_s.

We define t_h (overload *h*alts) as the time after which a transient overload is no longer occurring. For our analysis, we assume that no overload occurs at all after t_h at all, as specified in Properties 6.8 and 6.9 below. If this is not the case, then the analysis remains correct until those properties are violated.

Property 6.8. *Each task τ_i has a constant $C_i \le T_i$ such that for any $\tau_{i,k}$, if $t^c_{i,k} \ge t_h$, then $e_{i,k} \le C_i$.*

Property 6.9. *For each P_p, there is some constant σ_p such that if $t_h \le t_0 \le t_1$, then $o_p(t_0, t_1) \le \widehat{u_p}\sigma_p$.*

We do not specify any upper bound on t_1 due to our assumption that *no* new overload occurs after t_h.

We now define the start of the recovery interval.

Definition 6.13. t_r is the earliest time not before t_h or t_s such that every job with $y_{i,k} < t_s$ is complete.

There can only be a finite number of jobs with $y_{i,k} < t_s$, so by Property 6.3, such a time must exist as long as t_s exists.

For convenience, we state the following properties about t_r. These properties follow from the definition of t_r in Definition 6.13 and Properties 6.8 and 6.9.

Property 6.10. *If any $\tau_{i,k}$ is pending at t_r, then $y_{i,k} \ge t_s$.*

Property 6.11. *Each task τ_i has a constant $C_i \le T_i$ such that for any $\tau_{i,k}$, if $t^c_{i,k} \ge t_r$, then $e_{i,k} \le C_i$.*

Property 6.12. *For each P_p, there is some constant σ_p such that if $t_r \le t_0 \le t_1$, then $o_p(t_0, t_1) \le \widehat{u_p}\sigma_p$.*

Property 6.11 states that C_i is the worst-case execution time for any job of τ_i that influences our analysis within $[t_r, \infty)$. Property 6.12 eliminates some of the generality of our supply model from t_r onward, so that our supply model becomes identical to that used in Leontyev and Anderson (2010) from t_r onward.

Property 6.10 will typically be used with the following lemma.

Lemma 6.9. *If some $\tau_{j,\ell}$ is pending at time $t_2 \ge t_r$, then $y_{j,\ell} \ge t_s$.*

Proof. If $\tau_{j,\ell}$ is pending at t_r, then by Property 6.10, $y_{j,\ell} \ge t_s$. Otherwise,

$$y_{j,\ell} \ge r_{j,\ell}$$

$\geq$ {Because $\tau_{j,\ell}$ is not pending at t_r but is pending at $t_2 \geq t_r$}

t_r

$\geq$ {By the definition of t_r in Definition 6.13}

t_s. □

In light of Property 6.11, we define a task's *base utilization* (with respect to virtual time)

$$U_i^v = \frac{C_i}{T_i} \tag{6.27}$$

and its recovery utilization (with respect to actual time in $[t_r, t_n)$)

$$U_i^r = U_i^v \cdot s_r. \tag{6.28}$$

Observe that the utilization of τ_i with respect to actual time in $[t_n, \infty)$ is simply U_i^v, because $s(t) = 1$ for all $t \in [t_n, \infty)$.

If we can guarantee that $x_i(t) = x_i^s(1)$ is x-sufficient for $t \in [t_n, \infty)$ under Properties 6.6–6.12, then we define a *dissipation bound* as $t_n - t_h$.

Whenever $s(t)$ remains constant over an interval (as it does over $[t_r, t_n)$ and $[t_n, \infty)$), it is possible to correctly choose $x_i(t)$ such that it asymptotically approaches a constant value. We will below define this (task-dependent) constant value as $x_i^s(s_I)$, where s_I is the constant value of $s(t)$ (s_r in $[t_r, t_n)$ and 1 in $[t_n, \infty)$). We will then define a task-independent function $\Delta(t)$ that guarantees that $x_i(t) = x_i^s(s_r) + \Delta(t)$ is x-sufficient for every τ_i and time $t \in [t_r, t_n)$, as graphed in Figure 6.7.

Recall that, in Section 6.2, L was arbitrary for each $\tau_{i,k}$. Our analysis will require us to choose a particular L for each *task*, so in Section 6.3.1 below, we discuss how to make this choice. In Section 6.3.2, we then turn our attention to formally defining $x_i^s(s_I)$ and $\Delta(t)$. Then, in Section 6.3.3, we formally prove that $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient for $t_a \in [t_r, t_n)$. In Section 6.3.4 we then upper bound t_n. Finally, in Section 6.3.5, we formally prove that $x_i(t) = x_i^s(1)$ is x-sufficient for $t \in [t_n, \infty)$.

6.3.1 Choosing L

In Section 6.2, L was arbitrary for any $\tau_{i,k}$. In this subsection, we will choose a specific per-task L_i that will take the place of L in several of our bounds. Because L_i will appear in our definition of $x_i^s(s_I)$, we first describe its selection here. We will then define $x_i^s(s_I)$ and $\Delta(t)$ in Section 6.3.2.

The choice of L appears in the definition of $x_{i,k}^m$ in (6.24), in the term $Le_{i,k}^p$, and in the definition of $x_{i,k}^f$ in (6.15), in the argument to $A_{i,k}(m-L-1)$. In order to analyze $x_{i,k}^f$, we first upper bound $A_{i,k}(m-L-1)$ in the case that will be relevant to our choice of $x_i^s(s_I)$. Lemma 6.10 does so, using arbitrary $v = m-L-1$ to match the notation used in Lemma 6.2. Let v be an integer with $0 \le v < m$, and let

$$A_i^{rn}(v) \triangleq \begin{cases} \frac{C_i+\sum_{P_p\in\Theta_{rn}}\widehat{u_p}\sigma_p}{1-v+\sum_{P_p\in\Theta_{rn}}\widehat{u_p}} & \text{If } 1-v+\sum_{P_p\in\Theta_{rn}}\widehat{u_p} > 0 \\ \infty & \text{Otherwise,} \end{cases} \tag{6.29}$$

(for *r*ecovery and *n*ormal intervals), where Θ_{rn} is the set of v processors that minimizes $A_i^{rn}(v)$.

Lemma 6.10. *$A_i^{rn}(v) > 0$ holds. Furthermore, if*

$$k > 0 \tag{6.30}$$

and

$$t_{i,k-1}^c > t_r, \tag{6.31}$$

then $A_{i,k}(v) \le A_i^{rn}(v)$ and $A_{i,k}(v) - e_{i,k} \le A_i^{rn}(v) - C_i$.

Proof. If $A_i^{rn}(v) = \infty$, then the lemma holds. Furthermore, if $A_{i,k}(v) = \infty$, then by the definition of $A_{i,k}(v)$ in (6.13), for any choice of v processors Θ, $1-v+\sum_{P_p\in\Theta}\widehat{u_p} \le 0$. Therefore, by the definition of $A_i^{rn}(v)$ in (6.29), $A_i^{rn}(v) = \infty$, and the lemma holds. Thus, we assume that $A_i^{rn}(v)$ is finite, implying by the definition of $A_i^{rn}(v)$ in (6.29) that

$$1-v+\sum_{P_p\in\Theta_{rn}}\widehat{u_p} > 0, \tag{6.32}$$

and that $A_{i,k}(v)$ is finite, implying by the definition of $A_{i,k}(v)$ in (6.13) that

$$1-v+\sum_{P_p\in\Theta}\widehat{u_p} > 0. \tag{6.33}$$

We first show that $A_i^{rn}(v) > 0$. To do so, we first show that, for arbitrary P_p,

$$\begin{aligned} \widehat{u_p}\sigma_p \geq\ & \{\text{By Property 6.12}\} \\ & o_p(t_r, t_r) \\ \geq\ & \{\text{By the definition of } o_p(t_r, t_r) \text{ in (6.8)}\} \\ & 0. \end{aligned} \tag{6.34}$$

Thus, we have

$$\begin{aligned} A_i^{rn}(v) =\ & \frac{C_i + \sum_{P_p \in \Theta_{rn}} \widehat{u_p}\sigma_p}{1 - v + \sum_{P_p \in \Theta_{rn}} \widehat{u_p}} \\ >\ & \{\text{Because } C_i > 0\text{, because the denominator is positive by (6.32), and by (6.34)}\} \\ & 0. \end{aligned}$$

For the remainder of the proof, we assume that (6.30) and (6.31) hold. Thus, we also have

$$\begin{aligned} t_{i,k}^c >\ & \{\text{By our task model and (6.30)}\} \\ & t_{i,k-1}^c \\ >\ & \{\text{By (6.31)}\} \\ & t_r. \end{aligned} \tag{6.35}$$

Therefore, we have

$$\begin{aligned} A_i^{rn}(v) =\ & \{\text{By the definition of } A_i^{rn}(v) \text{ in (6.29) and by (6.32)}\} \\ & \frac{C_i + \sum_{P_p \in \Theta_{rn}} \widehat{u_p}\sigma_p}{1 - v + \sum_{P_p \in \Theta_{rn}} \widehat{u_p}} \\ \geq\ & \{\text{By Property 6.12 and (6.31), and because the denominator is positive by (6.32)}\} \\ & \frac{C_i + \sum_{P_p \in \Theta_{rn}} o_i(t_{i,k-1}^c, t_{i,k}^c)}{1 - v + \sum_{P_p \in \Theta_{rn}} \widehat{u_p}} \\ \geq\ & \{\text{By Property 6.11 and (6.35), and because the denominator is positive by (6.32)}\} \\ & \frac{e_{i,k} + \sum_{P_p \in \Theta_{rn}} o_i(t_{i,k-1}^c, t_{i,k}^c)}{1 - v + \sum_{P_p \in \Theta_{rn}} \widehat{u_p}} \end{aligned}$$

$$\begin{aligned}
&\geq \{\text{Because } \Theta \text{ (as defined with (6.13)) is chosen to minimize } A_{i,k}(v)\text{, and because}\\
&\quad \text{the denominator is positive by (6.32)}\}\\
&\quad A_{i,k}(v).
\end{aligned}$$

Additionally,

$$\begin{aligned}
1 - v + \sum_{P_p \in \Theta_{rn}} \widehat{u_p} \leq{} & \{\text{Because each } \widehat{u_p} \leq 1\}\\
& 1 - v + \sum_{P_p \in \Theta_{rn}} 1\\
={} & \{\text{Because there are } v \text{ processors in } \Theta_{rn}\}\\
& 1. \qquad (6.36)
\end{aligned}$$

Therefore,

$$\begin{aligned}
A_i^{rn}(v) - C_i ={} & \{\text{By the definition of } A_i^{rn}(v) \text{ in (6.29) and because the denominator is positive by (6.32)}\}\\
& \frac{C_i + \sum_{P_p \in \Theta_{rn}} \widehat{u_p} \sigma_p}{1 - v + \sum_{P_p \in \Theta_{rn}} \widehat{u_p}} - C_i\\
\geq{} & \{\text{By Property 6.12 and (6.31), and because the denominator is positive by (6.32)}\}\\
& \frac{C_i + \sum_{P_p \in \Theta_{rn}} o_i(t_{i,k-1}^c, t_{i,k}^c)}{1 - v + \sum_{P_p \in \Theta_{rn}} \widehat{u_p}} - C_i\\
\geq{} & \{\text{By Property 6.11 and (6.35), because the denominator is positive by (6.32), and by (6.36)}\}\\
& \frac{e_{i,k} + \sum_{P_p \in \Theta_{rn}} o_i(t_{i,k-1}^c, t_{i,k}^c)}{1 - v + \sum_{P_p \in \Theta_{rn}} \widehat{u_p}} - e_{i,k}\\
\geq{} & \{\text{Because } \Theta \text{ (as defined with (6.13)) is chosen to minimize } A_{i,k}(v)\text{, and because the}\\
& \text{denominator is positive by (6.32)}\}\\
& A_{i,k}(v) - e_{i,k}.
\end{aligned}$$

□

We now define our choice of L_i.

Definition 6.14. For each τ_i, L_i is the smallest integer such that $0 \leq L_i < m$ and $A_i^{rn}(m - L_i - 1) \leq T_i$.

Such an integer must exist, because

$$\begin{aligned}
A_i^{rn}(m-(m-1)-1) &= \{\text{Rearranging}\} \\
&\quad A_i^{rn}(0) \\
&= \{\text{By the definition of } A_i^{rn}(0) \text{ in (6.29)}\} \\
&\quad C_i \\
&\leq \{\text{By Property 6.11}\} \\
&\quad T_i.
\end{aligned}$$

6.3.2 Defining $x_i^s(s_I)$ and $\Delta(t)$

In this subsection, we define $x_i^s(s_I)$ and $\Delta(t)$. We will prove in Section 6.3.3 below that they can be used to obtain x-sufficient bounds.

We first define $x_i^s(s_I)$. Its definition is implicit—$x_i^s(s_I)$ appears on both sides of (6.37) below.

Definition 6.15.

$$x_i^s(s_I) \triangleq \max\left\{0, \\ \frac{\sum_{m-1 \text{ largest}}(C_j + U_j^v \cdot s_I \cdot x_j^s(s_I) - S_j) + \sum_{\tau_j \in \tau} S_j + (m - u_{tot} - 1)C_i + O^{rn} + L_i \cdot U_i^v \cdot s_I \cdot x_i^s(s_I)}{u_{tot}}\right\}. \tag{6.37}$$

where

$$S_i \triangleq C_i \left(1 - \frac{Y_i}{T_i}\right), \tag{6.38}$$

and

$$O^{rn} \triangleq \sum_{P_p \in P} \widehat{u_p} \sigma_p \tag{6.39}$$

(for *r*ecovery and *n*ormal intervals).

In Appendix D, we discuss how to use linear programming to determine the specific value of $x_i^s(s_I)$, if it exists. We show that $x_i^s(s_I)$ exists if the provided linear program is feasible, and that the following condition is sufficient for feasibility.

Property 6.13.

$$\sum_{m-1 \text{ largest}} U_j^v + \max_{\tau_i \in \tau}(L_i \cdot U_i^v) < u_{tot}.$$

If Property 6.13 is not satisfied, then the methods provided in this chapter cannot provide dissipation bounds, but no response-time guarantees can be ensured in the absence of overload, either. Furthermore, our analysis also assumes the following property, without which bounded response times cannot be guaranteed even in the absence of overload.

Property 6.14.

$$\sum_{\tau_j \in \tau} U_j^v \leq u_{tot}.$$

In Appendix D, we also show that if $x_i^s(1)$ exists, then $x_i^s(s_I)$ must exist for all $s_I \leq 1$. Until Section 6.3.5, we will use $s_I = s_r$, because we are analyzing $[t_r, t_n)$.

We next define $\Delta(t)$. The definition of $\Delta(t)$ uses several upper bounds of quantities from Section 6.2. We will justify the correctness of these upper bounds in Section 6.3.3. We will describe each segment of $\Delta(t)$, as depicted in Figure 6.7, from left to right. We will provide necessary definitions as we proceed.

Observe in Figure 6.7 that for $t \leq t_r$, $\Delta(t)$ is constant. We will denote this constant value as λ, and we will define λ below. First, we describe a function closely related to $x_i(t)$ that will be used in defining λ. Observe in Definition 6.8 that the provided equation must hold for all $\tau_{i,k}$ with $y_{i,k} \leq t$. We define $\dot{x}_i(t)$ by changing this precondition to a strict inequality, in order to handle an edge case in our analysis.

Definition 6.16. $\dot{x}_i(t)$ is *$\dot{x}$-sufficient* if $\dot{x}_i(t) \geq 0$ and for all $\tau_{i,k}$ with $y_{i,k} < t$,

$$t_{i,k}^c \leq t + \dot{x}_i(t) + e_{i,k}.$$

With this definition in place, we now define λ.

Definition 6.17.

$$\lambda \triangleq \max\left\{ \max_{\tau_i \in \tau}(\dot{x}_i(t_r) - x_i^s(s_r) + A_i^{rn}(m - L_i - 1)), \right.$$

$$\delta,$$

$$\max_{\tau_{i,k} \in \psi}\left(\frac{W_{i,k}^o - R_{i,k}^o + (m - u_{tot} - 1)e_{i,k} + O_{i,k}^o + O^{rn} + L_i \cdot U_i^r \cdot x_i^s(s_r)}{u_{tot} - L_i \cdot U_i^r}\right),$$

$$\max_{\tau_{i,k}\in\kappa}(x_i\left(y_{i,k}\right)-x_i^s(s_r)),$$
$$\left.0\right\}, \tag{6.40}$$

where each $\dot{x}_i(t_r)$ is $\dot{x}$-sufficient (such values must exist by Property 6.3),

$$\delta \triangleq \min_{\tau_i\in\tau}(x_i^s(1)-x_i^s(s_r)), \tag{6.41}$$

ψ is the set of jobs with $y_{i,k}\in[t_r,\infty)$ and $t_{i,k}^b\in[0,t_r)$, κ is the set of jobs with $y_{i,k}\in[0,t_r)$ and $t_{i,k}^c\in[t_r,\infty)$, each $x_i\left(y_{i,k}\right)$ is x-sufficient (such values must exist by Property 6.3),

$$W_{i,k}^o \triangleq W_{i,k}-\sum_{\tau_j\in\tau}D_j^e(t_r,y_{i,k})+\sum_{\tau_j\in\tau}S_j \tag{6.42}$$

(for jobs with $t_{i,k}^b$ in the *o*verload interval), and

$$R_{i,k}^o \triangleq u_{tot}\cdot(t_r-t_{i,k}^b) \tag{6.43}$$

$$O_{i,k}^o \triangleq \sum_{P_p\in P}o_p(t_{i,k}^b,t_r) \tag{6.44}$$

(each for the *o*verload interval).

Observe in Figure 6.7 that, from t_r to t_e (switch to *e*xponential), $\Delta(t)$ is linear. We define this segment as its own function

$$\Delta^\ell(t) \triangleq \phi\cdot(t-t_r)+\lambda \tag{6.45}$$

(*l*inear), where

$$\phi \triangleq \max\left\{\max_{\tau_j\in\tau}\left(\frac{s_r\cdot A_j^{rn}(m-L_j-1)}{T_i}-1\right),\frac{\sum_{\tau_j\in\tau}U_j^r-u_{tot}}{u_{tot}}\right\}. \tag{6.46}$$

As can be seen in Figure 6.7, from t_e onward, $\Delta(t)$ decays exponentially. We will also define this segment as its own function,

$$\Delta^e(t) \triangleq \Delta^\ell(t_e)\cdot q^{\frac{t-t_e}{\rho}} \tag{6.47}$$

(*e*xponential), where

$$t_e \triangleq \begin{cases} t_r & \text{If } \lambda \le \phi \cdot \frac{\rho}{\ln q} \\ t_r + \frac{\rho}{\ln q} - \frac{\lambda}{\phi} & \text{Otherwise} \end{cases} \tag{6.48}$$

(switch to *e*xponential),

$$q \triangleq \frac{\sum_{m-1 \text{ largest}} U_j^r + \max_{\tau_j \in \tau}(L_j \cdot U_j^v \cdot s_r)}{u_{tot}}, \tag{6.49}$$

and

$$\rho \triangleq \max_{\tau_j \in \tau}(x_j^s(s_r) + \lambda + C_j). \tag{6.50}$$

We will show in Lemma 6.12 below that $t_e \ge t_r$. In order to do so, we first provide a result about the value of ϕ that will be used repeatedly in other places.

Lemma 6.11. $-1 < \phi < 0$.

Proof. By the definition of ϕ in (6.46), either $\phi = \frac{s_r \cdot A_j^{rn}(m-L_j-1)}{T_j} - 1$ for some τ_j, or $\phi = \frac{\sum_{\tau_j \in \tau} U_j^r - u_{tot}}{u_{tot}}$. We consider each of these cases.

Case 1: $\phi = \frac{s_r \cdot A_j^{rn}(m-L_j-1)}{T_j} - 1$ **for Some** τ_j**.** In this case, $\phi > -1$ because $s_r > 0$, $A_j^{rn}(m - L_j - 1) > 0$ by Lemma 6.10, and $T_i > 0$.

We also have

$$\begin{aligned} \phi &= \frac{s_r \cdot A_j^{rn}(m-L_j-1)}{T_j} - 1 \\ &\le \{\text{By the definition of } L_j \text{ in Definition 6.14}\} \\ &\quad \frac{s_r \cdot T_j}{T_j} - 1 \\ &= \{\text{Cancelling}\} \\ &\quad s_r - 1 \\ &< \{\text{By Property 6.6}\} \\ &\quad 0. \end{aligned}$$

Case 2: $\phi = \frac{\sum_{\tau_j \in \tau} U_j^r - u_{tot}}{u_{tot}}$. In this case, we have

$$\begin{aligned}
\phi &= \frac{\sum_{\tau_j \in \tau} U_j^r - u_{tot}}{u_{tot}} \\
&= \{\text{Rewriting}\} \\
&\quad \frac{\sum_{\tau_j \in \tau} U_j^r}{u_{tot}} - 1 \\
&> \{\text{Because each } U_j^r > 0 \text{ and } u_{tot} > 0\} \\
&\quad -1.
\end{aligned}$$

We also have

$$\begin{aligned}
\phi &= \frac{\sum_{\tau_j \in \tau} U_j^r - u_{tot}}{u_{tot}} \\
&= \{\text{By the definition of } U_j^r \text{ in (6.28)}\} \\
&\quad \frac{\sum_{\tau_j \in \tau} (U_j^v \cdot s_r) - u_{tot}}{u_{tot}} \\
&< \{\text{By Property 6.6}\} \\
&\quad \frac{\sum_{\tau_j \in \tau} U_j^v - u_{tot}}{u_{tot}} \\
&\leq \{\text{By Property 6.14}\} \\
&\quad \frac{u_{tot} - u_{tot}}{u_{tot}} \\
&= \{\text{Simplifying}\} \\
&\quad 0.
\end{aligned}$$

□

We now show that t_r and t_e occur in the order they appear in Figure 6.7.

Lemma 6.12. *$t_e \geq t_r$ holds. Moreover, if $\lambda > \phi \cdot \frac{\rho}{\ln q}$, then this inequality is strict.*

Proof. If $\lambda \leq \phi \cdot \frac{\rho}{\ln q}$, then the lemma follows immediately from the definition of t_e in (6.48).

Otherwise, we have

$$t_e = \{\text{By the definition of } t_e \text{ in (6.48)}\}$$

$$
\begin{aligned}
& t_r + \frac{\rho}{\ln q} - \frac{\lambda}{\phi} \\
> & \{\text{Because } \phi < 0 \text{ by Lemma 6.11 and } \lambda > \phi \cdot \tfrac{\rho}{\ln q}\} \\
& t_r + \frac{\rho}{\ln q} - \frac{\rho}{\ln q} \\
= & \{\text{Cancelling}\} \\
& t_r.
\end{aligned}
$$

□

Finally, we fully define $\Delta(t)$ for all t.

$$
\Delta(t) = \begin{cases} \lambda & \text{If } t \in (-\infty, t_r) \\ \Delta^{\ell}(t) & \text{If } t \in [t_r, t_e) \\ \Delta^{e}(t) & \text{If } t \in [t_e, \infty). \end{cases} \tag{6.51}
$$

6.3.3 Proving that $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient for $t_a \in [t_r, t_n)$

In this subsection, we provide a x-sufficient choice of $x_i(t_a)$ for each τ_i and $t_a \in [t_r, t_n)$. For each such combination of t_a and τ_i, we will exhaustively consider the cases depicted in Figure 6.8 to show that $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient. We will prove this result by induction on its correctness for smaller choices of t_a. Specifically, we will use the following precondition as needed

$$
x_j(y_{j,\ell}) = x_j^s(s_r) + \Delta(y_{j,\ell}) \text{ is } x\text{-sufficient for all } \tau_{j,\ell} \text{ such that } y_{j,\ell} \in [t_r, t_a). \tag{6.52}
$$

The proofs for the different cases can then be used with induction over jobs in PP order to prove that a particular $x_j(t_a) = x_j^s(s_r) + \Delta(t_a)$ is x-sufficient.

Furthermore, a necessary condition in Definition 6.8 for $x_i(t)$ to be x-sufficient is that $x_i(t) \geq 0$. Lemma 6.17 below establishes that this is the case for $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ for arbitrary t_a. By the definition of $x_i^s(s_r)$ in (6.37), $x_i^s(s_r)$ is nonnegative. Therefore, showing that $\Delta(t_a)$ is also nonnegative for arbitrary t_a will be sufficient to prove Lemma 6.17.

By the definition of $\Delta(t_a)$ in (6.51), we will consider the three intervals $(-\infty, t_r)$, $[t_r, t_e)$, and $[t_e, \infty)$ separately. $\Delta(t_a)$ is nonnegative for $t_a \in (-\infty, t_r)$, because by the definition of λ in (6.40), $\lambda \geq 0$. We thus consider $t_a \in [t_r, t_e)$. By the definition of $\Delta^{\ell}(t)$ in (6.45) and Lemma 6.11, $\Delta^{\ell}(t)$ is decreasing over $[t_r, t_e)$.

A. $t_a < y_{i,0}$ (Lemma 6.18).

B. $t_a = y_{i,k}$ for some k and $t^c_{i,k} \le y_{i,k} + e_{i,k}$ (Lemma 6.19).

C. $t_a \in (y_{i,k}, y_{i,k+1})$ for some k (Lemma 6.25).

D. $t_a = y_{i,k}$ for some k, $t^c_{i,k} > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is f-dominant for L_i (Lemma 6.31).

E. $t_a = y_{i,k}$ for some k, $t^c_{i,k} > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is m-dominant for L_i (Lemma 6.57).

Figure 6.8: Cases considered when proving that $x_i(t_a) = x^s_i(s_r) + \Delta(t_a)$ is x-sufficient for $t_a \in [t_r, t_n)$

We show, in Lemma 6.15 below, that $\Delta^\ell(t_e)$ is nonnegative. The result that $\Delta^\ell(t_e)$ is nonnegative will also be used to prove that $\Delta(t_a)$ is nonnegative for $t_a \in [t_e, \infty)$.

We need to show that $\Delta^\ell(t_e)$ is nonnegative. By the definition of t_e in (6.48), the value of t_e is dependent on $\ln q$. Thus, we first characterize the value of q.

Lemma 6.13. $0 < q < 1$.

Proof. We first show that $0 < q$. All variables that appear in the definition of q in (6.49) are nonnegative, and U^r_j for each τ_j is strictly positive. Therefore, $0 < q$.

We now show that $q < 1$. We have

$$\begin{aligned}
q = {} & \{\text{By the definition of } q \text{ in (6.49)}\} \\
& \frac{\sum_{m-1 \text{ largest}} U^r_j + \max_{\tau_j \in \tau}(L_j \cdot U^v_j \cdot s_r)}{u_{tot}} \\
= {} & \{\text{By the definition of } U^r_j \text{ in (6.28)}\} \\
& \frac{\sum_{m-1 \text{ largest}} (U^v_j \cdot s_r) + \max_{\tau_j \in \tau}(L_j \cdot U^v_j \cdot s_r)}{u_{tot}} \\
< {} & \{\text{By Property 6.6}\} \\
& \frac{\sum_{m-1 \text{ largest}} U^v_j + \max_{\tau_j \in \tau}(L_j \cdot U^v_j)}{u_{tot}} \\
< {} & \{\text{By Property 6.13}\} \\
& \frac{u_{tot}}{u_{tot}} \\
= {} & \{\text{Simplifying}\} \\
& 1.
\end{aligned}$$

□

Also by the definition of t_e in (6.48), the value of t_e is also dependent on ρ. Thus, we also characterize the value of ρ.

Lemma 6.14. $\rho > 0$.

Proof. We have

$$\begin{aligned}
\rho = &\{\text{By the definition of } \rho \text{ in (6.50)}\} \\
&\max_{\tau_j \in \tau}(x_j^s(s_r) + \lambda + C_j) \\
\geq &\{\text{By the definition of } x_j^s(s_r) \text{ in (6.37) and the definition of } \lambda \text{ in (6.40)}\} \\
&\max_{\tau_j \in \tau}(C_j) \\
> &\{\text{Because each } C_j > 0\} \\
&0.
\end{aligned}$$

□

We finally show that $\Delta^\ell(t_e)$ is nonnegative. Furthermore, the value of $\Delta^\ell(t_e)$ and the identical values of $\Delta(t_e)$ and $\Delta^e(t_e)$ are used in later proofs. For convenience, we consider all of these terms in a single lemma.

Lemma 6.15.

$$\Delta(t_e) = \Delta^e(t_e) = \Delta^\ell(t_e) = \min\left\{\lambda, \phi \cdot \frac{\rho}{\ln q}\right\} \geq 0.$$

Proof. We will demonstrate the equalities in the order they appear in the statement of the lemma.

First, we have

$$\begin{aligned}
\Delta(t_e) = &\{\text{By the definition of } \Delta(t_e) \text{ in (6.51)}\} \\
&\Delta^e(t_e) \\
= &\{\text{By the definition of } \Delta^e(t_e) \text{ in (6.47)}\} \\
&\Delta^\ell(t_e) \cdot q^{\frac{t_e - t_e}{\rho}} \\
= &\{\text{Simplifying}\} \\
&\Delta^\ell(t_e).
\end{aligned}$$

We now establish that $\Delta^\ell(t_e) = \min\left\{\lambda, \phi \cdot \frac{\rho}{\ln q}\right\}$ by considering two cases.

Case 1: $\lambda \leq \phi \cdot \frac{\rho}{\ln q}$. In this case,

$$
\begin{aligned}
\Delta^{\ell}\left(t_e\right) &= \{\text{By the definition of } t_e \text{ in (6.48)}\} \\
&\quad \Delta^{\ell}\left(t_r\right) \\
&= \{\text{By the definition of } \Delta^{\ell}\left(t_r\right) \text{ in (6.45)}\} \\
&\quad \phi \cdot \left(t_r - t_r\right) + \lambda \\
&= \{\text{Simplifying}\} \\
&\quad \lambda \\
&= \{\text{By the case we are considering}\} \\
&\quad \min\left\{\lambda, \phi \cdot \frac{\rho}{\ln q}\right\}.
\end{aligned}
$$

Case 2: $\lambda > \phi \cdot \frac{\rho}{\ln q}$. In this case,

$$
\begin{aligned}
\Delta^{\ell}\left(t_e\right) &= \{\text{By the definition of } t_e \text{ in (6.48)}\} \\
&\quad \Delta^{\ell}\left(t_r + \frac{\rho}{\ln q} - \frac{\lambda}{\phi}\right) \\
&= \{\text{By the definition of } \Delta^{\ell}\left(t_r + \tfrac{\rho}{\ln q} - \tfrac{\lambda}{\phi}\right) \text{ in (6.45)}\} \\
&\quad \phi \cdot \left(t_r + \frac{\rho}{\ln q} - \frac{\lambda}{\phi} - t_r\right) + \lambda \\
&= \{\text{Simplifying}\} \\
&\quad \phi \cdot \frac{\rho}{\ln q} \\
&= \{\text{By the case we are considering}\} \\
&\quad \min\left\{\lambda, \phi \cdot \frac{\rho}{\ln q}\right\}.
\end{aligned}
$$

Finally, we demonstrate that $\min\left\{\lambda, \phi \cdot \frac{\rho}{\ln q}\right\} \geq 0$. By the definition of λ in (6.40), $\lambda \geq 0$. Furthermore, because $\phi < 0$ by Lemma 6.11, $0 < q < 1$ by Lemma 6.13, and $\rho > 0$ by Lemma 6.14, $\phi \cdot \frac{\rho}{\ln q} > 0$. Therefore, $\min\left\{\lambda, \phi \cdot \frac{\rho}{\ln q}\right\} \geq 0$. □

We are now ready to establish that $x_i^s(s_r) + \Delta\left(t_a\right) \geq 0$. By Definition 6.8, this is a necessary condition to achieve the goal discussed at the beginning of Section 6.3.3 for $x_i\left(t_a\right) = x_i^s(s_r) + \Delta\left(t_a\right)$ to be x-sufficient.

Later, we will need to use the fact that $\Delta(t_a) \geq 0$, so we first provide this fact as a separate lemma.

Lemma 6.16. *For all* t_a, $\Delta(t_a) \geq 0$.

Proof. We consider three cases, depending on the value of t_a.

Case 1: $t_a \in (-\infty, t_r)$**.** In this case,

$$\begin{aligned}
\Delta(t_a) &= \{\text{By the definition of } \Delta(t_a) \text{ in (6.51)}\} \\
&\quad \lambda \\
&\geq \{\text{By the definition of } \lambda \text{ in (6.40)}\} \\
&\quad 0.
\end{aligned}$$

Case 2: $t_a \in [t_r, t_e)$**.** In this case,

$$\begin{aligned}
\Delta(t_a) &= \{\text{By the definition of } \Delta(t_a) \text{ in (6.51)}\} \\
&\quad \Delta^{\ell}(t_a) \\
&= \{\text{By the definition of } \Delta^{\ell}(t_a) \text{ in (6.45)}\} \\
&\quad \phi \cdot (t_a - t_r) + \lambda \\
&= \{\text{Rearranging}\} \\
&\quad \phi \cdot (t_e - t_r) + \lambda + \phi \cdot (t_a - t_e) \\
&= \{\text{By the definition of } \Delta^{\ell}(t_e) \text{ in (6.45)}\} \\
&\quad \Delta^{\ell}(t_e) + \phi \cdot (t_a - t_e) \\
&> \{\text{Because } \phi < 0 \text{ by Lemma 6.11 and } t_a < t_e \text{ by the case we are considering}\} \\
&\quad \Delta^{\ell}(t_e) \\
&\geq \{\text{By Lemma 6.15}\} \\
&\quad 0.
\end{aligned}$$

Case 3: $t_a \in [t_e, \infty)$**.** In this case,

$$\Delta(t_a) = \{\text{By the definition of } \Delta(t_a) \text{ in (6.51)}\}$$

$$
\begin{aligned}
&\Delta^{e}(t_a) \\
&= \{\text{By the definition of } \Delta^{e}(t_a) \text{ in (6.47)}\} \\
&\quad \Delta^{\ell}(t_e) \cdot q^{\frac{t_a - t_e}{\rho}} \\
&\geq \{\text{Because } \Delta^{\ell}(t_e) \geq 0 \text{ by Lemma 6.15 and } q > 0 \text{ by Lemma 6.13}\} \\
&\quad 0.
\end{aligned}
$$
□

We now provide the necessary bound on $x_i^s(s_r) + \Delta(t_a)$.

Lemma 6.17. *For all t_a, $x_i^s(s_r) + \Delta(t_a) \geq 0$.*

Proof. $x_i^s(s_r) \geq 0$ by the definition of $x_i^s(s_r)$ (i.e., using $s_I = s_r$) in (6.37). Thus, by Lemma 6.16, $x_i^s(s_r) + \Delta(t_a) \geq 0$. □

We now show that $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient by considering all the cases depicted in Figure 6.8, which match those in Figure 6.2 in Section 6.2.

We first consider Case A in Figure 6.8, in which $t_a < y_{i,0}$.

Lemma 6.18. *If $t_a < y_{i,0}$, then $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient.*

Proof. If $t_a < y_{i,0}$, then by Theorem 6.1, $x_i(t_a) = 0$ is x-sufficient. Furthermore, by Lemma 6.17, $x_i^s(s_r) + \Delta(t_a) \geq 0$. Therefore, by Property 6.4 with $c_0 = 0$ and $c_1 = x_i^s(s_r) + \Delta(t_a)$, $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient. □

The analysis of Case B in Figure 6.8 is simple.

Lemma 6.19. *If $t_a = y_{i,k}$ for some k and $t_{i,k}^c \leq y_{i,k} + e_{i,k}$, then $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient.*

Proof. The lemma follows immediately from Theorem 6.2, Lemma 6.17, and Property 6.4 with $c_0 = 0$ and $c_1 = x_i^s(s_r) + \Delta(t_a)$. □

We next consider Case C in Figure 6.8, in which $t_a \in (y_{i,k}, y_{i,k+1})$ for some k. For this case, as well as most subsequent cases, correctness is proved from the fact that the value of $\Delta(t)$ decreases sufficiently slowly as t increases. Therefore, we will prove Lemma 6.24 below, which explicitly bounds the decrease between two values of $\Delta(t)$. Our analysis will be based on the following form of the Fundamental Theorem of Calculus (FTC).

Fundamental Theorem of Calculus (FTC). *If $f(t)$ is continuous over $[t_0,t_1]$ and $f'(t)$ is the derivative of $f(t)$ at all but finitely many points within $[t_0,t_1]$, then*

$$f(t_1) = f(t_0) + \int_{t_0}^{t_1} f'(t)\,dt.$$

In order to use the FTC, we will demonstrate in Lemma 6.21 below that $\Delta(t)$ is continuous over all real numbers, and in Lemma 6.23 below we will provide a function $\Delta'(t)$ that is equal to the derivative of $\Delta(t)$ at all but finitely many points. Lemma 6.23 also provides bounds on $\Delta'(t)$ that are used to prove Lemma 6.24. In several parts of our analysis throughout this section, the value of $\Delta(t_r)$ and/or $\Delta^{\ell}(t_r)$ will be used. For convenience, we provide this value now in a separate lemma.

Lemma 6.20. $\Delta(t_r) = \Delta^{\ell}(t_r) = \lambda$.

Proof. We first establish that $\Delta(t_r) = \Delta^{\ell}(t_r)$. By Lemma 6.12, either $t_r = t_e$ or $t_r < t_e$. We consider each of these two cases.

Case 1: $t_e = t_r$. In this case,

$$\begin{aligned}\Delta(t_r) &= \Delta(t_e)\\ &= \{\text{By Lemma 6.15}\}\\ &\quad \Delta^{\ell}(t_e)\\ &= \{\text{Because } t_e = t_r\}\\ &\quad \Delta^{\ell}(t_r).\end{aligned}$$

Case 2: $t_r < t_e$. In this case, by the definition of $\Delta(t_r)$ in (6.51), $\Delta(t_r) = \Delta^{\ell}(t_r)$.

To conclude the proof, note that

$$\begin{aligned}\Delta^{\ell}(t_r) &= \{\text{By the definition of } \Delta^{\ell}(t_r) \text{ in (6.45)}\}\\ &\quad \phi\cdot(t_r - t_r) + \lambda\\ &= \{\text{Simplifying}\}\\ &\quad \lambda.\end{aligned}$$

□

The following property is a standard result in real analysis. (If $t_2 \leq t_1$, then it holds vacuously.)

Property 6.15. *For arbitrary t_0, t_1, t_2, and continuous functions $f(t)$ and $g(t)$, if $t_1 > t_0$, $\Delta(t) = f(t)$ for $t \in (t_0, t_1)$, $\Delta(t) = g(t)$ for $t \in [t_1, t_2)$, and $f(t_1) = g(t_1)$, then $\Delta(t)$ is continuous over $[t_1, t_2)$.*

We now use this property to prove that $\Delta(t)$ is continuous over the reals, so that we can use the FTC with $f(t) = \Delta(t)$.

Lemma 6.21. *$\Delta(t)$ is continuous over all real numbers.*

Proof. We first observe that, by the definition of $\Delta(t)$ in (6.51), $\Delta(t)$ is constant (and therefore continuous) over $(-\infty, t_r)$.

To prove that $\Delta(t)$ is continuous over $[t_r, \infty)$, we consider two cases, depending on the relationship between λ and $\phi \cdot \frac{\rho}{\ln q}$.

Case 1: $\lambda \leq \phi \cdot \frac{\rho}{\ln q}$. We will use Property 6.15 with $t_0 = -\infty$, $t_1 = t_r$, $t_2 = \infty$, $f(t) = \lambda$, and $g(t) = \Delta^e(t)$. It is trivially the case that $t_1 > t_0$.

In this case, by the definition of t_e in (6.48),

$$t_r = t_e. \tag{6.53}$$

Therefore, by the definition of $\Delta(t)$ in (6.51), $\Delta(t) = \lambda = f(t)$ for $t \in (t_0, t_1) = (-\infty, t_r)$, and $\Delta(t) = \Delta^e(t) = g(t)$ for $t \in [t_1, t_2) = [t_r, \infty)$, as desired.

$f(t)$ is continuous because it is constant. $g(t)$ is continuous by the definition of $\Delta^e(t)$ in (6.47), because exponential functions are continuous.

Finally, we show that

$$\begin{aligned} g(t_1) &= \Delta^e(t_r) \\ &= \{\text{By (6.53)}\} \\ &\quad \Delta^e(t_e) \\ &= \{\text{By Lemma 6.15 and the case we are considering}\} \\ &\quad \lambda \\ &= f(t_1). \end{aligned} \tag{6.54}$$

Therefore, all the preconditions for Property 6.15 are met, so $\Delta(t)$ is continuous over $[t_r, \infty)$.

Case 2: $\lambda > \phi \cdot \frac{\rho}{\ln q}$**.** We first prove that $\Delta(t)$ is continuous over $[t_r, t_e)$, using Property 6.15 with $f(t) = \lambda$, $g(t) = \Delta^\ell(t)$, $t_0 = -\infty$, $t_1 = t_r$, and $t_2 = t_e$. It is trivially the case that $t_1 > t_0$. By the definition of $\Delta(t)$ in (6.51), $\Delta(t) = \lambda = f(t)$ for $t \in (t_0, t_1) = (-\infty, t_r)$ and $\Delta(t) = \Delta^\ell(t) = g(t)$ for $t \in [t_1, t_2) = [t_r, t_e)$. $f(t)$ is continuous because it is constant. $g(t)$ is continuous by the definition of $\Delta^\ell(t)$ in (6.45), because linear functions are continuous. Furthermore,

$$\begin{aligned}
g(t_1) &= \Delta^\ell(t_r) \\
&= \{\text{By Lemma 6.20}\} \\
&\quad \lambda \\
&= f(t_1).
\end{aligned}$$

Therefore, all the preconditions for Property 6.15 are met, so $\Delta(t)$ is continuous over $[t_r, t_e)$.

We next prove that $\Delta(t)$ is continuous over $[t_e, \infty)$ using Property 6.15 with $f(t) = \Delta^\ell(t)$, $g(t) = \Delta^e(t)$, $t_0 = t_r$, $t_1 = t_e$, and $t_2 = \infty$. By Lemma 6.12 and the case we are considering (which implies $t_e \neq t_r$), $t_1 > t_0$. By the definition of $\Delta(t)$ in (6.51), $\Delta(t) = \Delta^\ell(t) = f(t)$ for $t \in (t_0, t_1) \subset [t_r, t_e)$ and $\Delta(t) = \Delta^e(t) = g(t)$ for $t \in [t_1, t_2) = [t_e, \infty)$. $f(t)$ is continuous by the definition of $\Delta^\ell(t)$ in (6.45), because linear functions are continuous. $g(t)$ is continuous by the definition of $\Delta^e(t)$ in (6.47), because exponential functions are continuous. Furthermore,

$$\begin{aligned}
g(t_1) &= \Delta^e(t_e) \\
&= \{\text{By Lemma 6.15}\} \\
&\quad \Delta^\ell(t_e) \\
&= f(t_1).
\end{aligned}$$

Therefore, all the preconditions for Property 6.15 are met, so $\Delta(t)$ is also continuous over $[t_e, \infty)$. We reasoned above that $\Delta(t)$ is continuous over $[t_r, t_e)$, so $\Delta(t)$ is continuous over $[t_r, \infty)$. □

In order to use the FTC, we must also provide a function $\Delta'(t)$ that is equal to the derivative of $\Delta(t)$ at all but finitely many points. Both for the purposes of determining $\Delta'(t)$, and for a later proof, we must reason about the derivative of $\Delta^e(t)$. The following lemma provides a necessary property of that derivative.

Lemma 6.22. *Let $\Delta^{e\prime}(t)$ be the derivative of $\Delta^e(t)$ with respect to t. If $t \geq t_e$, then $0 \geq \Delta^{e\prime}(t) \geq \phi$.*

Proof. If $t \geq t_e$, then we have

$$\begin{aligned}
\Delta^{e\prime}(t) &= \{\text{By the definition of } \Delta^e(t) \text{ in (6.47) and differentiation}\} \\
&\quad \frac{\ln q}{\rho} \cdot \Delta^{\ell}(t_e) \cdot q^{\frac{t-t_e}{\rho}} \qquad (6.55) \\
&\leq \{\text{By Lemma 6.15, because } 0 < q < 1 \text{ by Lemma 6.13 so that } \ln q < 0\} \\
&\quad 0.
\end{aligned}$$

Also,

$$\begin{aligned}
\Delta^{e\prime}(t) &= \{\text{By (6.55)}\} \\
&\quad \frac{\ln q}{\rho} \cdot \Delta^{\ell}(t_e) \cdot q^{\frac{t-t_e}{\rho}} \\
&\geq \{\text{By Lemma 6.15, because } 0 < q < 1 \text{ by Lemma 6.13 so that } \ln q < 0\} \\
&\quad \frac{\ln q}{\rho} \cdot \phi \cdot \frac{\rho}{\ln q} \cdot q^{\frac{t-t_e}{\rho}} \\
&= \{\text{Simplifying}\} \\
&\quad \phi \cdot q^{\frac{t-t_e}{\rho}}. \qquad (6.56)
\end{aligned}$$

By (6.56), because $\phi < 0$ by Lemma 6.11, $0 < q < 1$ by Lemma 6.13, and $t \geq t_e$, $\Delta^{e\prime}(t) \geq \phi$. □

We finally provide the derivative of $\Delta(t)$, except at $t = t_r$ and $t = t_e$ (finitely many points), and provide bounds on the resulting $\Delta'(t)$.

Lemma 6.23. *Let*

$$\Delta'(t) \triangleq \begin{cases} 0 & \text{If } t \in (-\infty, t_r) \\ \phi & \text{If } t \in [t_r, t_e) \\ \Delta^{e\prime}(t) & \text{If } t \in [t_e, \infty). \end{cases} \qquad (6.57)$$

$\Delta'(t)$ is the derivative of $\Delta(t)$ everywhere except at $t = t_r$ and at $t = t_e$. Furthermore, for all t, $\phi \le \Delta'(t) \le 0$.

Proof. We prove the lemma for arbitrary t, considering each of the three intervals that occur in the definition of $\Delta(t)$ in (6.51) and in the definition of $\Delta'(t)$ in (6.57).

Case 1: $t \in (-\infty, t_r)$. In this case, by the definition of $\Delta'(t)$ in (6.57), $\Delta'(t) = 0$. Therefore, by Lemma 6.11, $\phi < \Delta'(t) = 0$. By the definition of $\Delta(t)$ in (6.51), for $t \in (-\infty, t_r)$, $\Delta(t) = \lambda$. Thus, the derivative of $\Delta(t)$ at t is $\Delta'(t) = 0$.

Case 2: $t \in [t_r, t_e)$. In this case, by the definition of $\Delta'(t)$ in (6.57), $\Delta'(t) = \phi$. Therefore, by Lemma 6.11, $\Delta'(t) = \phi < 0$. By the definition of $\Delta(t)$ in (6.51), for $t \in [t_r, t_e)$, $\Delta(t) = \Delta^{\ell}(t)$. Therefore, by the definition of $\Delta^{\ell}(t)$ in (6.45), if $t \ne t_r$, then the derivative of $\Delta(t)$ at t is $\phi = \Delta'(t)$.

Case 3: $t \in [t_e, \infty)$. In this case, by the definition of $\Delta'(t)$ in (6.57), $\Delta'(t) = \Delta^{e\prime}(t)$. By Lemma 6.22, $\phi \le \Delta^{e\prime}(t) \le 0$. Therefore, $\phi \le \Delta'(t) \le 0$. By the definition of $\Delta(t)$ in (6.51), for $t \in [t_e, \infty)$, $\Delta(t) = \Delta^{e}(t)$. Therefore, by the definition of $\Delta^{e\prime}(t)$ as the derivative of $\Delta^{e}(t)$, if $t \ne t_e$, then the derivative of $\Delta(t)$ at t is $\Delta^{e\prime}(t) = \Delta'(t)$. □

We can now provide bounds on the value of $\Delta(t_1)$ relative to $\Delta(t_0)$ for arbitrary $t_0 \le t_1$, as required for the proof of Lemma 6.25 and several later lemmas.

Lemma 6.24. *For arbitrary $t_0 \le t_1$,*

$$\Delta(t_0) \ge \Delta(t_1) \ge \Delta(t_0) + \phi \cdot (t_1 - t_0).$$

Proof. We have

$$\begin{aligned}
\Delta(t_1) = {} & \{\text{By the FTC with } f(t) = \Delta(t)\text{, and by Lemmas 6.21 and 6.23}\} \\
& \Delta(t_0) + \int_{t_0}^{t_1} \Delta'(t)\, dt \\
\le {} & \{\text{By Lemma 6.23}\} \\
& \Delta(t_0) + \int_{t_0}^{t_1} 0\, dt \\
= {} & \{\text{Simplifying}\} \\
& \Delta(t_0).
\end{aligned}$$

Also,

$$\begin{aligned}
\Delta(t_1) &= \{\text{By the FTC with } f(t) = \Delta(t)\text{, and by Lemmas 6.21 and 6.23}\}\\
&\quad \Delta(t_0) + \int_{t_0}^{t_1} \Delta'(t)\, dt\\
&\geq \{\text{By Lemma 6.23}\}\\
&\quad \Delta(t_0) + \int_{t_0}^{t_1} \phi\, dt\\
&= \{\text{Rearranging}\}\\
&\quad \Delta(t_0) + \phi \cdot (t_1 - t_0).
\end{aligned}$$

□

We now provide the lemma that addresses Case C in Figure 6.8.

Lemma 6.25. *If $t_a \in [t_r, t_n)$, $t_a \in (y_{i,k}, y_{i,k+1})$ for some k, and (6.52) holds, then $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient.*

Proof. Observe that $\tau_{i,k}$ is the last job of τ_i that has a PP prior to t_a. We consider two cases, depending on the location of $y_{i,k}$.

Case 1: $y_{i,k} < t_r$. Let $\dot{x}_i(t_r)$ be the value used in the definition of λ in (6.40). We have

$$\begin{aligned}
t_{i,k}^c &\leq \{\text{By the definition of } \dot{x}\text{-sufficient in Definition 6.16}\}\\
&\quad t_r + \dot{x}_i(t_r) + e_{i,k}\\
&\leq \{\text{Rearranging}\}\\
&\quad t_r + x_i^s(s_r) + \dot{x}_i(t_r) - x_i^s(s_r) + e_{i,k}\\
&\leq \{\text{Because } A_i^{rn}(m - L_i - 1) > 0 \text{ by Lemma 6.10}\}\\
&\quad t_r + x_i^s(s_r) + \dot{x}_i(t_r) - x_i^s(s_r) + A_i^{rn}(m - L_i - 1) + e_{i,k}\\
&\leq \{\text{By the definition of } \lambda \text{ in (6.40)}\}\\
&\quad t_r + x_i^s(s_r) + \lambda + e_{i,k}\\
&= \{\text{By Lemma 6.20}\}\\
&\quad t_r + x_i^s(s_r) + \Delta(t_r) + e_{i,k}\\
&= \{\text{Rearranging}\}
\end{aligned}$$

$$
\begin{aligned}
&t_a + x_i^s(s_r) + \Delta(t_r) - (t_a - t_r) + e_{i,k} \\
< &\{\text{By Lemma 6.11}\} \\
&t_a + x_i^s(s_r) + \Delta(t_r) + \phi \cdot (t_a - t_r) + e_{i,k} \\
\leq &\{\text{By Lemma 6.24 with } t_0 = t_r \text{ and } t_1 = t_a\} \\
&t_a + x_i^s(s_r) + \Delta(t_a) + e_{i,k}.
\end{aligned}
$$

Therefore, by the definition of x-sufficient in Definition 6.8, $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient.

Case 2: $y_{i,k} \geq t_r$**.** We have

$$
\begin{aligned}
x_i^s(s_r) + \Delta(t_a) \geq &\{\text{By Lemma 6.24 with } t_0 = y_{i,k} \text{ and } t_1 = t_a\text{; recall that } t_a > y_{i,k} \text{ by the statement of} \\
&\text{the lemma}\} \\
&x_i^s(s_r) + \Delta(y_{i,k}) + \phi \cdot (t_a - y_{i,k}) \\
\geq &\{\text{By Lemma 6.11}\} \\
&x_i^s(s_r) + \Delta(y_{i,k}) - (t_a - y_{i,k}). \qquad (6.58)
\end{aligned}
$$

By (6.58) and Lemma 6.17,

$$
x_i^s(s_r) + \Delta(t_a) \geq \max\{0,\ x_i^s(s_r) + \Delta(y_{i,k}) - (t_a - y_{i,k})\}. \qquad (6.59)
$$

By (6.52) with $j = i$ and $\ell = k$, $x_i(y_{i,k}) = x_i^s(s_r) + \Delta(y_{i,k})$ is x-sufficient. Thus, by Theorem 6.3, $x_i(t_a) = \max\{0,\ x_i^s(s_r) + \Delta(y_{i,k}) - (t_a - y_{i,k})\}$ is x-sufficient. Therefore, by (6.59) and Property 6.4 with $c_0 = \max\{0,\ x_i^s(s_r) + \Delta(y_{i,k}) - (t_a - y_{i,k})\}$ and $c_1 = x_i^s(s_r) + \Delta(t_a)$, $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient. □

We now consider Case D in Figure 6.8, where $t_a = y_{i,k}$ for some k, $t_{i,k}^c > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is f-dominant for L_i. In this case, by the definition of f-dominant for L_i in Definition 6.10, $k > 0$, and thus $\tau_{i,k-1}$ exists. Because we are considering $y_{i,k} = t_a \in [t_r, t_n)$, $y_{i,k-1} < y_{i,k} < t_n$. Therefore, $y_{i,k-1} \in [0, t_n)$. In Lemma 6.27, we consider the case that $y_{i,k-1} \in [0, t_r)$, and in Lemma 6.30, we consider the case that $y_{i,k-1} \in [t_r, t_n)$. In either case, we will use Lemma 6.10 within our proofs, which requires (6.30) and (6.31) as preconditions. We first provide a short lemma that provides inequalities that match both preconditions. The same lemma is also used when analyzing $[t_n, \infty)$.

Lemma 6.26. *If $t_a \in [t_r, \infty)$, $t_a = y_{i,k}$ for some $\tau_{i,k}$, $t^c_{i,k} > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is f-dominant for L_i, then $k > 0$ and $t^c_{i,k-1} > t_r$.*

Proof. By the definition of f-dominant for L in Definition 6.10, $k > 0$. Furthermore, we also have

$$
\begin{aligned}
t^c_{i,k-1} &> \{\text{By the definition of f-dominant for } L \text{ in Definition 6.10}\}\\
&\quad y_{i,k}\\
&= \{\text{By the statement of the lemma}\}\\
&\quad t_a\\
&\geq \{\text{By the statement of the lemma}\}\\
&\quad t_r.
\end{aligned}
$$

□

Lemma 6.27. *If $t_a \in [t_r, t_n)$, $t_a = y_{i,k}$ for some $\tau_{i,k}$, $t^c_{i,k} > y_{i,k} + e_{i,k}$, $\tau_{i,k}$ is f-dominant for L_i, and $y_{i,k-1} \in [0, t_r)$, then $x_i(t_a) = x^s_i(s_r) + \Delta(t_a)$ is x-sufficient.*

Proof. By Lemma 6.26, which has only preconditions implied by those of this lemma, the following facts hold.

$$k > 0 \tag{6.60}$$

$$t^c_{i,k-1} > t_r \tag{6.61}$$

Thus, we have

$$
\begin{aligned}
&x^s_i(s_r) + \Delta(t_a)\\
&= \{\text{Because } t_a = y_{i,k}\}\\
&\quad x^s_i(s_r) + \Delta(y_{i,k})\\
&\geq \{\text{By Lemma 6.24 with } t_0 = t_r \text{ and } t_1 = y_{i,k}\}\\
&\quad x^s_i(s_r) + \Delta(t_r) + \phi \cdot (y_{i,k} - t_r)\\
&= \{\text{By Lemma 6.20}\}\\
&\quad x^s_i(s_r) + \lambda + \phi \cdot (y_{i,k} - t_r)
\end{aligned}
$$

$$\geq \{\text{By the definition of } \lambda \text{ in (6.40)}\}$$

$$x_i^s(s_r) + \dot{x}_i(t_r) - x_i^s(s_r) + A_i^{rn}(m - L_i - 1) + \phi \cdot (y_{i,k} - t_r)$$

$$= \{\text{Rearranging}\}$$

$$t_r + \dot{x}_i(t_r) + e_{i,k-1} + A_i^{rn}(m - L_i - 1) - e_{i,k-1} - t_r + \phi \cdot (y_{i,k} - t_r)$$

$$\geq \{\text{By the definition of } \dot{x}\text{-sufficient in Definition 6.16, because } y_{i,k-1} < t_r\}$$

$$t^c_{i,k-1} + A_i^{rn}(m - L_i - 1) - e_{i,k-1} - t_r + \phi \cdot (y_{i,k} - t_r)$$

$$\geq \{\text{By the definition of } \phi \text{ in (6.46)}\}$$

$$t^c_{i,k-1} + A_i^{rn}(m - L_i - 1) - e_{i,k-1} - t_r + \left(\frac{s_r \cdot A_i^{rn}(m - L_i - 1)}{T_i} - 1\right) \cdot (y_{i,k} - t_r)$$

$$= \{\text{Rearranging}\}$$

$$t^c_{i,k-1} - y_{i,k} + A_i^{rn}(m - L_i - 1) - e_{i,k-1} + \frac{s_r \cdot A_i^{rn}(m - L_i - 1)}{T_i} \cdot (y_{i,k} - t_r)$$

$$\geq \{\text{Because } s_r \geq 0,\, A_i^{rn}(m - L_i - 1) > 0 \text{ by Lemma 6.10, } T_i > 0, \text{ and } y_{i,k} \geq t_r\}$$

$$t^c_{i,k-1} - y_{i,k} + A_i^{rn}(m - L_i - 1) - e_{i,k-1}$$

$$\geq \{\text{By (6.61) and Property 6.11}\}$$

$$t^c_{i,k-1} - y_{i,k} + A_i^{rn}(m - L_i - 1) - C_i$$

$$\geq \{\text{By Lemma 6.10 with (6.60) and (6.61), which match (6.30) and (6.31)}\}$$

$$t^c_{i,k-1} - y_{i,k} + A_{i,k}(m - L_i - 1) - e_{i,k}$$

$$= \{\text{By the definition of } x^f_{i,k} \text{ in (6.15)}\}$$

$$x^f_{i,k}. \tag{6.62}$$

The preconditions of the lemma establish the preconditions of Theorem 6.4: $t_a = y_{i,k}$ for some k, $t^c_{i,k} > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is f-dominant for L_i. Therefore, by (6.62) and Theorem 6.4, $x_i(t_a) = x^f_{i,k}$ is x-sufficient. Therefore, by (6.62) and Property 6.4 with $c_0 = x^f_{i,k}$ and $c_1 = x_i^s(s_r) + \Delta(t_a)$, the lemma holds. □

We next consider, in Lemma 6.30, the case that $y_{i,k-1} \in [t_r, t_n)$. We will explicitly consider the difference between $y_{i,k-1}$ and $y_{i,k}$, based on the following lemma, which will also be used in Section 6.3.5.

Lemma 6.28. *For $k > 0$, $v(y_{i,k}) \geq v(y_{i,k-1}) + T_i$.*

Proof. We have

$$\begin{aligned}
v(y_{i,k}) &= \{\text{By the definition of } Y_i \text{ in (6.6)}\} \\
&\quad v(r_{i,k}) + Y_i \\
&\geq \{\text{By the definition of } T_i \text{ in (6.5)}\} \\
&\quad v(r_{i,k-1}) + T_i + Y_i \\
&= \{\text{By the definition of } Y_i \text{ in (6.6)}\} \\
&\quad v(y_{i,k-1}) + T_i. \qquad \square
\end{aligned}$$

We now consider Case D when $y_{i,k-1} \in [t_r, t_n)$. Here, as in several other places in this section, we will need to translate between differences in virtual time and differences in actual time. When $t_0 \in [t_s, t_n]$ and $t_1 \in [t_s, t_n]$, an exact result can be used.

Lemma 6.29. *If $t_0 \in [t_s, t_n]$ and $t_1 \in [t_s, t_n]$, then $v(t_1) - v(t_0) = s_r \cdot (t_1 - t_0)$.*

Proof. We have

$$\begin{aligned}
v(t_1) - v(t_0) &= \{\text{By Property 6.2}\} \\
&\quad \int_{t_0}^{t_1} s(t)\,dt \\
&= \{\text{By Property 6.6}\} \\
&\quad \int_{t_0}^{t_1} s_r\,dt \\
&= \{\text{Simplifying}\} \\
&\quad s_r \cdot (t_1 - t_0). \qquad \square
\end{aligned}$$

With that result, we can finish our analysis of Case D.

Lemma 6.30. *If $t_a \in [t_r, t_n)$, $t_a = y_{i,k}$ for some $\tau_{i,k}$, $t^c_{i,k} > y_{i,k} + e_{i,k}$, $\tau_{i,k}$ is f-dominant for L_i, $y_{i,k-1} \in [t_r, t_n)$, and (6.52) holds, then $x_i(t_a) = x^s_i(s_r) + \Delta(t_a)$ is x-sufficient.*

Proof. In Lemma 6.28 we considered the difference between $y_{i,k}$ and $y_{i,k-1}$ in *virtual* time. We now consider their difference in *actual* time.

$$\begin{aligned}
(y_{i,k} - y_{i,k-1}) \cdot s_r = {} & \{\text{By Lemma 6.29, because } t_r \geq t_s \text{ by the definition of } t_r \text{ in Definition 6.13}\} \\
& v(y_{i,k}) - v(y_{i,k-1}) \\
\geq {} & \{\text{By Lemma 6.28; } k > 0 \text{ by the definition of f-dominant for } L_i \text{ in Definition 6.10}\} \\
& v(y_{i,k-1}) + T_i - v(y_{i,k-1}) \\
= {} & \{\text{Simplifying}\} \\
& T_i.
\end{aligned}$$

Rearranging,

$$y_{i,k-1} \leq y_{i,k} - \frac{T_i}{s_r}. \tag{6.63}$$

We now demonstrate the following claim.

Claim 6.30.1. *$x_i\left(y_{i,k} - \frac{T_i}{s_r}\right) = x_i^s(s_r) + \Delta\left(y_{i,k} - \frac{T_i}{s_r}\right)$ is x-sufficient.*

We have that $y_{i,k-1} \in [t_r, t_n)$ by the statement of the lemma. Also,

$$\begin{aligned}
y_{i,k-1} < {} & \{\text{By our task model}\} \\
& y_{i,k} \\
= {} & \{\text{By the statement of the lemma}\} \\
& t_a.
\end{aligned}$$

Thus, by (6.52) with $j = i$ and $\ell = k-1$, we have that $x_i(y_{i,k-1}) = x_i^s(s_r) + \Delta(y_{i,k-1})$ is x-sufficient. If $y_{i,k-1} = y_{i,k} - \frac{T_i}{s_r}$, then Claim 6.30.1 follows immediately. Otherwise, by (6.63), $y_{i,k} - \frac{T_i}{s_r} \in (y_{i,k-1}, y_{i,k})$. Thus, because $y_{i,k-1} \in [t_r, t_n)$ and $y_{i,k} \in [t_r, t_n)$ by the statement of the lemma, $y_{i,k} - \frac{T_i}{s_r} \in [t_r, t_n)$. Therefore, by Lemma 6.25 with k shifted earlier by 1 and $t_a = y_{i,k} - \frac{T_i}{s_r}$, Claim 6.30.1 holds.

Continuing the proof of Lemma 6.30, by Lemma 6.26 (which has only preconditions that are implied by those of this lemma), the following facts hold.

$$k > 0 \tag{6.64}$$

$$t^c_{i,k-1} > t_r \tag{6.65}$$

Thus, we have

$$
\begin{aligned}
& x^s_i(s_r) + \Delta(t_a) \\
= \; & \{\text{Because } t_a = y_{i,k}\} \\
& x^s_i(s_r) + \Delta(y_{i,k}) \\
\geq \; & \{\text{By Lemma 6.24 with } t_0 = y_{i,k} - \tfrac{T_i}{s_r} \text{ and } t_1 = y_{i,k}\} \\
& x^s_i(s_r) + \Delta\left(y_{i,k} - \frac{T_i}{s_r}\right) + \phi \cdot \frac{T_i}{s_r} \\
= \; & \{\text{Rewriting}\} \\
& y_{i,k} - \frac{T_i}{s_r} + x^s_i(s_r) + \Delta\left(y_{i,k} - \frac{T_i}{s_r}\right) + e_{i,k-1} - y_{i,k} + \frac{T_i}{s_r} + \phi \cdot \frac{T_i}{s_r} - e_{i,k-1} \\
\geq \; & \{\text{By Claim 6.30.1 and the definition of } x\text{-sufficient in Definition 6.8 with } t = y_{i,k} - \tfrac{T_i}{s_r}\text{, and by (6.63)}\} \\
& t^c_{i,k-1} - y_{i,k} + \frac{T_i}{s_r} + \phi \cdot \frac{T_i}{s_r} - e_{i,k-1} \\
\geq \; & \{\text{By the definition of } \phi \text{ in (6.46)}\} \\
& t^c_{i,k-1} - y_{i,k} + \frac{T_i}{s_r} + \left(\frac{s_r \cdot A^{rn}_i(m - L_i - 1)}{T_i} - 1\right) \cdot \frac{T_i}{s_r} - e_{i,k-1} \\
= \; & \{\text{Simplifying}\} \\
& t^c_{i,k-1} - y_{i,k} + A^{rn}_i(m - L_i - 1) - e_{i,k-1} \\
\geq \; & \{\text{By (6.65) and Property 6.11}\} \\
& t^c_{i,k-1} - y_{i,k} + A^{rn}_i(m - L_i - 1) - C_i \\
\geq \; & \{\text{By Lemma 6.10 with (6.64) and (6.65), which match (6.30) and (6.31)}\} \\
& t^c_{i,k-1} - y_{i,k} + A_{i,k}(m - L_i - 1) - e_{i,k} \\
= \; & \{\text{By the definition of } x^f_{i,k} \text{ in (6.15)}\}
\end{aligned}
$$

$$x_{i,k}^{f}. \tag{6.66}$$

The preconditions of the lemma establish the preconditions of Theorem 6.4: $t_a = y_{i,k}$ for some k, $t_{i,k}^{c} > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is f-dominant for L_i. Thus, by Theorem 6.4, $x_i(t_a) = x_{i,k}^{f}$ is x-sufficient. Therefore, by (6.66) and Property 6.4 with $c_0 = x_{i,k}^{f}$ and $c_1 = x_i^s(s_r) + \Delta(t_a)$, the lemma holds. □

We now provide a combined lemma that addresses Case D.

Lemma 6.31. *If $t_a \in [t_r, t_n)$, $t_a = y_{i,k}$ for some $\tau_{i,k}$, $t_{i,k}^{c} > y_{i,k} + e_{i,k}$, $\tau_{i,k}$ is f-dominant for L_i, and* (6.52) *holds, then $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient.*

Proof. If $y_{i,k-1} \in [0, t_r)$, then the lemma follows from Lemma 6.27. Otherwise, it follows from Lemma 6.30.

□

We now address Case E in Figure 6.8, where $t_a = y_{i,k}$ for some k, $t_{i,k}^{c} > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is m-dominant for L_i. In this case, by Theorem 6.5, $x_{i,k}^{m}$ is x-sufficient. Observe that $e_{i,k}^{p}$, the total amount of work remaining at $y_{i,k}$ from jobs of τ_i prior to $\tau_{i,k}$, appears in the expression for $x_{i,k}^{m}$ in (6.24). Thus, we must bound $e_{i,k}^{p}$. In order to do so, we will use the result of Lemma 6.3 with $t_0 = y_{i,k}$.

Observe in Lemma 6.3 the presence of $D_j^e(b_{j,\ell}, y_{i,k})$, and in the definition of $D_j^e(b_{j,\ell}, y_{i,k})$ in Definition 6.6 the presence of $e_{i,k}$. Although when accounting for overloads it was necessary to account for specific parameters of specific jobs, we want to develop general dissipation bounds that do not require such parameters. We will eliminate all such job references by deriving upper bounds using Properties 6.6–6.12 and inductively establishing (6.52).

Similarly, observe that the definition of $W_{i,k}$ in (6.17) involves $D_j^e(b_{j,\ell}, y_{i,k})$ and $D_j^e(t_{i,k}^{b}, y_{i,k})$. Therefore, many of the same lemmas used when analyzing $e_{i,k}^{p}$ will be additionally used when analyzing $W_{i,k}$.

We first provide a general upper bound to $D_j^e(t_0, t_1)$ (defined in Definition 6.6) when Property 6.11 applies.

Definition 6.18.

$$D_i^C(t_0, t_1) = \sum_{\tau_{i,k} \in \omega} C_i,$$

where ω is the set of jobs with $t_0 \le r_{i,k} \le y_{i,k} \le t_1$.

Observe that the definition of $D_i^C(t_0,t_1)$ in Definition 6.18 differs from that of $D_i^e(t_0,t_1)$ in Definition 6.6 only in that it uses C_i in place of $e_{i,k}$. Lemma 6.32 below justifies this definition. However, in some cases, the replacement of some $e_{i,\ell}$ with C_i creates excessive pessimism. Thus, Lemma 6.32 also provides a version of the bound that eliminates this pessimism for one particular job.

Lemma 6.32. *If*

$$\text{For all jobs } \tau_{i,k} \text{ with } r_{i,k} \geq t_0,\, t_{i,k}^c \geq t_r, \tag{6.67}$$

then

$$D_i^e(t_0,t_1) \leq D_i^C(t_0,t_1).$$

If, furthermore, there is a $\tau_{i,\ell}$ such that $t_0 \leq r_{i,\ell} \leq y_{i,\ell} \leq t_1$, then

$$D_i^e(t_0,t_1) \leq D_i^C(t_0,t_1) + e_{i,\ell} - C_i.$$

Proof. The definitions of ω in Definition 6.6 and in Definition 6.18 are identical: ω is the set of jobs $\tau_{i,k}$ with $t_0 \leq r_{i,k} \leq y_{i,k} \leq t_1$. By (6.67), $t_{i,k}^c \geq t_r$ for each such $\tau_{i,k}$. Thus, by Property 6.11, $e_{i,k} \leq C_i$ for all $\tau_{i,k} \in \omega$. Therefore,

$$\begin{aligned}
D_i^e(t_0,t_1) &= \{\text{By the definition of } D_i^e(t_0,t_1) \text{ in Definition 6.6}\} \\
&\quad \sum_{\tau_{i,k}\in\omega} e_{i,k} \\
&\leq \{\text{Because } e_{i,k} \leq C_i \text{ for all } \tau_{i,k} \in \omega\} \\
&\quad \sum_{\tau_{i,k}\in\omega} C_i \\
&= \{\text{By the definition of } D_i^C(t_0,t_1) \text{ in Definition 6.18}\} \\
&\quad D_i^C(t_0,t_1).
\end{aligned}$$

If, furthermore, there is a $\tau_{i,\ell}$ such that $t_0 \leq r_{i,\ell} \leq y_{i,\ell} \leq t_1$, then $\tau_{i,\ell} \in \omega$ by the definition of ω in Definitions 6.6 and 6.18. Therefore,

$$D_i^e(t_0,t_1) = \{\text{By the definition of } D_i^e(t_0,t_1) \text{ in Definition 6.6}\}$$

$$
\begin{aligned}
&\sum_{\tau_{i,k}\in\omega} e_{i,k}\\
&= \{\text{Rearranging}\}\\
&\quad \sum_{\tau_{i,k}\in\omega} C_i + \sum_{\tau_{i,k}\in\omega\setminus\{\tau_{i,\ell}\}} (e_{i,k}-C_i) \;+\; (e_{i,\ell}-C_i)\\
&\leq \{\text{Because } e_{i,k}\leq C_i \text{ for } \tau_{i,k}\in\omega\}\\
&\quad \sum_{\tau_{i,k}\in\omega} C_i + (e_{i,\ell}-C_i)\\
&= \{\text{By the definition of } D_i^C(t_0,t_1) \text{ in Definition 6.18}\}\\
&\quad D_i^C(t_0,t_1) + e_{i,\ell} - C_i.
\end{aligned}
$$

□

Either of the next two lemmas can be used to establish that (6.67) holds for a given t_0.

Lemma 6.33. *If $t_0 = b_{i,\ell}$ for some $\tau_{i,\ell}$ pending at some time $t_2 \geq t_r$, then* (6.67) *holds.*

Proof. By (6.5) and the definition of $b_{i,\ell}$ in Definition 6.5, for any $\tau_{i,k}$ with $r_{i,k} \geq t_0$, $k > \ell$. Therefore,

$$
\begin{aligned}
t_{i,k}^c &> \{\text{By our task model}\}\\
&\quad t_{i,\ell}^c\\
&\geq \{\text{By the definition of ``pending'' in Definition 6.4}\}\\
&\quad t_2\\
&\geq \{\text{By the statement of the lemma}\}\\
&\quad t_r.
\end{aligned}
$$

□

Lemma 6.34. *If $t_0 \geq t_r$, then* (6.67) *holds.*

Proof. For arbitrary $\tau_{i,k}$ with $r_{i,k} \geq t_0$, we have

$$
\begin{aligned}
t_{i,k}^c &\geq \{\text{Because jobs cannot commence execution before their release times}\}\\
&\quad r_{i,k}\\
&\geq \{\text{By the case we are considering}\}\\
&\quad t_0
\end{aligned}
$$

$$\begin{aligned} &\geq \{\text{By the statement of the lemma}\} \\ &\quad t_r. \end{aligned}$$ □

We now provide a general upper bound on $D_i^C(t_0, t_1)$ that will be used in conjunction with Lemma 6.32. This upper bound uses a result from Chapter 3.

Lemma 6.35. *If $t_0 \leq t_1$, then $D_j^C(t_0, t_1) \leq U_j^v \cdot (v(t_1) - v(t_0)) + S_j$.*

Proof. When considering virtual time instead of actual time for the purpose of job separation and PPs, and given the use of C_i in Definition 6.18 in place of $e_{i,k}$ in Definition 6.6, GEL-v scheduling under the SVO model reduces to traditional GEL scheduling under the ordinary sporadic task model. Thus, after translating t_0 and t_1 from actual time to virtual time, the lemma is identical to Lemma 3.2. □

In Lemma 6.3 and in the definition of $W_{i,k}$ in (6.17), $D_j^e(b_{j,\ell}, y_{i,k})$ appears for some $\tau_{j,\ell}$. Using Lemma 6.32, this term can be upper bounded using $D_j^C(b_{j,\ell}, y_{i,k})$ and possibly an extra term to reduce pessimism. We more specifically characterize $D_j^C(b_{j,\ell}, y_{i,k})$ in the following lemma. Using (6.52) and the definition of x-sufficient in Definition 6.8, we will be able to reason about $y_{j,\ell}$. Therefore, on the right hand side in Lemma 6.36 we use $y_{j,\ell}$ instead of $b_{j,\ell}$. As we show in the lemma, the analysis required to do so effectively cancels out the S_j term that would appear using Lemma 6.35. To facilitate later use of Lemma 6.36, we use t_3 in place of $y_{i,k}$.

Lemma 6.36.

$$D_j^C(b_{j,\ell}, t_3) \leq \max\{0, U_j^v \cdot (v(t_3) - v(y_{j,\ell}))\}.$$

Proof. We consider two cases, depending on the relative values of $b_{j,\ell}$ and t_3.

Case 1: $b_{j,\ell} > t_3$. If $b_{j,\ell} > t_3$, then there are no jobs $\tau_{j,v}$ such that $b_{j,\ell} \leq r_{j,v} \leq y_{j,v} \leq t_3$. Therefore, by the definition of $D_j^C(b_{j,\ell}, t_3)$ in Definition 6.18, $D_j^C(b_{j,\ell}, t_3) = 0 \leq \max\{0, U_j^v \cdot (v(t_3) - v(y_{j,\ell}))\}$.

Case 2: $b_{j,\ell} \leq t_3$. We first relate $v(b_{j,\ell})$ to $v(y_{j,\ell})$.

$$\begin{aligned} v(b_{j,\ell}) &= \{\text{By the definition of } b_{j,\ell} \text{ in Definition 6.5}\} \\ &\quad v(r_{j,\ell}) + T_j \\ &= \{\text{Rearranging}\} \end{aligned}$$

$$\begin{aligned}
&v(r_{j,\ell})+Y_j+T_j-Y_j\\
=&\{\text{By (6.6)}\}\\
&v(y_{j,\ell})+T_j-Y_j. \qquad (6.68)
\end{aligned}$$

We have

$$\begin{aligned}
D_j^C(b_{j,\ell},t_3) \leq &\{\text{By Lemma 6.35 with } t_0=b_{i,\ell} \text{ and } t_1=t_3\text{; } b_{i,\ell}\leq t_3 \text{ holds by the case we are considering}\}\\
&U_j^v\cdot(v(t_3)-v(b_{i,\ell}))+S_j\\
=&\{\text{By (6.68)}\}\\
&U_j^v\cdot(v(t_3)-v(y_{j,\ell})-T_j+Y_j)+S_j\\
=&\{\text{Rearranging}\}\\
&U_j^v\cdot(v(t_3)-v(y_{j,\ell}))-U_j^v\cdot(T_j-Y_j)+S_j\\
=&\{\text{By the definition of } U_j^v \text{ in (6.27) and the definition of } S_j \text{ in (6.38)}\}\\
&U_j^v\cdot(v(t_3)-v(y_{j,\ell}))-C_j\cdot\left(1-\frac{Y_j}{T_j}\right)+C_j\cdot\left(1-\frac{Y_j}{T_j}\right)\\
=&\{\text{Cancelling}\}\\
&U_j^v\cdot(v(t_3)-v(y_{j,\ell}))\\
\leq&\{\text{By the definition of “max”}\}\\
&\max\{0,U_j^v\cdot(v(t_3)-v(y_{j,\ell}))\}. \qquad \square
\end{aligned}$$

In several of our proofs, we will require a lower bound on $y_{j,\ell}$. In a similar manner to how we defined a notion of x-sufficient for a value of the function $x_i(t)$ for a particular i and t, we define a notion of *x^p-sufficient* for a function $x_i^p(t)$ (*p*ending). We will show that $x_i^p(t)$ is closely related to $x_i(t)$, hence the similar notation.

Definition 6.19. $x_i^p(t)$ is *x^p-sufficient* if $x_i^p(t)\geq 0$ and, for all $\tau_{i,k}$ pending at t,

$$y_{i,k}\geq t-(x_i^p(t)+e_{i,k}^c(t)).$$

In Lemma 6.38 below, we will provide a specific x^p-sufficient choice of $x_i^p(t_2)$ for an arbitrary τ_i and $t_2\in[t_r,t_n)$. (We use t_2 in place of t_0 to avoid later conflicts in notation.) That choice will be based on the simple observation in the following lemma. Comparing this lemma to Definition 6.19 shows the reason for the similar notation between $x_i(t)$ and $x_i^p(t)$. (A different choice of $x_i^p(t)$, also based on Lemma 6.37, will be used in Section 6.3.5.)

Lemma 6.37. *If $\tau_{j,\ell}$ is pending at t_2 and $x_j\left(y_{j,\ell}\right)$ is x-sufficient, then $y_{j,\ell} \geq t_2 - (x_j\left(y_{j,\ell}\right) + e^c_{j,\ell}(t_2))$.*

Proof. We use proof by contradiction. Suppose that $x_j\left(y_{j,\ell}\right)$ is x-sufficient, but

$$y_{j,\ell} < t_2 - (x_j\left(y_{j,\ell}\right) + e^c_{j,\ell}(t_2)). \tag{6.69}$$

Then,

$$\begin{aligned}
t^c_{j,\ell} \leq\ & \{\text{By Definition 6.8}\} \\
& y_{j,\ell} + x_j\left(y_{j,\ell}\right) + e_{j,\ell} \\
<\ & \{\text{By (6.69)}\} \\
& t_2 - (x_j\left(y_{j,\ell}\right) + e^c_{j,\ell}(t_2)) + x_j\left(y_{j,\ell}\right) + e_{j,\ell} \\
=\ & \{\text{Simplifying, and by Property 6.1}\} \\
& t_2 + e^r_{j,\ell}(t_2).
\end{aligned} \tag{6.70}$$

If $e^r_{j,\ell}(t_2) = 0$, then (6.70) contradicts the assumption that $\tau_{j,\ell}$ is pending at t_2. Otherwise, (6.70) contradicts the definition of $e^r_{j,\ell}(t_2)$ in Definition 6.3. □

We now define an x^p-sufficient choice of $x_i^p(t_2)$ for $t_2 \in [t_r, t_n)$. Observe that, by (6.51), $\Delta(t)$ is defined over all real numbers, so in particular is defined for negative numbers.

Lemma 6.38. *If*

$$t_2 \in [t_r, t_n) \tag{6.71}$$

and

$$\text{For all } \tau_{j,\ell} \text{ with } y_{j,\ell} \in [t_r, t_2),\ x_j\left(y_{j,\ell}\right) = x^s_j(s_r) + \Delta\left(y_{j,\ell}\right) \text{ is } x\text{-sufficient.} \tag{6.72}$$

then $x^p_j(t_2) = x^{p_r}_j(t_2)$ is x^p-sufficient, where

$$x^{p_r}_j(t_2) \triangleq x^s_j(s_r) + \Delta(t_2 - \rho). \tag{6.73}$$

Proof. By the definition of $x_j^{p_r}(t_2)$ in (6.73) and by Lemma 6.17 with $t_a = t_2 - \rho$, $x_j^{p_r}(t_2) \geq 0$. To show the remaining condition for $x_j^p(t_2) = x_j^{p_r}(t_2)$ to be x^p-sufficient, we consider an arbitrary job $\tau_{j,\ell}$ pending at t_2. By showing that $y_{j,\ell} \geq t_2 - (x_j^{p_r}(t_2) + e_{j,\ell}^c(t_2))$ for such an *arbitrary* job, we show that $x_j^p(t_2) = x_j^{p_r}(t_2)$ is x^p-sufficient. We consider three cases, depending on the value of $y_{j,\ell}$.

Case 1: $y_{j,\ell} \in (-\infty, t_r)$**.** We first bound the value of t_2 for this case to apply. Let $\dot{x}_j(t_r)$ be the value used in the definition of of λ in (6.40).

$$
\begin{aligned}
t_2 \leq{} & \{\text{By the definition of "pending" in Definition 6.4}\} \\
& t_{j,\ell}^c \\
\leq{} & \{\text{By the definition of } \dot{x}\text{-sufficient in Definition 6.16}\} \\
& t_r + \dot{x}_j(t_r) + e_{j,\ell} \\
\leq{} & \{\text{By Property 6.11, because } \tau_{j,\ell} \text{ is pending at } t_2 \geq t_r\} \\
& t_r + \dot{x}_j(t_r) + C_j \\
<{} & \{\text{Because } A_j^{rn}(m - L_j - 1) > 0 \text{ by Lemma 6.10}\} \\
& t_r + \dot{x}_j(t_r) + A_j^{rn}(m - L_j - 1) + C_j \\
={} & \{\text{Rearranging}\} \\
& t_r + x_j^s(s_r) + (\dot{x}_j(t_r) - x_j^s(s_r) + A_j^{rn}(m - L_j - 1)) + C_j \\
\leq{} & \{\text{By the definition of } \lambda \text{ in (6.40)}\} \\
& t_r + x_j^s(s_r) + \lambda + C_j \\
\leq{} & \{\text{By the definition of } \rho \text{ in (6.50)}\} \\
& t_r + \rho.
\end{aligned}
$$

Therefore, $t_2 - \rho < t_r$, so by the definition of $\Delta(t_2 - \rho)$ in (6.51),

$$\Delta(t_2 - \rho) = \lambda. \tag{6.74}$$

By the case we are considering, $y_{j,\ell} < t_r$. Furthermore, because $\tau_{j,\ell}$ is pending at $t_2 > t_r$, $t^c_{j,\ell} \in [t_r, \infty)$. Therefore, by the definition of κ in Definition 6.17, $\tau_{j,\ell} \in \kappa$. Let c_0 be the value of $x_j\left(y_{j,\ell}\right)$ used in the definition of λ in (6.40).

We have

$$\begin{aligned}
x_j^{p_r}(t_2) &= \{\text{By the definition of } x_j^{p_r}(t_2) \text{ in (6.73)}\} \\
&\quad x_j^s(s_r) + \Delta(t_2 - \rho) \\
&= \{\text{By (6.74)}\} \\
&\quad x_j^s(s_r) + \lambda \\
&\geq \{\text{By the definition of } \lambda \text{ in (6.40)}\} \\
&\quad x_j^s(s_r) + (c_0 - x_j^s(s_r)) \\
&= \{\text{Cancelling}\} \\
&\quad c_0.
\end{aligned}$$

Therefore, by Property 6.4 with $c_1 = x_j^{p_r}(t_2)$, $x_j\left(y_{j,\ell}\right) = x_j^{p_r}(t_2)$ is x-sufficient. Therefore, by Lemma 6.37, $y_{j,\ell} \geq t_2 - (x_j^{p_r}(t_2) + e^c_{j,\ell}(t_2))$.

Case 2: $y_{j,\ell} \in [t_r, t_2)$**.** As in the previous case, we again bound the value of t_2 in order for this case to apply.

$$\begin{aligned}
t_2 &\leq \{\text{By the definition of ``pending'' in Definition 6.4}\} \\
&\quad t^c_{j,\ell} \\
&\leq \{\text{By (6.72) and the definition of } x\text{-sufficient in Definition 6.8}\} \\
&\quad y_{j,\ell} + x_j^s(s_r) + \Delta\left(y_{j,\ell}\right) + e_{j,\ell} \\
&\leq \{\text{By Property 6.11, because } \tau_{j,\ell} \text{ is pending at } t_2 \geq t_r\} \\
&\quad y_{j,\ell} + x_j^s(s_r) + \Delta\left(y_{j,\ell}\right) + C_j \\
&\leq \{\text{By Lemma 6.24 with } t_0 = t_r \text{ and } t_1 = y_{j,\ell}\} \\
&\quad y_{j,\ell} + x_j^s(s_r) + \Delta(t_r) + C_j \\
&= \{\text{By Lemma 6.20}\} \\
&\quad y_{j,\ell} + x_j^s(s_r) + \lambda + C_j
\end{aligned}$$

$$
\begin{aligned}
&\leq \{\text{By the definition of } \rho \text{ in (6.50)}\}\\
&\quad y_{j,\ell}+\rho.
\end{aligned}
$$

Rearranging,

$$t_2-\rho\leq y_{j,\ell}. \tag{6.75}$$

Thus, we have

$$
\begin{aligned}
x_j^{p_r}(t_2) &= \{\text{By the definition of } x_j^{p_r}(t_2) \text{ in (6.73)}\}\\
&\quad x_j^s(s_r)+\Delta(t_2-\rho)\\
&\geq \{\text{By Lemma 6.24 with } t_0=t_2-\rho \text{ and } t_1=y_{j,\ell}, \text{ and by (6.75)}\}\\
&\quad x_j^s(s_r)+\Delta\left(y_{j,\ell}\right). \qquad (6.76)
\end{aligned}
$$

Therefore, by (6.72), (6.76), and Property 6.4 with $c_0=x_j^s(s_r)+\Delta\left(y_{j,\ell}\right)$ and $c_1=x_j^{p_r}(t_2)$, $x_j\left(y_{j,\ell}\right)=x_j^{p_r}(t_2)$ is x-sufficient. Therefore, by Lemma 6.37, $y_{j,\ell}\geq t_2-(x_j^{p_r}(t_2)+e_{j,\ell}^c(t_2))$.

Case 3: $t_2\leq y_{j,\ell}$. In this case,

$$
\begin{aligned}
y_{j,\ell} &\geq t_2\\
&\geq \{\text{By Lemma 6.17 with } t_a=t_2-\rho, \text{ and because } e_{j,\ell}^c(t_2)\geq 0\}\\
&\quad t_2-(x_j^s(t_r)+\Delta(t_2-\rho)+e_{j,\ell}^c(t_2))\\
&= \{\text{By the definition of } x_j^{p_r}(t_2) \text{ in (6.73)}\}\\
&\quad t_2-(x_j^{p_r}(t_2)+e_{j,\ell}^c(t_2)).
\end{aligned}
$$

□

In Lemma 6.40 below, we will bound an expression that corresponds to one in Lemma 6.3 and in the definition of $W_{i,k}$ in (6.17), by combining the results of Lemmas 6.32 and 6.36. Observe in Lemma 6.36 that virtual times are used, in the form of $v(y_{j,\ell})$ and $v(t_3)$. However, in Lemma 6.32, actual times are used. In order to combine the results of these two lemmas, we will need to characterize as Lemma 6.39 the behavior of $v(t)$, using Property 6.2. Unlike in Lemma 6.29, we consider arbitrary $t_1\geq t_0$.

If we were only concerned with the analysis when $t_0 \in [t_s, t_n]$ and $t_1 \in [t_s, t_n]$, then we could simply use Lemma 6.29. However, Lemma 6.40 is general enough to be used in Section 6.3.5 in analysis that involves $[t_n, \infty)$, during which $s(t) = 1$. Therefore, we instead define as s_{ub} (*u*pper *b*ound) an upper bound on $s(t)$ for $t \in [t_0, t_1)$. In this section, we will use $s_{ub} = s_r$ for the just-noted reason, and in Section 6.3.5 we will use $s_{ub} = 1$, which is always valid by the definition of $s(t)$.

Lemma 6.39. *If $t_1 \geq t_0$ and $s(t) \leq s_{ub}$ for $t \in [t_0, t_1)$, then $v(t_1) - v(t_0) \leq s_{ub} \cdot (t_1 - t_0)$.*

Proof. We have

$$\begin{aligned}
v(t_1) - v(t_0) &= \{\text{By Property 6.2}\} \\
&\quad \int_{t_0}^{t_1} s(t)\, dt \\
&\leq \{\text{By the statement of the lemma}\} \\
&\quad \int_{t_0}^{t_1} s_{ub}\, dt \\
&= \{\text{Rearranging}\} \\
&\quad s_{ub} \cdot (t_1 - t_0). \qquad \square
\end{aligned}$$

We now provide a result in Lemma 6.40 that allows us to upper bound expressions that appear in Lemma 6.3 and in the definition of $W_{i,k}$ in (6.17). We use the general $x_j^p(t_2)$ in place of $x_j^{p_r}(t_2)$ so that we can reuse this lemma in Section 6.3.5.

Lemma 6.40. *If*

$$t_2 \in [t_r, t_3], \tag{6.77}$$

$$\tau_{j,\ell} \text{ is pending at } t_2, \tag{6.78}$$

$$s_{ub} \in (0, 1], \tag{6.79}$$

$$s(t) \leq s_{ub} \text{ for } t \in [y_{j,\ell}, t_2), \tag{6.80}$$

and

$$x_j^p(t_2) \text{ is } x^p\text{-sufficient}, \tag{6.81}$$

then

$$e^r_{j,\ell}(t_2) + D^e_j(b_{j,\ell}, t_3) \leq C_j + U^v_j \cdot s_{ub} \cdot x^p_j(t_2) + U^v_j \cdot (v(t_3) - v(t_2)).$$

If furthermore $t_3 = y_{i,k}$ *with* $j = i$ *and* $k \geq \ell$*, then*

$$e^r_{i,\ell}(t_2) + D^e_i(b_{i,\ell}, t_3) \leq e_{i,k} + U^v_i \cdot s_{ub} \cdot x^p_i(t_2) + U^v_i \cdot (v(t_3) - v(t_2)).$$

Proof. We will use the following fact.

If $y_{j,\ell} < t_2$, then the remaining precondition of Lemma 6.39 with $t_0 = y_{j,\ell}$ and $t_1 = t_2$ follows immediately from (6.80). Thus, by Lemma 6.39, the following claim holds.

Claim 6.40.1. *If* $y_{j,\ell} < t_2$*, then* $v(t_2) - v(y_{j,\ell}) \leq s_{ub} \cdot (t_2 - y_{j,\ell})$*.*

Continuing the proof of Lemma 6.40, we first provide reasoning that will address both the cases present in the statement of the lemma, which correspond to the two cases present in Lemma 6.32. We will then apply specific reasoning for each case. We have

$$\begin{aligned}
& e^r_{j,\ell}(t_2) + D^C_j(b_{j,\ell}, t_3) \\
\leq\ & \{\text{By Lemma 6.36}\} \\
& e^r_{j,\ell}(t_2) + \max\{0, U^v_j \cdot (v(t_3) - v(y_{j,\ell}))\} \\
=\ & \{\text{Rearranging}\} \\
& e^r_{j,\ell}(t_2) + \max\{0, U^v_j \cdot (v(t_2) - v(y_{j,\ell})) + U^v_j \cdot (v(t_3) - v(t_2))\} \\
\leq\ & \{\text{By (6.77)}\} \\
& e^r_{j,\ell}(t_2) + \max\{0, U^v_j \cdot (v(t_2) - v(y_{j,\ell}))\} + U^v_j \cdot (v(t_3) - v(t_2)) \\
\leq\ & \{\text{By Claim 6.40.1 if } y_{j,\ell} < t_2\text{, or by the 0 term in the ``max'' otherwise}\} \\
& e^r_{j,\ell}(t_2) + \max\{0, U^v_j \cdot s_{ub} \cdot (t_2 - y_{j,\ell})\} + U^v_j \cdot (v(t_3) - v(t_2)) \\
\leq\ & \{\text{By the definition of } x^p\text{-sufficient in Definition 6.19 with } t = t_2\} \\
& e^r_{j,\ell}(t_2) + \max\{0, U^v_j \cdot s_{ub} \cdot (x^p_j(t_2) + e^c_{j,\ell}(t_2))\} + U^v_j \cdot (v(t_3) - v(t_2)) \\
=\ & \{\text{Because } x^p_j(t_2) \geq 0 \text{ (by Definition 6.19) and } e^c_{j,\ell}(t_2) \geq 0\} \\
& e^r_{j,\ell}(t_2) + U^v_j \cdot s_{ub} \cdot (x^p_j(t_2) + e^c_{j,\ell}(t_2)) + U^v_j \cdot (v(t_3) - v(t_2))
\end{aligned}$$

$$\begin{aligned}
&\leq \{\text{By Property 6.1 with } i = j \text{ and } k = \ell\text{, and because } U_j^v \leq 1 \text{ and } s_{ub} \leq 1 \text{ by (6.79)}\}\\
&\quad e_{j,\ell} + U_j^v \cdot s_{ub} \cdot x_j^p(t_2) + U_j^v \cdot (v(t_3) - v(t_2)). \qquad (6.82)
\end{aligned}$$

Furthermore,

$$\begin{aligned}
&e^r_{j,\ell}(t_2) + D^e_j(b_{j,\ell}, t_3)\\
\leq\ &\{\text{By Lemmas 6.32 and 6.33 with } t_0 = b_{j,\ell} \text{ and } t_1 = t_3\text{, because } \tau_{j,\ell} \text{ is pending at } t_2 \geq t_r \text{ by}\\
&\text{(6.77) and (6.78)}\}\\
&e^r_{j,\ell}(t_2) + D^C_j(b_{j,\ell}, t_3)\\
\leq\ &\{\text{By (6.82)}\}\\
&e_{j,\ell} + U_j^v \cdot s_{ub} \cdot x_j^p(t_2) + U_j^v \cdot (v(t_3) - v(t_2)) \qquad (6.83)\\
\leq\ &\{\text{By Property 6.11, because } \tau_{j,\ell} \text{ is pending at } t_2 \geq t_r \text{ by (6.77) and (6.78)}\}\\
&C_j + U_j^v \cdot s_{ub} \cdot x_j^p(t_2) + U_j^v \cdot (v(t_3) - v(t_2)).
\end{aligned}$$

We divide the more specific case, when $t_3 = y_{i,k}$ with $j = i$ and $k \geq \ell$, into two subcases. If $k = \ell$, then

$$\begin{aligned}
&e^r_{i,\ell}(t_2) + D^e_i(b_{i,\ell}, t_3)\\
\leq\ &\{\text{By (6.83) with } j = i \text{ and } \ell = k\}\\
&e_{i,k} + U_i^v \cdot s_{ub} \cdot x_i^p(t_2) + U_i^v \cdot (v(t_3) - v(t_2)).
\end{aligned}$$

If $k > \ell$, then by (6.5) and the definition of $b_{i,\ell}$ in Definition 6.5, $r_{i,k} \geq b_{i,\ell}$. Therefore,

$$\begin{aligned}
&e^r_{i,\ell}(t_2) + D^e_i(b_{i,\ell}, t_3)\\
\leq\ &\{\text{By Lemmas 6.32 and 6.33 with } t_0 = b_{i,\ell} \text{ and } t_1 = t_3\text{, because } \tau_{i,\ell} \text{ is pending at } t_2 \geq t_r \text{ by}\\
&\text{(6.77) and (6.78)}\}\\
&e^r_{i,\ell}(t_2) + D^C_i(b_{i,\ell}, t_3) + e_{i,k} - C_i\\
\leq\ &\{\text{By (6.82) with } j = i\}\\
&e_{i,\ell} + U_i^v \cdot s_{ub} \cdot x_i^p(t_2) + U_i^v \cdot (v(t_3) - v(t_2)) + e_{i,k} - C_i
\end{aligned}$$

$$\leq \{\text{By Property 6.11, because } \tau_{i,\ell} \text{ is pending at } t_2 \geq t_r \text{ by (6.77) and (6.78)}\}$$

$$C_i + U_i^v \cdot s_{ub} \cdot x_i^p(t_2) + U_i^v \cdot (v(t_3) - v(t_2)) + e_{i,k} - C_i$$

$$= \{\text{Simplifying}\}$$

$$e_{i,k} + U_i^v \cdot s_{ub} \cdot x_i^p(t_2) + U_i^v \cdot (v(t_3) - v(t_2)). \qquad \square$$

We first use Lemma 6.40 to bound $e_{i,k}^p$ in Lemma 6.41.

Lemma 6.41. *If*

$$y_{i,k} \in [t_r, \infty), \tag{6.84}$$

$$t_{i,k}^c > y_{i,k} + e_{i,k}, \tag{6.85}$$

$$s_{ub} \in (0,1], \tag{6.86}$$

$$s(t) \leq s_{ub} \text{ for all } t \in [y_{i,\ell}, y_{i,k}) \text{ where } \tau_{i,\ell} \text{ is the earliest pending job of } \tau_i \text{ at } y_{i,k}, \tag{6.87}$$

and

$$x_i^p(y_{i,k}) \text{ is } x^p\text{-sufficient}, \tag{6.88}$$

then $e_{i,k}^p \leq U_i^v \cdot s_{ub} \cdot x_i^p(y_{i,k})$.

Proof. By (6.85), $\tau_{i,k}$ is pending at $y_{i,k}$. We consider two cases.

Case 1: $\tau_{i,k}$ is the Earliest Pending Job of τ_i at $y_{i,k}$. In this case,

$$e_{i,k}^p = \{\text{By the case we are considering}\}$$

$$0$$

$$\leq \{\text{By (6.86) and (6.88), because } x_i^p(y_{i,k}) \geq 0 \text{ by the definition of } x^p\text{-sufficient in Definition 6.19}\}$$

$$U_i^v \cdot s_{ub} \cdot x_i^p(y_{i,k}).$$

Case 2: $\tau_{i,\ell}$ with $\ell < k$ is the Earliest Pending Job of τ_i at $y_{i,k}$. In this case, we will use the following fact.

We will use Lemma 6.40 with $j = i$, $t_2 = y_{i,k}$, and $t_3 = y_{i,k}$. We have

$$t_2 = y_{i,k}$$

$$\begin{aligned}
&\geq \{\text{By (6.84)}\}\\
&t_r.
\end{aligned}$$

Furthermore $t_2 = y_{i,k} = t_3$. Therefore, $t_2 \in [t_r, t_3]$, which matches (6.77). By the case we are considering, $\tau_{i,\ell}$ is pending at $y_{i,k} = t_2$, so (6.78) holds. (6.79) holds by (6.86). Because $i = j$, (6.80) follows from (6.87). Finally, because $t_2 = y_{i,k}$, (6.81) follows from (6.88). Therefore, by Lemma 6.40,

$$e^r_{i,\ell}(y_{i,k}) + D^e_i(b_{i,\ell}, y_{i,k}) \leq e_{i,k} + U^v_i \cdot s_{ub} \cdot x^p_i(y_{i,k}) + U^v_i \cdot (v(y_{i,k}) - v(y_{i,k})). \tag{6.89}$$

Continuing the analysis of Case 2, by Lemma 6.3 with $t_0 = y_{i,k}$, the *total* remaining work from τ_i at $y_{i,k}$ is at most

$$\begin{aligned}
&e^r_{i,\ell}(y_{i,k}) + D^e_i(b_{i,\ell}, y_{i,k})\\
\leq\ &\{\text{By (6.89)}\}\\
&e_{i,k} + U^v_i \cdot s_{ub} \cdot x^p_i(y_{i,k}) + U^v_i \cdot (v(y_{i,k}) - v(y_{i,k}))\\
=\ &\{\text{Simplifying}\}\\
&e_{i,k} + U^v_i \cdot s_{ub} \cdot x^p_i(y_{i,k}). \qquad (6.90)
\end{aligned}$$

Of this work, $e_{i,k}$ units are from $\tau_{i,k}$ itself. Thus, subtracting $e_{i,k}$ yields the lemma. □

We will now provide, in Lemma 6.42, a more constrained bound on $e^p_{i,k}$ for the case that $y_{i,k} \in [t_r, t_n)$.

Lemma 6.42. *If $t_a \in [t_r, t_n)$, $t_a = y_{i,k}$, $t^c_{i,k} > y_{i,k} + e_{i,k}$, and (6.52) holds, then*

$$e^p_{i,k} \leq U^r_j \cdot x^{p_r}_i(y_{i,k}). \tag{6.91}$$

Proof. We will use the following facts.

We will make use of Lemma 6.38 with $t_2 = y_{i,k}$. Because we are using $t_2 = y_{i,k} = t_a$ and $t_a \in [t_r, t_n)$ by the statement of the lemma, we have $t_2 \in [t_r, t_n)$, which matches (6.71). Similarly, also because we are using $t_2 = y_{i,k} = t_a$, (6.72) follows immediately from (6.52). Therefore, by Lemma 6.38, we have

$$x^p_i(y_{i,k}) = x^{p_r}_i(y_{i,k}) \text{ is } x^p\text{-sufficient.} \tag{6.92}$$

We will also use Lemma 6.41 with $s_{ub} = s_r$ and $x_i^p(y_{i,k}) = x_i^{p_r}(y_{i,k})$. We have

$$\begin{aligned} y_{i,k} &= \{\text{By the statement of the lemma}\} \\ &\quad t_a \\ &\geq \{\text{By the statement of the lemma}\} \\ &\quad t_r, \end{aligned} \tag{6.93}$$

which matches (6.84). Furthermore, (6.85) holds by the statement of the lemma. Because $s_{ub} = s_r$, (6.86) holds by Property 6.6. As in (6.87), we denote as $\tau_{i,\ell}$ the earliest pending job of τ_i at $y_{i,k}$. By Lemma 6.9 with $j = i$ and $t_2 = y_{i,k}$, and by (6.93), $y_{i,\ell} \geq t_s$. Furthermore, we have

$$\begin{aligned} y_{i,k} &= \{\text{By the statement of the lemma}\} \\ &\quad t_a \\ &\leq \{\text{By the statement of the lemma}\} \\ &\quad t_n. \end{aligned}$$

Thus, (6.87) holds by Property 6.6. (6.88) holds by (6.92). Thus, by Lemma 6.41, we have

$$e_{i,k}^p \leq U_i^v \cdot s_r \cdot x_i^{p_r}(y_{i,k}). \tag{6.94}$$

Continuing the proof of Lemma 6.42, we have

$$\begin{aligned} e_{i,k}^p &\leq \{\text{By (6.94)}\} \\ &\quad U_i^v \cdot s_r \cdot x_i^{p_r}(y_{i,k}) \\ &= \{\text{By the definition of } U_i^r \text{ in (6.28)}\} \\ &\quad U_i^r \cdot x_i^{p_r}(y_{i,k}). \end{aligned}$$

□

The result of Lemma 6.42 involves $x_i^{p_r}(y_{i,k})$. The next lemma provides an upper bound on this quantity.

Lemma 6.43. *If $y_{i,k} < t_n$, then $x_i^{p_r}(y_{i,k}) \leq x_i^s(s_r) + \lambda$.*

Proof. We consider two cases, depending on the value of $y_{i,k}$.

Case 1: $y_{i,k} < t_r + \rho$**.** In this case, $y_{i,k} - \rho < t_r$. Therefore,

$$\begin{aligned}
x_i^s(s_r) + \lambda = &\{\text{By the definition of } \Delta(y_{i,k} - \rho) \text{ in (6.51)}\} \\
& x_i^s(s_r) + \Delta(y_{i,k} - \rho) \\
= &\{\text{By the definition of } x_i^{p_r}(y_{i,k}) \text{ in (6.73)}\} \\
& x_i^{p_r}(y_{i,k}).
\end{aligned}$$

Case 2: $y_{i,k} \geq t_r + \rho$**.** In this case, $y_{i,k} - \rho \geq t_r$. Therefore,

$$\begin{aligned}
x_i^s(s_r) + \lambda = &\{\text{By Lemma 6.20}\} \\
& x_i^s(s_r) + \Delta(t_r) \\
\geq &\{\text{By Lemma 6.24 with } t_0 = t_r \text{ and } t_1 = y_{i,k} - \rho\} \\
& x_i^s(s_r) + \Delta(y_{i,k} - \rho) \\
= &\{\text{By the definition of } x_i^{p_r}(y_{i,k}) \text{ in (6.73)}\} \\
& x_i^{p_r}(y_{i,k}).
\end{aligned}$$

□

We will next consider the remaining terms that appear in the definition of $x_{i,k}^m$ in (6.24). We will first consider in Lemmas 6.44–6.48 the case that $t_{i,k}^b \in [0, t_r)$, and then in Lemmas 6.49–6.56 we will consider the case that $t_{i,k}^b \in [t_r, t_n)$. In Lemma 6.57 we will prove that $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient for either range of $t_{i,k}^b$.

First, we upper bound $W_{i,k}$ when $t_{i,k}^b \in [0, t_r)$.

Lemma 6.44. *If $t_{i,k}^b \in [0, t_r)$ and $y_{i,k} \in [t_r, t_n)$, then*

$$W_{i,k} \leq W_{i,k}^o + \sum_{\tau_j \in \tau} U_j^r \cdot (y_{i,k} - t_r).$$

Proof. We have

$$W_{i,k}^o + \sum_{\tau_j \in \tau} U_j^r \cdot (y_{i,k} - t_r)$$

$= \{$By Lemma 6.29, because $t_r \geq t_s$ by the definition of t_r in Definition 6.13, and by the definition of U_j^r in (6.28)$\}$

$$W_{i,k}^o + \sum_{\tau_j \in \tau} U_j^v \cdot (v(y_{i,k}) - v(t_r))$$

$= \{$By the definition of $W_{i,k}^o$ in (6.42)$\}$

$$W_{i,k} - \sum_{\tau_j \in \tau} D_j^e(t_r, y_{i,k}) + \sum_{\tau_j \in \tau} S_j + \sum_{\tau_j \in \tau} U_j^v \cdot (v(y_{i,k}) - v(t_r))$$

$= \{$Rearranging$\}$

$$W_{i,k} - \sum_{\tau_j \in \tau} D_j^e(t_r, y_{i,k}) + \sum_{\tau_j \in \tau} (U_j^v \cdot (v(y_{i,k}) - v(t_r)) + S_j)$$

$\geq \{$By Lemmas 6.32 and 6.34 with $t_0 = t_r$ and $t_1 = y_{i,k}\}$

$$W_{i,k} - \sum_{\tau_j \in \tau} D_j^C(t_r, y_{i,k}) + \sum_{\tau_j \in \tau} (U_j^v \cdot (v(y_{i,k}) - v(t_r)) + S_j)$$

$\geq \{$By Lemma 6.35 with $t_0 = t_r$ and $t_1 = y_{i,k}\}$

$$W_{i,k}.$$

□

In a similar manner, we next consider the value of $R_{i,k}$ when $t_{i,k}^b \in [0, t_r)$.

Lemma 6.45. *If $t_{i,k}^b \in [0, t_r)$ and $y_{i,k} \in [t_r, t_n)$, then $R_{i,k}^o + u_{tot} \cdot (y_{i,k} - t_r) = R_{i,k}$.*

Proof. We have

$$R_{i,k}^o + u_{tot} \cdot (y_{i,k} - t_r)$$

$= \{$By the definition of $R_{i,k}^o$ in (6.43)$\}$

$$u_{tot} \cdot (t_r - t_{i,k}^b) + u_{tot} \cdot (y_{i,k} - t_r)$$

$= \{$Simplifying$\}$

$$u_{tot} \cdot (y_{i,k} - t_{i,k}^b)$$

$= \{$By the definition of $R_{i,k}$ in (6.21)$\}$

$$R_{i,k}.$$

□

We will provide an upper bound on $O_{i,k}$ in Lemma 6.46.

Lemma 6.46. *If $t^c_{i,k} > y_{i,k} + e_{i,k}$, $t^b_{i,k} \in [0, t_r)$, and $y_{i,k} \in [t_r, t_n)$, then $O^o_{i,k} + O^{rn} \geq O_{i,k}$.*

Proof. We first observe that, by the definition of $\beta_p(t_0, t_1)$ in Definition 6.7,

$$\beta_p(t^b_{i,k}, t^c_{i,k}) = \beta_p(t^b_{i,k}, t_r) + \beta_p(t_r, t^c_{i,k}). \tag{6.95}$$

We have

$$\begin{aligned}
O^o_{i,k} + O^{rn} &= \{\text{By the definition of } O^o_{i,k} \text{ in (6.44) and the definition of } O^{rn} \text{ in (6.39)}\} \\
&\quad \sum_{P_p \in P} o_p(t^b_{i,k}, t_r) + \sum_{P_p \in P} \widehat{u_p}\sigma_p \\
&\geq \{\text{By Property 6.12}\} \\
&\quad \sum_{P_p \in P} o_p(t^b_{i,k}, t_r) + \sum_{P_p \in P} o_p(t_r, t^c_{i,k}) \\
&= \{\text{Rearranging}\} \\
&\quad \sum_{P_p \in P} (o_p(t^b_{i,k}, t_r) + o_p(t_r, t^c_{i,k})) \\
&= \{\text{By the definition of } o_p(t_0, t_1) \text{ in (6.8)}\} \\
&\quad \sum_{P_p \in P} (\max\{0, \widehat{u_p} \cdot (t_r - t^b_{i,k}) - \beta_p(t^b_{i,k}, t_r)\} + \max\{0, \widehat{u_p} \cdot (t^c_{i,k} - t_r) - \beta_p(t_r, t^c_{i,k})\}) \\
&\geq \{\text{By the definition of “max”}\} \\
&\quad \sum_{P_p \in P} (\max\{0, \widehat{u_p} \cdot (t_r - t^b_{i,k}) - \beta_p(t^b_{i,k}, t_r) + \widehat{u_p} \cdot (t^c_{i,k} - t_r) - \beta_p(t_r, t^c_{i,k})\}) \\
&= \{\text{Rearranging, and by (6.95)}\} \\
&\quad \sum_{P_p \in P} (\max\{0, \widehat{u_p} \cdot (t^c_{i,k} - t^b_{i,k}) - \beta_p(t^b_{i,k}, t^c_{i,k})\}) \\
&= \{\text{By the definition of } o_p(t^b_{i,k}, t^c_{i,k}) \text{ in (6.8)}\} \\
&\quad \sum_{P_p \in P} o_p(t^b_{i,k}, t^c_{i,k}) \\
&= \{\text{By the definition of } O_{i,k} \text{ in (6.20)}\} \\
&\quad O_{i,k}.
\end{aligned}$$

□

When considering $t^b_{i,k}$ in $[0, t_r)$, we will consider the value of λ. We now provide a lower bound on λ that closely resembles the bound we desire.

Lemma 6.47. *For arbitrary $\tau_{i,k}$ such that $t^b_{i,k} \in [0, t_r)$ and $y_{i,k} \in [t_r, \infty)$,*

$$\lambda \geq \frac{W^o_{i,k} - R^o_{i,k} + (m - u_{tot} - 1)e_{i,k} + O^o_{i,k} + O^{rn} + L_i \cdot U^r_i \cdot (x^s_i(s_r) + \lambda)}{u_{tot}}.$$

Proof. By the definition of λ in (6.40), we have

$$\lambda \geq \frac{W^o_{i,k} - R^o_{i,k} + (m - u_{tot} - 1)e_{i,k} + O^o_{i,k} + O^{rn} + L_i \cdot U^r_i \cdot x^s_i(s_r)}{u_{tot} - L_i \cdot U^r_i}.$$

Adding $\lambda \cdot \frac{L_i \cdot U^r_i}{u_{tot} - L_i \cdot U^r_i}$ to both sides yields

$$\lambda \cdot \frac{u_{tot}}{u_{tot} - L_i \cdot U^r_i} \geq \frac{W^o_{i,k} - R^o_{i,k} + (m - u_{tot} - 1)e_{i,k} + O^o_{i,k} + O^{rn} + L_i \cdot U^r_i \cdot (x^s_i(s_r) + \lambda)}{u_{tot} - L_i \cdot U^r_i}.$$

By Property 6.13, the definition of U^r_i in (6.28), and the restriction that $s_r < 1$ in Property 6.6, $L_i \cdot U^r_i < u_{tot}$. Therefore, $\frac{u_{tot} - L_i \cdot U^r_i}{u_{tot}} > 0$. Thus, multiplying both sides by $\frac{u_{tot} - L_i \cdot U^r_i}{u_{tot}}$ yields the lemma. □

We now consider the $t^b_{i,k} \in [0, t_r)$ subcase of Case E.

Lemma 6.48. *If $t_a \in [t_r, t_n)$, $t_a = y_{i,k}$ for some k, $t^c_{i,k} > y_{i,k} + e_{i,k}$, $\tau_{i,k}$ is m-dominant for L_i, $t^b_{i,k} \in [0, t_r)$, and (6.52) holds, then $x_i(t_a) = x^s_i(s_r) + \Delta(t_a)$ is x-sufficient.*

Proof. We have

$$\begin{aligned}
& x^s_i(s_r) + \Delta(t_a) \\
\geq & \{\text{By Lemma 6.24 with } t_0 = t_r \text{ and } t_1 = y_{i,k};\ y_{i,k} \geq t_r \text{ holds by the statement of the lemma}\} \\
& x^s_i(s_r) + \Delta(t_r) + \phi \cdot (y_{i,k} - t_r) \\
= & \{\text{By Lemma 6.20}\} \\
& x^s_i(s_r) + \lambda + \phi \cdot (y_{i,k} - t_r) \\
\geq & \{\text{By Lemma 6.47}\} \\
& x^s_i(s_r) + \frac{W^o_{i,k} - R^o_{i,k} + (m - u_{tot} - 1)e_{i,k} + O^o_{i,k} + O^{rn} + L_i \cdot U^r_i \cdot (x^s_i(s_r) + \lambda)}{u_{tot}} + \phi \cdot (y_{i,k} - t_r)
\end{aligned}$$

$$
\begin{aligned}
&\geq \{\text{Because } x_i^s(s_r) \geq 0 \text{ by its definition in (6.37)}\}\\
&\quad \frac{W^o_{i,k} - R^o_{i,k} + (m - u_{tot} - 1)e_{i,k} + O^o_{i,k} + O^{rn} + L_i \cdot U_i^r \cdot (x_i^s(s_r) + \lambda)}{u_{tot}} + \phi \cdot (y_{i,k} - t_r)\\
&= \{\text{Rearranging}\}\\
&\quad \frac{W^o_{i,k} + R^o_{i,k} + u_{tot} \cdot \phi \cdot (y_{i,k} - t_r) + (m - u_{tot} - 1)e_{i,k} + O^o_{i,k} + O^{rn} + L_i \cdot U_i^r \cdot (x_i^s(s_r) + \lambda)}{u_{tot}}\\
&\geq \{\text{By Lemma 6.43}\}\\
&\quad \frac{W^o_{i,k} + R^o_{i,k} + u_{tot} \cdot \phi \cdot (y_{i,k} - t_r) + (m - u_{tot} - 1)e_{i,k} + O^o_{i,k} + O^{rn} + L_i \cdot U_i^r \cdot x_i^{p_r}(y_{i,k})}{u_{tot}}\\
&\geq \{\text{By Lemma 6.46, which only has preconditions implied by those of this lemma}\}\\
&\quad \frac{W^o_{i,k} + R^o_{i,k} + u_{tot} \cdot \phi \cdot (y_{i,k} - t_r) + (m - u_{tot} - 1)e_{i,k} + O_{i,k} + L_i \cdot U_i^r \cdot x_i^{p_r}(y_{i,k})}{u_{tot}}\\
&\geq \{\text{By Lemma 6.42, which only has preconditions implied by those of this lemma}\}\\
&\quad \frac{W^o_{i,k} + R^o_{i,k} + u_{tot} \cdot \phi \cdot (y_{i,k} - t_r) + (m - u_{tot} - 1)e_{i,k} + O_{i,k} + L_i \cdot e^p_{i,k}}{u_{tot}}. \qquad (6.96)
\end{aligned}
$$

For simplicity, we now consider some of the terms separately.

$$
\begin{aligned}
&W^o_{i,k} - R^o_{i,k} + u_{tot} \cdot \phi \cdot (y_{i,k} - t_r)\\
&\geq \{\text{By the definition of } \phi \text{ in (6.46)}\}\\
&W^o_{i,k} - R^o_{i,k} + \left(\sum_{\tau_j \in \tau} U_j^r - u_{tot}\right) \cdot (y_{i,k} - t_r)\\
&= \{\text{Rearranging}\}\\
&W^o_{i,k} + \sum_{\tau_j \in \tau} U_j^r \cdot (y_{i,k} - t_r) - R^o_{i,k} - u_{tot} \cdot (y_{i,k} - t_r)\\
&\geq \{\text{By Lemmas 6.44 and 6.45, which only have preconditions implied by those of this lemma}\}\\
&W_{i,k} - R_{i,k}. \qquad (6.97)
\end{aligned}
$$

Combining, we have

$$
\begin{aligned}
x_i^s(s_r) + \Delta(t_a) \geq\ & \{\text{By (6.96) and (6.97)}\}\\
& \frac{W_{i,k} - R_{i,k} + (m - u_{tot} - 1)e_{i,k} + O_{i,k} + L_i \cdot e^p_{i,k}}{u_{tot}}
\end{aligned}
$$

$$= \{\text{By the definition of } x^m_{i,k} \text{ in (6.24)}\}$$

$$x^m_{i,k}. \tag{6.98}$$

By Theorem 6.5, which has only preconditions implied by those of this lemma, $x_i(t_a) = x^m_{i,k}$ is x-sufficient. Therefore, by Property 6.4 with $c_0 = x^m_{i,k}$ and $c_1 = x^s_i(s_r) + \Delta(t_a)$ and by (6.98), $x_i(t_a) = x^s_i(s_r) + \Delta(t_a)$ is x-sufficient. □

In Lemmas 6.49–6.56, we now turn our attention to the case when $t^b_{i,k} \in [t_r, t_n)$. In order to facilitate use in the next subsection, some of these lemmas also apply when $t^b_{i,k} \in [t_n, \infty)$. Observe in the definition of $x^m_{i,k}$ in (6.24) the presence of the term $(m - u_{tot} - 1)e_{i,k}$. This term was previously accounted for by its explicit inclusion in the definition of λ in (6.40). However, we only use the value of λ while analyzing $\tau_{i,k}$ with $t^b_{i,k} \in [t_r, t_n)$. In most lemmas in this section, terms with $e_{i,k}$ can be upper bounded by using C_i instead. However, depending on the size of u_{tot}, $m - u_{tot} - 1$ can be as small as -1 or as big as $m - 1$. If $(m - u_{tot} - 1) \in [0, m-1)$, then $(m - u_{tot} - 1)C_i \geq (m - u_{tot} - 1)e_{i,k}$. However, if $(m - u_{tot} - 1) \in [-1, 0)$, then $(m - u_{tot} - 1)C_i$ could be smaller than $(m - u_{tot} - 1)e_{i,k}$ by as much as $C_i - e_{i,k}$.

As we will show, we can use the less general cases in Lemmas 6.32 and 6.40 while bounding $W_{i,k}$ in order to cancel out this discrepancy. Our reasoning will depend on whether τ_i has a job in $\theta_{i,k}$ or is in $\overline{\theta_{i,k}}$. To handle these cases, in the next lemma we define an indicator variable $\Omega_{i,k}(j)$ that will be used for all tasks in either case, but that will be nonzero only when $j = i$. We first consider how to handle a task τ_j with a job in $\theta_{i,k}$, based on the expression that appears in the sum for $\theta_{i,k}$ in the definition of $W_{i,k}$ in (6.17).

Lemma 6.49. *If*

$$\tau_{j,\ell} \in \theta_{i,k}, \tag{6.99}$$

$$t^b_{i,k} \in [t_r, \infty), \tag{6.100}$$

$$s_{ub} \in (0, 1], \tag{6.101}$$

$$s(t) \leq s_{ub} \text{ for } t \in [y_{j,\ell}, t^b_{i,k}), \tag{6.102}$$

and

$$x^p_j\left(t^b_{i,k}\right) \text{ is } x^p\text{-sufficient}, \tag{6.103}$$

then

$$e^r_{j,\ell}(t^b_{i,k}) + D^e_j(b_{j,\ell}, y_{i,k}) \leq C_j + U^v_j \cdot s_{ub} \cdot x^p_j\left(t^b_{i,k}\right) + U^v_j \cdot (v(y_{i,k}) - v(t^b_{i,k})) + \Omega_{i,k}(j),$$

where

$$\Omega_{i,k}(j) \triangleq \begin{cases} e_{i,k} - C_i & \text{If } j = i \\ 0 & \text{If } j \neq i. \end{cases} \tag{6.104}$$

Proof. We will use the following fact.

We will use Lemma 6.40 with $t_2 = t^b_{i,k}$ and $t_3 = y_{i,k}$. Because $t_2 = t^b_{i,k}$, by (6.100), $t_2 \geq t_r$, and by the definition of $t^b_{i,k}$ in Definition 6.12, $t_2 \leq y_{i,k} = t_3$. Thus, (6.77) holds. By (6.99) and the definition of $\theta_{i,k}$ in Lemma 6.4, (6.78) holds. (6.79) holds by (6.101). Furthermore, because $t_2 = t^b_{i,k}$, (6.80) follows from (6.102), and (6.81) follows from (6.103). Thus, by Lemma 6.40, the following claim holds.

Claim 6.49.1.

$$e^r_{j,\ell}(t^b_{i,k}) + D^e_j(b_{j,\ell}, y_{i,k}) \leq C_j + U^v_j \cdot s_{ub} \cdot x^p_j\left(t^b_{i,k}\right) + U^v_j \cdot (v(y_{i,k}) - v(t^b_{i,k})).$$

If furthermore $j = i$ and $k \geq \ell$, then

$$e^r_{i,\ell}(t^b_{i,k}) + D^e_i(b_{i,\ell}, y_{i,k}) \leq e_{i,k} + U^v_i \cdot s_{ub} \cdot x^p_i\left(t^b_{i,k}\right) + U^v_i \cdot (v(y_{i,k}) - v(t^b_{i,k})).$$

Continuing the proof of Lemma 6.49, we consider two cases, depending whether $j = i$.

Case 1: $j = i$**.** Because $\tau_{j,\ell} \in \theta_{i,k}$ by (6.99), by the definition of $\theta_{i,k}$ in Lemma 6.4, $y_{j,\ell} \leq y_{i,k}$. Thus, because $j = i$, $y_{i,\ell} \leq y_{i,k}$. Therefore, $k \geq \ell$. Thus,

$$\begin{aligned}
& e^r_{j,\ell}(t^b_{i,k}) + D^e_j(b_{j,\ell}, y_{i,k}) \\
\leq & \{\text{By Claim 6.49.1, and by the case we are considering}\} \\
& e_{i,k} + U^v_i \cdot s_{ub} \cdot x^p_i\left(t^b_{i,k}\right) + U^v_i \cdot (v(y_{i,k}) - v(t^b_{i,k})) \\
= & \{\text{Rearranging}\} \\
& C_i + U^v_i \cdot s_{ub} \cdot x^p_i\left(t^b_{i,k}\right) + U^v_i \cdot (v(y_{i,k}) - v(t^b_{i,k})) + e_{i,k} - C_i \\
= & \{\text{By the definition of } \Omega_{i,k}(i) \text{ in (6.104)}\}
\end{aligned}$$

$$
\begin{aligned}
&C_i + U_i^v \cdot s_{ub} \cdot x_i^p\left(t_{i,k}^b\right) + U_i^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) + \Omega_{i,k}(i) \\
&= \{\text{By the case we are considering}\} \\
&\quad C_j + U_j^v \cdot s_{ub} \cdot x_j^p\left(t_{i,k}^b\right) + U_j^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) + \Omega_{i,k}(j)
\end{aligned}
\tag{6.105}
$$

Case 2: $j \neq i$**.**

$$
\begin{aligned}
&e_{j,\ell}^r(t_{i,k}^b) + D_j^e(b_{j,\ell}, y_{i,k}) \\
&\leq \{\text{By Claim 6.49.1}\} \\
&\quad C_j + U_j^v \cdot s_{ub} \cdot x_j^p\left(t_{i,k}^b\right) + U_j^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) \\
&= \{\text{By the definition of } \Omega_{i,k}(j) \text{ in (6.104)}\} \\
&\quad C_j + U_j^v \cdot s_{ub} \cdot x_j^p\left(t_{i,k}^b\right) + U_j^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) + \Omega_{i,k}(j).
\end{aligned}
$$

□

The next lemma is similar, but handles the expression that appears in the sum for $\overline{\theta_{i,k}}$ in the definition of $W_{i,k}$ in (6.17).

Lemma 6.50. *If* $\tau_j \in \overline{\theta_{i,k}}$, $t_{i,k}^c > y_{i,k} + e_{i,k}$, *and* $t_{i,k}^b \in [t_r, \infty)$, *then*

$$D_j^e(t_{i,k}^b, y_{i,k}) \leq U_j^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) + S_j + \Omega_{i,k}(j).$$

Proof. We consider two cases, depending whether $j = i$.

Case 1: $j = i$**.** In this case, we have

$$
\begin{aligned}
t_{i,k}^c &> \{\text{By the statement of the lemma}\} \\
&\quad y_{i,k} + e_{i,k} \\
&> \{\text{Because } e_{i,k} > 0\} \\
&\quad y_{i,k} \\
&\geq \{\text{By the definition of } t_{i,k}^b \text{ in Definition 6.12}\} \\
&\quad t_{i,k}^b.
\end{aligned}
$$

Thus, $\tau_{i,k}$ must not be complete at $t^b_{i,k}$.

Suppose $r_{i,k} < t^b_{i,k}$. In that case, by the definition of $\theta_{i,k}$ in Lemma 6.4, $\tau_{i,k}$ must be in $\theta_{i,k}$, unless it has a predecessor in $\theta_{i,k}$ (which would have an earlier release and PP). However, $\tau_i = \tau_j$, so by the definition of $\overline{\theta_{i,k}}$ in Lemma 6.4, this contradicts the precondition that $\tau_j \in \overline{\theta_{i,k}}$. Therefore, $r_{i,k} \geq t^b_{i,k}$. Thus, we have

$$\begin{aligned}
D^e_j(t^b_{i,k}, y_{i,k}) \leq {} & \{\text{By Lemmas 6.32 and 6.34 with } t_0 = t^b_{i,k} \text{ and } t_1 = y_{i,k}\} \\
& D^C_j(t^b_{i,k}, y_{i,k}) + e_{i,k} - C_i \\
= {} & \{\text{By the definition of } \Omega_{i,k}(j) \text{ in (6.104) and the case we are considering}\} \\
& D^C_j(t^b_{i,k}, y_{i,k}) + \Omega_{i,k}(j) \\
\leq {} & \{\text{By Lemma 6.35 with } t_0 = t^b_{i,k} \text{ and } t_1 = y_{i,k};\ t^b_{i,k} \leq y_{i,k} \text{ holds by the definition of} \\
& t^b_{i,k} \text{ in Definition 6.12}\} \\
& U^v_j \cdot (v(y_{i,k}) - v(t^b_{i,k})) + S_j + \Omega_{i,k}(j).
\end{aligned}$$

Case 2: $j \neq i$. In this case, we have

$$\begin{aligned}
D^e_j(t^b_{i,k}, y_{i,k}) \leq {} & \{\text{By Lemmas 6.32 and 6.34 with } t_0 = t^b_{i,k} \text{ and } t_1 = y_{i,k}\} \\
& D^C_j(t^b_{i,k}, y_{i,k}) \\
= {} & \{\text{By the definition of } \Omega_{i,k}(j) \text{ in (6.104) and the case we are considering}\} \\
& D^C_j(t^b_{i,k}, y_{i,k}) + \Omega_{i,k}(j) \\
\leq {} & \{\text{By Lemma 6.35 with } t_0 = t^b_{i,k} \text{ and } t_1 = y_{i,k};\ t^b_{i,k} \leq y_{i,k} \text{ holds by the definition of} \\
& t^b_{i,k} \text{ in Definition 6.12}\} \\
& U^v_j \cdot (v(y_{i,k}) - v(t^b_{i,k})) + S_j + \Omega_{i,k}(j). \qquad \Box
\end{aligned}$$

We now combine these results, in Lemma 6.51 below, to provide a bound on $W_{i,k}$ when $y_{i,k} \in [t_r, \infty)$. This bound is valid both when $t^b_{i,k} \in [t_r, t_n)$ and when $t^b_{i,k} \in [t_n, \infty)$.

Lemma 6.51. *If*

$$t^c_{i,k} > y_{i,k} + e_{i,k}, \tag{6.106}$$

$$t^b_{i,k} \in [t_r, \infty), \tag{6.107}$$

$$s_{ub} \in (0,1] \tag{6.108}$$

$$s(t) \leq s_{ub} \textit{ for } t \in [y_{j,\ell}, t^b_{i,k}) \textit{ for each } \tau_{j,\ell} \in \theta_{i,k}, \tag{6.109}$$

and

$$\textit{For each } \tau_j,\, x^p_j\left(t^b_{i,k}\right) \textit{ is } x^p\textit{-sufficient,} \tag{6.110}$$

then

$$W_{i,k} \leq \sum_{m-1 \text{ largest}} \left(C_j + U^v_j \cdot s_{ub} \cdot x^p_j\left(t^b_{i,k}\right) - S_j\right) + \sum_{\tau_j \in \tau} S_j + \sum_{\tau_j \in \tau} U^v_j \cdot (v(y_{i,k}) - v(t^b_{i,k})) + e_{i,k} - C_i.$$

Proof. We will use the following fact.

We will use Lemma 6.49 for each $\tau_{j,\ell} \in \theta_{i,k}$. In each such case, (6.99) holds by assumption. Furthermore, (6.100)–(6.103) are implied by (6.107)–(6.110). Therefore, the following claim holds by Lemma 6.49:

Claim 6.51.1. *If $\tau_{j,\ell} \in \theta_{i,k}$, then*

$$e^r_{j,\ell}(t^b_{i,k}) + D^e_j(b_{j,\ell}, y_{i,k}) \leq C_j + U^v_j \cdot s_{ub} \cdot x^p_j\left(t^b_{i,k}\right) + U^v_j \cdot (v(y_{i,k}) - v(t^b_{i,k})) + \Omega_{i,k}(j).$$

We now continue the proof of Lemma 6.51. First, observe that, by the definitions of $\theta_{i,k}$ and $\overline{\theta_{i,k}}$, and because a given task can only have one *earliest* pending job at $t^b_{i,k}$, the following claim holds.

Claim 6.51.2. *Each τ_j either has exactly one $\tau_{j,\ell}$ in $\theta_{i,k}$ or is in $\overline{\theta_{i,k}}$. These possibilities are mutually exclusive.*

We have

$$\begin{aligned} W_{i,k} = {} & \{\text{By the definition of } W_{i,k} \text{ in (6.17)}\} \\ & \sum_{\tau_{j,\ell} \in \theta_{i,k}} (e^r_{j,\ell}(t^b_{i,k}) + D^e_j(b_{j,\ell}, y_{i,k})) + \sum_{\tau_j \in \overline{\theta_{i,k}}} D^e_j(t^b_{i,k}, y_{i,k}) \\ \leq {} & \{\text{By Claim 6.51.1 and Lemma 6.50, which only has preconditions implied by those of this lemma}\} \\ & \sum_{\tau_{j,\ell} \in \theta_{i,k}} \left(C_j + U^v_j \cdot s_{ub} \cdot x^p_j\left(t^b_{i,k}\right) + U^v_j \cdot (v(y_{i,k}) - v(t^b_{i,k})) + \Omega_{i,k}(j)\right) \end{aligned}$$

$$
\begin{aligned}
&+ \sum_{\tau_j \in \overline{\theta_{i,k}}} \left(U_j^v \cdot \left(v(y_{i,k}) - v(t_{i,k}^b)\right) + S_j + \Omega_{i,j}(k)\right) \\
= {} & \{\text{Rearranging}\} \\
& \sum_{\tau_{j,\ell} \in \theta_{i,k}} \left(C_j + U_j^v \cdot s_{ub} \cdot x_j^p\left(t_{i,k}^b\right)\right) \\
&+ \sum_{\tau_{j,\ell} \in \theta_{i,k}} \left(U_j^v \cdot \left(v(y_{i,k}) - v(t_{i,k}^b)\right)\right) + \sum_{\tau_j \in \overline{\theta_{i,k}}} \left(U_j^v \cdot \left(v(y_{i,k}) - v(t_{i,k}^b)\right) + S_j\right) \\
&+ \sum_{\tau_{j,\ell} \in \theta_{i,k}} \Omega_{i,k}(j) + \sum_{\tau_j \in \overline{\theta_{i,k}}} \Omega_{i,k}(j) \\
= {} & \{\text{By Claim 6.51.2}\} \\
& \sum_{\tau_{j,\ell} \in \theta_{i,k}} \left(C_j + U_j^v \cdot s_{ub} \cdot x_j^p\left(t_{i,k}^b\right) - S_j\right) + \sum_{\tau_j \in \tau} S_j + \sum_{\tau_j \in \tau} U_j^v \cdot \left(v(y_{i,k}) - v(t_{i,k}^b)\right) + \sum_{\tau_j \in \tau} \Omega_{i,k}(j). \qquad (6.111)
\end{aligned}
$$

By the definition of $\theta_{i,k}$ in Lemma 6.4, any τ_j with a $\tau_{j,\ell} \in \theta_{i,k}$ must be executing immediately before $t_{i,k}^b$, because $\tau_{j,\ell}$ was released before $t_{i,k}^b$, $\tau_{j,\ell}$ is still pending at $t_{i,k}^b$, and there is an idle processor just before $t_{i,k}^b$. Therefore, there can be at most $m-1$ tasks with jobs in $\theta_{i,k}$. Furthermore, it is more pessimistic to assume that a task has a job in $\theta_{i,k}$ than that it is in $\overline{\theta_{i,k}}$, because

$$
\begin{aligned}
& C_j + U_j^v \cdot s_{ub} \cdot x_j^p\left(t_{i,k}^b\right) \\
\geq {} & \{\text{Because } U_j^v > 0 \text{ and } s_{ub} > 0 \text{, and because } x_j^p\left(t_{i,k}^b\right) \geq 0 \text{ by the definition of } x^p\text{-sufficient in} \\
& \text{Definition 6.19}\} \\
& C_j \\
\geq {} & \{\text{Because } Y_i \geq 0 \text{ and } T_i > 0\} \\
& C_j \cdot \left(1 - \frac{Y_i}{T_i}\right) \\
= {} & \{\text{By the definition of } S_j \text{ in (6.38)}\} \\
& S_i.
\end{aligned}
$$

Therefore,

$$
W_{i,k} \leq \{\text{By (6.111) and the above reasoning}\}
$$

$$\sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_{ub} \cdot x_j^p\left(t_{i,k}^b\right) - S_j) + \sum_{\tau_j \in \tau} S_j + \sum_{\tau_j \in \tau} U_j^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) + \sum_{\tau_j \in \tau} \Omega_{i,j}(k)$$

$$= \{\text{By the definition of } \Omega_{i,k}(j) \text{ in (6.104)}\}$$

$$\sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_{ub} \cdot x_j^p\left(t_{i,k}^b\right) - S_j) + \sum_{\tau_j \in \tau} S_j + \sum_{\tau_j \in \tau} U_j^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) + e_{i,k} - C_i. \qquad \square$$

We now offer a bound that accounts for the $-e_{i,k}$ term that occurs as part of the $(m - u_{tot} - 1)e_{i,k}$ term in the definition of $x_{i,k}^m$ in (6.24). This lemma plays a similar role to Lemma 6.44 above, but for the case that $t_{i,k}^b \in [t_r, t_n)$.

Lemma 6.52. *If $t_{i,k}^c > y_{i,k} + e_{i,k}$, $t_{i,k}^b \in [t_r, t_n)$, $y_{i,k} \in [t_r, t_n)$, and (6.72) holds with $t_2 = t_{i,k}^b$, then*

$$W_{i,k}^r + \sum_{\tau_j \in \tau} U_j^r \cdot (y_{i,k} - t_{i,k}^b) - C_i \geq W_{i,k} - e_{i,k},$$

where

$$W_{i,k}^r \triangleq \sum_{m-1 \text{ largest}} \left(C_j + U_j^r \cdot x_j^{p_r}\left(t_{i,k}^b\right) - S_j\right) + \sum_{\tau_j \in \tau} S_j. \tag{6.112}$$

Proof. We will use the following facts.

We will use Lemma 6.38 with $t_2 = t_{i,k}^b$ and arbitrary τ_j. Because $t_{i,k}^b \in [t_r, t_n)$ by the statement of the lemma, (6.71) holds. Furthermore, (6.72) holds by the statement of the lemma. Therefore, by Lemma 6.38,

$$\text{For each } \tau_j,\ x_j^p\left(t_{i,k}^b\right) = x_j^{p_r}\left(t_{i,k}^b\right) \text{ is } x^p\text{-sufficient.} \tag{6.113}$$

We will use Lemma 6.51 with $s_{ub} = s_r$ and $x_j^p\left(t_{i,k}^b\right) = x_j^{p_r}\left(t_{i,k}^b\right)$. (6.106) and (6.107) hold by the statement of the lemma. Because $s_{ub} = s_r$, (6.108) follows from Property 6.6. To establish (6.109), we consider arbitrary $\tau_{j,\ell}$ in $\theta_{i,k}$. By the definition of $\theta_{i,k}$ in Lemma 6.4, such a $\tau_{j,\ell}$ is pending at $t_{i,k}^b$, and $t_{i,k}^b \geq t_r$ by the statement of the lemma. Thus, by Lemma 6.9 with $t_2 = t_{i,k}^b$, we have $y_{j,\ell} \geq t_s$. Therefore, by Property 6.6, for all $t \in [y_{j,\ell}, t_{i,k}^b)$, $s(t) = s_r$. (This statement is vacuously true if $y_{j,\ell} \geq t_{i,k}^b$.) (6.113) matches (6.110). Therefore, by Lemma 6.51,

$$W_{i,k} \leq \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_{ub} \cdot x_j^p\left(t_{i,k}^b\right) - S_j) + \sum_{\tau_j \in \tau} S_j + \sum_{\tau_j \in \tau} U_j^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) + e_{i,k} - C_i. \tag{6.114}$$

Continuing the proof of Lemma 6.52, we have

$$
\begin{aligned}
& W^r_{i,k} + \sum_{\tau_j \in \tau} U^r_j \cdot (y_{i,k} - t^b_{i,k}) - C_i \\
= {} & \{\text{By Lemma 6.29, because } t_r \geq t_s \text{ by the definition of } t_r \text{ in Definition 6.13, and by} \\
& \text{the definition of } U^r_i \text{ in (6.28)}\} \\
& W^r_{i,k} + \sum_{\tau_j \in \tau} U^v_j \cdot (v(y_{i,k}) - v(t^b_{i,k})) - C_i \\
= {} & \{\text{By the definition of } W^r_{i,k} \text{ in (6.112)}\} \\
& \sum_{m-1 \text{ largest}} (C_j + U^r_j \cdot x^{p_r}_j \left(t^b_{i,k}\right) - S_j) + \sum_{\tau_j \in \tau} S_j + \sum_{\tau_j \in \tau} U^v_j \cdot (v(y_{i,k}) - v(t^b_{i,k})) - C_i \\
= {} & \{\text{Rewriting}\} \\
& \sum_{m-1 \text{ largest}} (C_j + U^r_j \cdot x^{p_r}_j \left(t^b_{i,k}\right) - S_j) + \sum_{\tau_j \in \tau} S_j + \sum_{\tau_j \in \tau} U^v_j \cdot (v(y_{i,k}) - v(t^b_{i,k})) + e_{i,k} - C_i - e_{i,k} \\
\geq {} & \{\text{By (6.114) and the definition of } U^r_j \text{ in (6.28)}\} \\
& W_{i,k} - e_{i,k}.
\end{aligned}
$$

□

We next bound $O_{i,k}$. This lemma plays the same role as Lemma 6.46, but for the case that $t^b_{i,k} \in [t_r, \infty)$.

Lemma 6.53. *If $t^c_{i,k} > y_{i,k} + e_{i,k}$ and $t^b_{i,k} \in [t_r, \infty)$, then $O^{rn} \geq O_{i,k}$.*

Proof. We have

$$
\begin{aligned}
t^c_{i,k} > {} & \{\text{By the statement of the lemma}\} \\
& y_{i,k} + e_{i,k} \\
> {} & \{\text{Because } e_{i,k} > 0\} \\
& y_{i,k} \\
\geq {} & \{\text{By the definition of } t^b_{i,k} \text{ in Definition 6.12}\} \\
& t^b_{i,k}.
\end{aligned}
\tag{6.115}
$$

Thus,

$$
\begin{aligned}
O^{rn} = {} & \{\text{By the definition of } O^{rn} \text{ in (6.39)}\} \\
& \sum_{P_p \in P} \widehat{u_p} \sigma_p \\
\geq {} & \{\text{By Property 6.12 and (6.115)}\} \\
& \sum_{P_p \in P} o_p(t_{i,k}^b, t_{i,k}^c) \\
= {} & \{\text{By the definition of } O_{i,k} \text{ in (6.20)}\} \\
& O_{i,k}. \qquad \square
\end{aligned}
$$

Many previous lemmas in this section were based on Lemma 6.24, which essentially states that $\Delta(t)$ decreases sufficiently slowly in an *additive* sense. In Lemma 6.55 below, we will bound the value of $x_i^s(s_r) + \Delta\left(t_{i,k}^b\right)$ by using the fact that $\Delta(t)$ decreases sufficiently slowly in a *multiplicative* sense. In Lemma 6.56, we will then use Lemma 6.55 to bound the value of $x_i^s(s_r) + \Delta(y_{i,k})$.

The next lemma shows that $\Delta(t)$ decreases sufficiently slowly in this multiplicative sense, in a similar manner to how Lemma 6.24 shows that $\Delta(t)$ decreases sufficiently slowly in an additive sense.

Lemma 6.54. *For all t_2, $\Delta(t_2) \geq q \cdot \Delta(t_2 - \rho)$.*

Proof. We will consider a function

$$g(t_0, t_1) \triangleq \Delta(t_1) \cdot q^{\frac{t_0 - t_1}{\rho}}. \tag{6.116}$$

(Recall that, by Lemma 6.14, $\rho > 0$.) Observe that

$$
\begin{aligned}
\Delta(t_2) = {} & \{\text{Rewriting}\} \\
& \Delta(t_2) \cdot q^{\frac{t_2 - \rho - t_2}{\rho}} \cdot q^{\frac{-t_2 + \rho + t_2}{\rho}} \\
= {} & \{\text{By the definition of } g(t_2 - \rho, t_2) \text{ in (6.116)}\} \\
& g(t_2 - \rho, t_2) \cdot q^{\frac{-t_2 + \rho + t_2}{\rho}} \\
= {} & \{\text{Simplifying}\} \\
& q \cdot g(t_2 - \rho, t_2).
\end{aligned} \tag{6.117}
$$

By (6.117), the following claim holds.

Claim 6.54.1. *If $g(t_2-\rho,t_2) \geq \Delta(t_2-\rho)$, the lemma holds.*

Therefore, we prove the lemma by establishing that $g(t_2-\rho,t_2) \geq \Delta(t_2-\rho)$. In order to facilitate re-use of intermediate results, we will start by proving the more general claim that $g(t_0,t_1) \geq \Delta(t_0)$ for $t_0 \leq t_1$. First, we observe that

$$\begin{aligned} g(t_1,t_1) &= \{\text{By the definition of } g(t_1,t_1) \text{ in (6.116)}\} \\ &\quad \Delta(t_1)\cdot q^{\frac{t_1-t_1}{\rho}} \\ &= \{\text{Simplifying}\} \\ &\quad \Delta(t_1). \end{aligned} \tag{6.118}$$

We consider the derivative of $g(t,t_1)$, with respect to t, denoted $g'(t,t_1)$, for $t \in [t_0,t_1)$. For such t, we have

$$\begin{aligned} g'(t,t_1) &= \{\text{By (6.116) and differentiation}\} \\ &\quad \frac{\ln q}{\rho}\cdot\Delta(t_1)\cdot q^{\frac{t-t_1}{\rho}} \\ &\leq \{\text{Because } t<t_1,\ \Delta(t_1)\geq 0 \text{ by Lemma 6.16 with } t_a=t_1,\ 0<q<1 \text{ by Lemma 6.13, and } \rho>0 \\ &\quad \text{by Lemma 6.14}\} \\ &\quad \frac{\ln q}{\rho}\cdot\Delta(t_1). \end{aligned} \tag{6.119}$$

By Lemma 6.23 and the FTC with $f(t) = \Delta(t)$,

$$\Delta(t_1) = \Delta(t_0) + \int_{t_0}^{t_1} \Delta'(t)\,dt.$$

Rearranging,

$$\Delta(t_0) = \Delta(t_1) - \int_{t_0}^{t_1} \Delta'(t)\,dt. \tag{6.120}$$

By identical reasoning,

$$\begin{aligned} g(t_0,t_1) &= g(t_1,t_1) - \int_{t_0}^{t_1} g'(t,t_1)\,dt \\ &= \{\text{By (6.118)}\} \\ &\quad \Delta(t_1) - \int_{t_0}^{t_1} g'(t,t_1)\,dt. \end{aligned} \tag{6.121}$$

We consider three cases, depending on the value of t_1. We use intervals closed on the right in order to reduce the number of edge cases we must consider.

Case 1: $t_1 \in (-\infty, t_r]$**.** In this case,

$$\begin{aligned} \Delta(t_1) &= \{\text{By the definition of } \Delta(t) \text{ in (6.51), or by Lemma 6.20}\} \\ &\quad \lambda \\ &\geq \{\text{By the definition of } \lambda \text{ in (6.40)}\} \\ &\quad 0. \end{aligned} \tag{6.122}$$

Therefore, by (6.119), because $0 < q < 1$ by Lemma 6.13, because $\Delta(t_1) \geq 0$ by Lemma 6.16, and because $\rho > 0$ by Lemma 6.14,

$$g'(t,t_1) \leq 0 \tag{6.123}$$

for $t \in [t_0, t_1)$. We have

$$\begin{aligned} g(t_0,t_1) &= \{\text{By (6.121)}\} \\ &\quad \Delta(t_1) - \int_{t_0}^{t_1} g'(t,t_1)\,dt \\ &\geq \{\text{By (6.123)}\} \\ &\quad \Delta(t_1) - \int_{t_0}^{t_1} 0\,dt \\ &= \{\text{By the definition of } \Delta'(t) \text{ in (6.57)}\} \\ &\quad \Delta(t_1) - \int_{t_0}^{t_1} \Delta'(t)\,dt \end{aligned}$$

$$
\begin{aligned}
&= \{\text{By (6.120)}\} \\
&\quad \Delta(t_0).
\end{aligned}
$$

We state this result as a claim.

Claim 6.54.2. *If $t_0 \le t_1$ and $t_1 \in (-\infty, t_r]$, then $g(t_0,t_1) \ge \Delta(t_0)$.*

Case 2: $t_1 \in (t_r, t_e]$. $(t_r, t_e]$ cannot be empty, because it contains t_1. Thus, $t_r < t_e$. Therefore, by the definition of t_e in (6.48), $\lambda > \phi \cdot \frac{\rho}{\ln q}$. Thus,

$$
\begin{aligned}
\Delta(t_1) &\ge \{\text{By Lemma 6.24 with } t_0 = t_1 \text{ and } t_1 = t_e\} \\
&\quad \Delta(t_e) \\
&= \{\text{By Lemma 6.15}\} \\
&\quad \phi \cdot \frac{\rho}{\ln q}. \qquad (6.124)
\end{aligned}
$$

Therefore, for $t \in [t_0, t_1)$, we have

$$
\begin{aligned}
g'(t,t_1) &\le \{\text{By (6.119)}\} \\
&\quad \frac{\ln q}{\rho} \cdot \Delta(t_1) \\
&\le \{\text{By (6.124); } \ln q < 0 \text{ holds because } 0 < q < 1 \text{ by Lemma 6.13,} \\
&\quad \text{and } \rho > 0 \text{ by Lemma 6.14}\} \\
&\quad \phi. \qquad (6.125)
\end{aligned}
$$

Thus,

$$
\begin{aligned}
g(t_0,t_1) &= \{\text{By (6.121)}\} \\
&\quad \Delta(t_1) - \int_{t_0}^{t_1} g'(t,t_1)\,dt \\
&\ge \{\text{By (6.125)}\} \\
&\quad \Delta(t_1) - \int_{t_0}^{t_1} \phi\,dt \\
&\ge \{\text{Because } \phi \le \Delta'(t) \text{ for all } t \text{ by Lemma 6.23}\}
\end{aligned}
$$

$$
\begin{aligned}
&\Delta(t_1) - \int_{t_0}^{t_1} \Delta'(t)\,dt \\
= &\{\text{By (6.120)}\} \\
&\Delta(t_0).
\end{aligned}
$$

We state this result as a claim.

Claim 6.54.3. *If $t_0 \le t_1$ and $t_1 \in (t_r, t_e]$, then $g(t_0, t_1) \ge \Delta(t_0)$.*

Case 3: $t_1 \in (t_e, \infty)$. In this case, for all t,

$$
\begin{aligned}
g(t, t_1) = &\{\text{By the definition of } g(t, t_1) \text{ in (6.116)}\} \\
&\Delta(t_1) \cdot q^{\frac{t - t_1}{\rho}} \\
= &\{\text{By the definition of } \Delta(t_1) \text{ in (6.51)}\} \\
&\Delta^e(t_1) \cdot q^{\frac{t - t_1}{\rho}} \\
= &\{\text{By the definition of } \Delta^e(t_1) \text{ in (6.47)}\} \\
&\Delta^\ell(t_e) \cdot q^{\frac{t_1 - t_e}{\rho}} \cdot q^{\frac{t - t_1}{\rho}} \\
= &\{\text{Simplifying}\} \\
&\Delta^\ell(t_e) \cdot q^{\frac{t - t_e}{\rho}}. \qquad (6.126)
\end{aligned}
$$

We will establish the following claim.

Claim 6.54.4. *If $t_0 < t_1$ and $t_1 \in (t_e, \infty)$, then $g(t_0, t_1) \ge \Delta(t_0)$.*

We consider two subcases, depending on the value of $t_0 \le t_1$ considered at the beginning of the lemma.

Case 3.1: $t_0 < t_e$. In this case, we have

$$
\begin{aligned}
g(t_0, t_1) = &\{\text{By (6.126) with } t = t_0\} \\
&\Delta^\ell(t_e) \cdot q^{\frac{t_0 - t_e}{\rho}} \\
= &\{\text{By Lemma 6.15}\} \\
&\Delta(t_e) \cdot q^{\frac{t_0 - t_e}{\rho}} \\
= &\{\text{By the definition of } g(t_0, t_e) \text{ in (6.116)}\}
\end{aligned}
$$

$$g(t_0, t_e) \tag{6.127}$$

By Lemma 6.12, $t_e \geq t_r$. We establish Claim 6.54.4 by considering two further subcases, depending on whether $t_e = t_r$ or $t_e > t_r$.

Case 3.1.1: $t_e = t_r$**.** In this case, we have

$$\begin{aligned} g(t_0, t_1) = &\{\text{By (6.127)}\} \\ & g(t_0, t_e) \\ \geq &\{\text{By Claim 6.54.2 with } t_1 = t_e\} \\ & \Delta(t_0). \end{aligned}$$

Thus Claim 6.54.4 holds.

Case 3.1.2: $t_e > t_r$**.** In this case, we have

$$\begin{aligned} g(t_0, t_1) = &\{\text{By (6.127)}\} \\ & g(t_0, t_e) \\ \geq &\{\text{By Claim 6.54.3 with } t_1 = t_e\} \\ & \Delta(t_0). \end{aligned}$$

Thus Claim 6.54.4 holds.

Case 3.2: $t_0 \geq t_e$**.** In this case, we have

$$\begin{aligned} g(t_0, t_1) = &\{\text{By (6.126) with } t = t_0\} \\ & \Delta^{\ell}(t_e) \cdot q^{\frac{t_0 - t_e}{\rho}} \\ = &\{\text{By the definition of } \Delta^{e}(t_0) \text{ in (6.47)}\} \\ & \Delta^{e}(t_0) \\ = &\{\text{By the definition of } \Delta(t) \text{ in (6.51)}\} \\ & \Delta(t_0). \end{aligned}$$

Thus, Claim 6.54.4 holds.

Correctness of Lemma. The correctness of the lemma follows from Claims 6.54.1–6.54.4 with $t_0 = t_2 - \rho$ and $t_1 = t_2$. □

We now bound the value of $x_i^s(s_r) + \Delta\left(t_{i,k}^b\right)$ using Lemma 6.54 when $t_{i,k}^b \in [t_r, t_n)$. Applying this result and Lemma 6.24 will allow us to prove Lemma 6.56, which is similar to Lemma 6.48 for the case that $t_{i,k}^b \in [t_r, t_n)$.

Lemma 6.55. *If $t_{i,k}^c > y_{i,k} + e_{i,k}$ and $t_{i,k}^b \in [t_r, t_n)$, then*

$$x_i^s(s_r) + \Delta\left(t_{i,k}^b\right) \geq \frac{W_{i,k}^r + (m - u_{tot} - 1)C_i + O_{i,k} + L_i \cdot U_i^r \cdot x_i^{p_r}\left(t_{i,k}^b\right)}{u_{tot}}.$$

Proof. We have

$$\begin{aligned}
& x_i^s(s_r) + \Delta\left(t_{i,k}^b\right) \\
\geq\ & \{\text{By Lemma 6.54}\} \\
& x_i^s(s_r) + \Delta\left(t_{i,k}^b - \rho\right) \cdot q \\
=\ & \{\text{By the definition of } q \text{ in (6.49)}\} \\
& x_i^s(s_r) + \Delta\left(t_{i,k}^b - \rho\right) \cdot \frac{\sum_{m-1 \text{ largest}} U_j^r + \max_{\tau_j \in \tau}(L_j \cdot U_j^v \cdot s_r)}{u_{tot}} \\
\geq\ & \{\text{By the definition of ``max''}\} \\
& x_i^s(s_r) + \Delta\left(t_{i,k}^b - \rho\right) \cdot \frac{\sum_{m-1 \text{ largest}} U_j^r + L_i \cdot U_i^v \cdot s_r}{u_{tot}} \\
\geq\ & \{\text{By the definition of } x_i^s(s_r) \text{ in (6.37)}\} \\
& \frac{\sum_{m-1 \text{ largest}}(C_j + U_j^v \cdot s_r \cdot x_j^s(s_r) - S_j) + \sum_{\tau_j \in \tau} S_j + (m - u_{tot} - 1)C_i + O^{rn} + L_i \cdot U_i^v \cdot s_r \cdot x_i^s(s_r)}{u_{tot}} \\
& + \Delta\left(t_{i,k}^b - \rho\right) \cdot \frac{\sum_{m-1 \text{ largest}} U_j^r + L_i \cdot U_i^r \cdot s_r}{u_{tot}} \\
=\ & \{\text{Rearranging}\} \\
& \Bigg(\sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_r \cdot x_j^s(s_r) - S_j) + \sum_{m-1 \text{ largest}} U_j^r \cdot \Delta\left(t_{i,k}^b - \rho\right) \\
& + \sum_{\tau_j \in \tau} S_j + (m - u_{tot} - 1)C_i + O^{rn} + L_i \cdot U_i^v \cdot s_r \cdot \left(x_i^s(s_r) + \Delta\left(t_{i,k}^b - \rho\right)\right)\Bigg) / u_{tot}.
\end{aligned} \tag{6.128}$$

For simplicity, we now consider part of this expression separately.

$$
\begin{aligned}
&\sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_r \cdot x_j^s(s_r) - S_j) + \sum_{m-1 \text{ largest}} U_j^r \cdot \Delta\left(t_{i,k}^b - \rho\right) + \sum_{\tau_j \in \tau} S_j \\
&= \{\text{By the definition of } U_j^r \text{ in (6.28)}\} \\
&\quad \sum_{m-1 \text{ largest}} (C_j + U_j^r \cdot x_j^s(s_r) - S_j) + \sum_{m-1 \text{ largest}} U_j^r \cdot \Delta\left(t_{i,k}^b - \rho\right) + \sum_{\tau_j \in \tau} S_j \\
&\geq \{\text{Rearranging; although the set of tasks in the new first sum may differ from either} \\
&\quad \text{corresponding sum in the previous expression, that can only produce a smaller result}\} \\
&\quad \sum_{m-1 \text{ largest}} \left(C_j + U_j^r \cdot \left(x_j^s(s_r) + \Delta\left(t_{i,k}^b - \rho\right)\right) - S_j\right) + \sum_{\tau_j \in \tau} S_j \\
&= \{\text{By the definition of } x_j^{p_r}\left(t_{i,k}^b\right) \text{ in (6.73)}\} \\
&\quad \sum_{m-1 \text{ largest}} \left(C_j + U_j^r \cdot x_j^{p_r}\left(t_{i,k}^b\right) - S_j\right) + \sum_{\tau_j \in \tau} S_j \\
&= \{\text{By the definition of } W_{i,k}^r \text{ in (6.112)}\} \\
&\quad W_{i,k}^r. \qquad (6.129)
\end{aligned}
$$

Thus,

$$
\begin{aligned}
&x_i^s(s_r) + \Delta\left(t_{i,k}^b\right) \\
&= \{\text{By (6.128) and (6.129)}\} \\
&\quad \frac{W_{i,k}^r + (m - u_{tot} - 1)C_i + O^{rn} + L_i \cdot U_i^v \cdot s_r \cdot \left(x_i^s(s_r) + \Delta\left(t_{i,k}^b - \rho\right)\right)}{u_{tot}} \\
&\geq \{\text{By the definition of } U_i^r \text{ in (6.28) and by Lemma 6.53, because } t_{i,k}^c > y_{i,k} + e_{i,k} \text{ by} \\
&\quad \text{the statement of the lemma, and } t_{i,k}^b \in [t_r, \infty) \text{ is implied by } t_{i,k}^b \in [t_r, t_n)\} \\
&\quad \frac{W_{i,k}^r + (m - u_{tot} - 1)C_i + O_{i,k} + L_i \cdot U_i^r \cdot \left(x_i^s(s_r) + \Delta\left(t_{i,k}^b - \rho\right)\right)}{u_{tot}} \\
&= \{\text{By the definition of } x_i^{p_r}\left(t_{i,k}^b\right) \text{ in (6.73)}\} \\
&\quad \frac{W_{i,k} + (m - u_{tot} - 1)C_i + O_{i,k} + L_i \cdot U_i^r \cdot x_i^{p_r}\left(t_{i,k}^b\right)}{u_{tot}}. \qquad \square
\end{aligned}
$$

The next lemma is identical to Lemma 6.48, but for the case that $t_{i,k}^b \in [t_r, t_n)$.

Lemma 6.56. *If $t_a \in [t_r, t_n)$, $t_a = y_{i,k}$ for some k, $t^c_{i,k} > y_{i,k} + e_{i,k}$, $\tau_{i,k}$ is m-dominant for L_i, $t^b_{i,k} \in [t_r, t_n)$, and (6.52) holds, then $x_i(t_a) = x^s_i(s_r) + \Delta(t_a)$ is x-sufficient.*

Proof. We have

$$\begin{aligned}
& x^s_i(s_r) + \Delta(y_{i,k}) \\
\geq\ & \{\text{By Lemma 6.24 with } t_0 = t^b_{i,k} \text{ and } t_1 = y_{i,k};\ t^b_{i,k} \leq y_{i,k} \text{ by the definition of } t^b_{i,k} \text{ in Definition 6.12}\} \\
& x^s_i(s_r) + \Delta\left(t^b_{i,k}\right) + \phi \cdot (y_{i,k} - t^b_{i,k}) \\
\geq\ & \{\text{By Lemma 6.55, which only has preconditions implied by those of this lemma}\} \\
& \frac{W^r_{i,k} + (m - u_{tot} - 1)C_i + O_{i,k} + L_i \cdot U^r_i \cdot x^{p_r}_i\left(t^b_{i,k}\right)}{u_{tot}} + \phi \cdot (y_{i,k} - t^b_{i,k}) \\
=\ & \{\text{Rearranging}\} \\
& \frac{W^r_{i,k} + u_{tot} \cdot \phi \cdot (y_{i,k} - t^b_{i,k}) + (m - u_{tot} - 1)C_i + O_{i,k} + L_i \cdot U^r_i \cdot x^{p_r}_i\left(t^b_{i,k}\right)}{u_{tot}}. \qquad (6.130)
\end{aligned}$$

For simplicity, we now consider two parts of this expression separately. For the first, we note that by the definition of $t^b_{i,k}$ in Definition 6.12, $t^b_{i,k} \leq y_{i,k} = t_a$. Therefore, (6.72) with $t_2 = t^b_{i,k}$ follows from (6.52). Thus,

$$\begin{aligned}
& W^r_{i,k} + u_{tot} \cdot \phi \cdot (y_{i,k} - t^b_{i,k}) - C_i \\
\geq\ & \{\text{By the definition of } \phi \text{ in (6.46)}\} \\
& W^r_{i,k} + \left(\sum_{\tau_j \in \tau} U^r_j - u_{tot}\right) \cdot (y_{i,k} - t^b_{i,k}) - C_i \\
=\ & \{\text{Rearranging}\} \\
& W^r_{i,k} + \sum_{\tau_j \in \tau} U^r_j \cdot (y_{i,k} - t^b_{i,k}) - u_{tot} \cdot (y_{i,k} - t^b_{i,k}) - C_i \\
\geq\ & \{\text{By Lemma 6.52 using (6.72); the remaining preconditions are implied by those of this lemma, and by} \\
& \text{the definition of } R_{i,k} \text{ in (6.21)}\} \\
& W_{i,k} - R_{i,k} - e_{i,k}. \qquad (6.131)
\end{aligned}$$

And for the second,

$$\begin{aligned}
x_i^{p_r}\left(t_{i,k}^b\right) = &\{\text{By the definition of } x_i^{p_r}\left(t_{i,k}^b\right) \text{ in (6.73)}\}\\
&x_i^s(s_r) + \Delta\left(t_{i,k}^b - \rho\right)\\
\geq &\{\text{By Lemma 6.24 with } t_0 = t_{i,k}^b - \rho \text{ and } t_1 = y_{i,k} - \rho;\ t_{i,k}^b - \rho \leq y_{i,k} - \rho \text{ by}\\
&\text{the definition of } t_{i,k}^b \text{ in Definition 6.12}\}\\
&x_i^s(s_r) + \Delta\left(y_{i,k} - \rho\right)\\
= &\{\text{By the definition of } x_i^{p_r}\left(y_{i,k}\right) \text{ in (6.73)}\}\\
&x_i^{p_r}\left(y_{i,k}\right). \qquad (6.132)
\end{aligned}$$

So that we can use Property 6.11, we establish that

$$\begin{aligned}
t_{i,k}^c > &\{\text{By the statement of the lemma}\}\\
&y_{i,k} + e_{i,k}\\
> &\{\text{Because } e_{i,k} > 0\}\\
&y_{i,k}\\
\geq &\{\text{By the definition of } t_{i,k}^b \text{ in Definition 6.12}\}\\
&t_{i,k}^b\\
\geq &\{\text{By the statement of the lemma}\}\\
&t_r. \qquad (6.133)
\end{aligned}$$

Putting it all together,

$$\begin{aligned}
&x_i^s(s_r) + \Delta\left(y_{i,k}\right)\\
\geq &\{\text{By (6.130)–(6.132)}\}\\
&\frac{W_{i,k} - R_{i,k} + (m - u_{tot})C_i - e_{i,k} + O_{i,k} + L_i \cdot U_j^r \cdot x_i^{p_r}\left(y_{i,k}\right)}{u_{tot}}\\
\geq &\{\text{By Property 6.11 and (6.133)}\}
\end{aligned}$$

$$\frac{W_{i,k} - R_{i,k} + (m - u_{tot} - 1)e_{i,k} + O_{i,k} + L_i \cdot U_j^r \cdot x_i^{p_r}(y_{i,k})}{u_{tot}}$$

$$\geq \{\text{By Lemma 6.42, which only has preconditions that are implied by those of this lemma}\}$$

$$\frac{W_{i,k} - R_{i,k} + (m - u_{tot} - 1)e_{i,k} + O_{i,k} + L_i \cdot e_{i,k}^p}{u_{tot}}$$

$$= \{\text{By the definition of } x_{i,k}^m \text{ in (6.24)}\}$$

$$x_{i,k}^m.$$

By Theorem 6.5, which only has preconditions that are implied by those of this lemma, $x_i(t_a) = x_{i,k}^m$ is x-sufficient. Therefore, by Property 6.4 with $c_0 = x_{i,k}^m$ and $c_1 = x_i^s(s_r) + \Delta(t_a)$, $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient. □

We now combine the results of Lemmas 6.48 and 6.56 into a single lemma that addresses Case E.

Lemma 6.57. *If $t_a \in [t_r, t_n)$, $t_a = y_{i,k}$ for some $\tau_{i,k}$, and $\tau_{i,k}$ is m-dominant for L_i, then $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient.*

Proof. If $t_{i,k}^b \in [0, t_r)$, then the lemma follows from Lemma 6.48. Otherwise, it follows from Lemma 6.56. □

We finally combine the lemmas previously proved in this section to show that $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient for arbitrary $t_a \in [t_r, t_n)$.

Theorem 6.6. *For arbitrary $t_a \in [t_r, t_n)$, $x_i(t_a) = x_i^s(s_r) + \Delta(t_a)$ is x-sufficient.*

Proof. The lemmas referenced in Figure 6.8 exhaustively consider all possible cases for t_a, in some cases assuming (6.52). For any given t_a, (6.52) can be established by inducting over all $y_{j,\ell} \in [t_r, t_a)$, using some lemma referenced in Figure 6.8 to establish that $x_j\left(y_{j,\ell}\right) = x_j^s(s_r) + \Delta\left(y_{j,\ell}\right)$ is x-sufficient for each $\tau_{j,\ell}$. The theorem then follows from the appropriate lemma referenced in Figure 6.8. □

6.3.4 Determining t_n

In this subsection, we provide a condition that the system can use to determine when to return the virtual time clock to normal speed, as our definition of t_n. We then provide a bound on when that condition must occur. Then, in Section 6.3.5, we will prove that $x_i(t) = x_i^s(1)$ is x-sufficient for $t_a \in [t_n, \infty)$.

Definition 6.20. t_d is the earliest time not before t_r such that some CPU is idle and, for each $\tau_{i,k}$ pending and incomplete at t_d, $x_i\left(y_{i,k}\right) = x_i^s(1)$ is x-sufficient.

We will later show that such a time must exist. The results in this section are actually correct for *any* time that satisfies the stated condition. However, for the smallest dissipation bounds, the earliest should be selected.

Definition 6.21. If there are no pending jobs at t_d, then $t_n = t_d$. Otherwise, t_n is the last completion time of any job pending at t_d.

We will prove that, if the system continues to operate without new overload with $s(t) = s_r$, then it will eventually achieve a state where $x_i\left(y_{i,k}\right) = x_i^s(1)$ is x-sufficient for all new $\tau_{i,k}$. We will then prove that, in this state, a CPU will eventually become idle. Such a point in time satisfies the conditions in the definition of t_d in Definition 6.20, unless an earlier time satisfying the same conditions exists. Therefore, by providing a bound on that time, we provide a dissipation bound.

We first provide analysis of a key time, which we will denote t_δ, such that $x_i\left(t_a\right) = x_i^s(1)$ is x-sufficient for $t \in [t_\delta, \infty)$. The following lemma considers the value of $\Delta\left(t_\delta\right)$.

Lemma 6.58. *Let*

$$t_\delta \triangleq \begin{cases} t_r + \frac{\delta - \lambda}{\phi} & \textit{If } \delta > \phi \cdot \frac{\rho}{\ln q} \\ t_e + \frac{\rho}{\ln q}(\ln(\delta) - \ln(\Delta^\ell\left(t_e\right))) & \textit{Otherwise.} \end{cases} \tag{6.134}$$

$t_\delta \geq t_r$ holds. Furthermore, $\Delta\left(t_\delta\right) = \delta$.

Proof. We first note that, by the definition of λ in (6.40),

$$\lambda \geq \delta. \tag{6.135}$$

We consider two cases.

Case 1: $\delta > \phi \cdot \frac{\rho}{\ln q}$. We have

$$\begin{aligned} t_\delta &= \{\text{By the definition of } t_\delta \text{ in (6.134)}\} \\ &\quad t_r + \frac{\delta - \lambda}{\phi} \\ &\geq \{\text{By (6.135), and because } \phi < 0 \text{ by Lemma 6.11}\} \end{aligned}$$

$$t_r. \tag{6.136}$$

Additionally,

$$\begin{aligned}
t_\delta = &\{\text{By the definition of } t_\delta \text{ in (6.134)}\}\\
&t_r + \frac{\delta - \lambda}{\phi}\\
< &\{\text{Because } \delta > \phi \cdot \tfrac{\rho}{\ln q} \text{ by the case we are considering, and } \phi < 0 \text{ by Lemma 6.11}\}\\
&t_r + \frac{\phi \cdot \frac{\rho}{\ln q} - \lambda}{\phi}\\
= &\{\text{Simplifying}\}\\
&t_r + \frac{\rho}{\ln q} - \frac{\lambda}{\phi}\\
= &\{\text{By the definition of } t_e \text{ in (6.48) and by (6.135), and because } \delta > \phi \cdot \tfrac{\rho}{\ln q} \text{ by the case we are}\\
&\text{considering}\}\\
&t_e. && (6.137)
\end{aligned}$$

Thus,

$$\begin{aligned}
\Delta(t_\delta) = &\{\text{By the definition of } \Delta(t_\delta) \text{ in (6.51), (6.136), and (6.137)}\}\\
&\Delta^\ell(t_\delta)\\
= &\{\text{By the definition of } \Delta^\ell(t_\delta) \text{ in (6.45)}\}\\
&\phi \cdot (t_\delta - t_r) + \lambda\\
= &\{\text{By the definition of } t_\delta \text{ in (6.134)}\}\\
&\phi \cdot \left(t_r + \frac{\delta - \lambda}{\phi} - t_r\right) + \lambda\\
= &\{\text{Simplifying}\}\\
&\delta.
\end{aligned}$$

Case 2: $\delta \leq \phi \cdot \frac{\rho}{\ln q}$. We use the following fact.

$$\begin{aligned}
\ln\delta - \ln(\Delta^{\ell}(t_e)) \leq{} & \{\text{Because } \Delta^{\ell}(t_e) \geq \delta \text{ by Lemma 6.15 and the case we are considering}\} \\
& 0. \qquad (6.138)
\end{aligned}$$

Continuing the analysis of Case 2,

$$\begin{aligned}
t_\delta ={} & \{\text{By the definition of } t_\delta \text{ in (6.134)}\} \\
& t_e + \frac{\rho}{\ln q} \cdot (\ln\delta - \ln(\Delta^{\ell}(t_e))) \\
\geq{} & \{\text{by (6.138), and because } \rho > 0 \text{ by Lemma 6.14 and } 0 < q < 1 \text{ by Lemma 6.13}\} \\
& t_e. \qquad (6.139)
\end{aligned}$$

By (6.139) and Lemma 6.12, $t_\delta \geq t_r$. Furthermore,

$$\begin{aligned}
\Delta(t_\delta) ={} & \{\text{By (6.139) and the definition of } \Delta(t_\delta) \text{ in (6.51)}\} \\
& \Delta^{e}(t_\delta) \\
={} & \{\text{By the definition of } \Delta^{e}(t_\delta) \text{ in (6.47)}\} \\
& \Delta^{\ell}(t_e) \cdot q^{\frac{t_\delta - t_e}{\rho}} \\
={} & \{\text{Rewriting}\} \\
& \Delta^{\ell}(t_e) \cdot e^{\frac{\ln q}{\rho}(t_\delta - t_e)} \\
={} & \{\text{By (6.134) and the case we are considering}\} \\
& \Delta^{\ell}(t_e) \cdot e^{\frac{\ln q}{\rho}(t_e + \frac{\rho}{\ln q}(\ln\delta - \ln(\Delta^{\ell}(t_e))) - t_e)} \\
={} & \{\text{Simplifying}\} \\
& \Delta^{\ell}(t_e) \cdot e^{\ln\delta - \ln(\Delta^{\ell}(t_e))} \\
={} & \{\text{Simplifying}\} \\
& \delta. \qquad \square
\end{aligned}$$

We next provide a sufficient condition to ensure that $x_i\left(y_{i,k}\right) = x_i^s(1)$ is x-sufficient for any job $\tau_{i,k}$ pending at t_a. We first define

$$t_n^{pre} \triangleq t_\delta + \rho, \tag{6.140}$$

and prove that $t_n^{pre} > t_r$.

Lemma 6.59. $t_n^{pre} > t_r$.

Proof. We have

$$\begin{aligned} t_n^{pre} &> \{\text{By the definition of } t_n^{pre} \text{ in (6.140), because } \rho > 0 \text{ by Lemma 6.14}\} \\ &\quad t_\delta \\ &\geq \{\text{By Lemma 6.58}\} \\ &\quad t_r. \end{aligned}$$

□

In some parts of our proofs, we will need to use (6.72) with arbitrary $t_2 \in t_a$. The following lemma shows that this is correct.

Lemma 6.60. *For arbitrary* $t_2 \in [t_r, t_n)$, (6.72) *holds.*

Proof. Let $\tau_{j,\ell}$ be an arbitrary job with $y_{j,\ell} \in [t_r, t_2)$. Because $t_2 < t_n$ by the statement of the lemma, $y_{j,\ell} \in [t_r, t_n)$. Therefore, by Theorem 6.6 with $t_a = y_{j,\ell}$, $x_j\left(y_{j,\ell}\right) = x_j^s(s_r) + \Delta\left(y_{j,\ell}\right)$ is x-sufficient. □

The next lemma essentially proves that the system has returned to "normal behavior" for all jobs *pending* at t_n^{pre}.

Lemma 6.61. *If* $t_a \in [t_n^{pre}, t_n)$, $\tau_{i,k}$ *is pending at* t_a, *and* $t_{i,k}^c \leq t_n$, *then* $x_i\left(y_{i,k}\right) = x_i^s(1)$ *is* x*-sufficient.*

Proof. We will use the following facts.

We will use Lemma 6.38 with $t_2 = t_a$. We have

$$\begin{aligned} t_a &> \{\text{Because } \rho > 0 \text{ by Lemma 6.14}\} \\ &\quad t_a - \rho \\ &\geq \{\text{By the statement of the lemma}\} \end{aligned}$$

$$\begin{aligned} & t_n^{pre} - \rho \\ &= \{\text{By the definition of } t_n^{pre} \text{ in (6.140)}\} \\ & t_\delta \\ &> \{\text{By Lemma 6.58}\} \\ & t_r. \end{aligned} \tag{6.141}$$

Thus, because $t_a < t_n$ by the statement of the lemma, $t_a \in [t_r, t_n)$. Therefore, (6.71) holds, and by Lemma 6.60 with $t_2 = t_a$, (6.72) also holds. Therefore, by Lemma 6.38 with $t_2 = t_a$, the following claim holds.

Claim 6.61.1. *$x_i^p(t_a) = x_i^{p_r}(t_a)$ is x^p-sufficient.*

Also, we will use Property 6.11, so we show

$$\begin{aligned} t_{i,k}^c &\geq \{\text{Because } \tau_{i,k} \text{ is pending at } t_a\} \\ & t_a \\ &\geq \{\text{By the statement of the lemma}\} \\ & t_n^{pre} \\ &> \{\text{By Lemma 6.59}\} \\ & t_r. \end{aligned} \tag{6.142}$$

Continuing the proof of Lemma 6.61,

$$\begin{aligned} y_{i,k} &\geq \{\text{By Claim 6.61.1 and the definition of } x^p\text{-sufficient in Definition 6.19}\} \\ & t_a - (x_i^{p_r}(t_a) + e_{i,k}^c(t_a)) \\ &= \{\text{By the definition of } x_i^{p_r}(t_a) \text{ in (6.73)}\} \\ & t_a - (x_i^s(s_r) + \Delta(t_a - \rho) + e_{i,k}^c(t_a)) \\ &\geq \{\text{By Lemma 6.24 with } t_0 = t_r \text{ and } t_1 = t_a - \rho\text{, and by (6.141)}\} \\ & t_a - (x_i^s(s_r) + \Delta(t_r) + e_{i,k}^c(t_a)) \\ &= \{\text{By Lemma 6.20}\} \end{aligned}$$

$$
\begin{aligned}
& t_a - (x_i^s(s_r) + \lambda + e_{i,k}^c(t_a)) \\
\geq\ & \{\text{By Property 6.11 with (6.142), and by the definition of } e_{i,k}^c(t_a) \text{ in Definition 6.2}\} \\
& t_a - (x_i^s(s_r) + \lambda + C_i) \\
\geq\ & \{\text{By the definition of } \rho \text{ in (6.50)}\} \\
& t_a - \rho \\
\geq\ & \{\text{By the definition of } t_a \text{ in the statement of the lemma}\} \\
& t_n^{pre} - \rho \\
=\ & \{\text{By the definition of } t_n^{pre} \text{ in (6.140)}\} \\
& t_\delta. \qquad (6.143)
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
x_i^s(1) =\ & \{\text{Rearranging}\} \\
& x_i^s(s_r) + x_i^s(1) - x_i^s(s_r) \\
\geq\ & \{\text{By the definition of “min”}\} \\
& x_i^s(s_r) + \min_{\tau_j \in \tau}(x_j^s(1) - x_j^s(s_r)) \\
=\ & \{\text{By the definition of } \delta \text{ in (6.41)}\} \\
& x_i^s(s_r) + \delta \\
=\ & \{\text{By Lemma 6.58}\} \\
& x_i^s(s_r) + \Delta(t_\delta) \\
\geq\ & \{\text{By Lemma 6.24 with } t_0 = t_\delta \text{ and } t_1 = y_{i,k}\text{, and by (6.143)}\} \\
& x_i^s(s_r) + \Delta(y_{i,k}). \qquad (6.144)
\end{aligned}
$$

We now establish the following claim.

Claim 6.61.2. *$x_i(y_{i,k}) = x_i^s(s_r) + \Delta(y_{i,k})$ is x-sufficient.*

If $t^c_{i,k} \leq y_{i,k}$, then by Theorem 6.2 with $t_a = y_{i,k}$, $x_i(y_{i,k}) = 0$ is x-sufficient. Furthermore, by Lemma 6.17 with $t_a = y_{i,k}$, $x^s_i(s_r) + \Delta(y_{i,k}) \geq 0$. Therefore, Claim 6.61.2 follows from Property 6.4 with $c_0 = 0$ and $c_1 = x^s_i(s_r) + \Delta(y_{i,k})$.

Alternatively, if $t^c_{i,k} > y_{i,k}$, then we have

$$\begin{aligned} y_{i,k} &< t^c_{i,k} \\ &\leq \{\text{By the statement of the lemma}\} \\ &\quad t_n. \end{aligned}$$

Furthermore, by (6.143) and Lemma 6.58, $y_{i,k} \geq t_r$. Thus, $y_{i,k} \in [t_r, t_n)$. Therefore, Claim 6.61.2 holds by Theorem 6.6 with $t_a = y_{i,k}$.

By Claim 6.61.2, (6.144), and Property 6.4 with $c_0 = x^s_i(s_r) + \Delta(y_{i,k})$ and $c_1 = x^s_i(1)$, $x_i(y_{i,k}) = x^s_i(1)$ is x-sufficient. □

We will now show that an idle instant must occur after t_δ. To do so, we will examine an interval over which more time is available to level C than is used by level-C tasks. We first bound in Lemma 6.62 the time available to level C in an interval starting at t^{pre}_n, with the ending point being arbitrary. Then, in Lemma 6.63, we bound the work executed at level C in an identically defined interval. In Lemma 6.64 we combine these results to show that idleness must occur in a sufficiently long interval.

Lemma 6.62. *If $t_1 \in [t^{pre}_n, t_n)$, then at least*

$$u_{tot} \cdot (t_1 - t^{pre}_n) - \sum_{P_p \in P} \widehat{u_p} \sigma_p$$

units of processor time are available to level C over $[t^{pre}_n, t_1)$.

Proof. By the definition of $\beta_p(t^{pre}_n, t_1)$ in Definition 6.7, the total amount of processor time available to level C over $[t^{pre}_n, t_1)$ is

$$\begin{aligned} &\sum_{P_p \in P} \beta_p(t^{pre}_n, t_1) \\ &\geq \{\text{By (6.9)}\} \end{aligned}$$

$$\sum_{P_p \in P} (\widehat{u_p} \cdot (t_1 - t_n^{pre}) - o_p(t_n^{pre}, t_1))$$

$$\geq \{\text{By Property 6.12, since } t_n^{pre} > t_r \text{ by Lemma 6.59}\}$$

$$\sum_{P_p \in P} (\widehat{u_p} \cdot (t_1 - t_n^{pre}) - \widehat{u_p}\sigma_p)$$

$$= \{\text{Rearranging}\}$$

$$\sum_{P_p \in P} \widehat{u_p} \cdot (t_1 - t_n^{pre}) - \sum_{P_p \in P} \widehat{u_p}\sigma_p$$

$$= \{\text{By the definition of } u_{tot} \text{ in (6.10)}\}$$

$$u_{tot} \cdot (t_1 - t_n^{pre}) - \sum_{P_p \in P} \widehat{u_p}\sigma_p. \qquad \square$$

We now upper bound the amount of work completed by arbitrary τ_i over an identically defined interval. By summing over all tasks, the total amount of work completed at level C over this interval can be derived.

Lemma 6.63. *If $t_1 \in [t_n^{pre}, t_n)$, then at most*

$$2C_i - S_i + U_i^r \cdot x_i^s(1) + U_i^r \cdot (t_1 - t_n^{pre})$$

units of work execute from τ_i over $[t_n^{pre}, t_1)$.

Proof. We will use the following facts.

We will use Lemma 6.38 with $j = i$ and $t_2 = t_n^{pre}$. Because $[t_n^{pre}, t_n)$ is not empty (as it contains t_1), $t_n^{pre} < t_n$. Therefore, by Lemma 6.59, $t_n^{pre} \in [t_r, t_n)$. Thus, (6.71) holds. Furthermore, by Lemma 6.60 with $t_2 = t_n^{pre}$, (6.72) also holds. Therefore,

$$x_i^p\left(t_n^{pre}\right) = x_i^{p_r}\left(t_n^{pre}\right) \text{ is } x^p\text{-sufficient.} \tag{6.145}$$

If a job of τ_i executes in $[t_n^{pre}, t_1)$, then it must have $r_{i,k} < t_1$. Thus, we have

$$v(y_{i,k}) = \{\text{By the definition of } Y_i \text{ in (6.6)}\}$$

$$v(r_{i,k}) + Y_i$$

$$< \{\text{Because } r_{i,k} < t_1\}$$

$$v(t_1)+Y_i.$$

We therefore define $y_{\max}$ as the time such that

$$v(y_{\max}) \triangleq v(t_1)+Y_i, \tag{6.146}$$

so that $y_{i,k} < y_{\max}$ for all $\tau_{i,k}$ executing in $[t_n^{pre}, t_1)$. Observe that

$$\begin{aligned} y_{\max} &\geq \{\text{Because } Y_i \geq 0\} \\ & t_1 \\ &\geq \{\text{By the statement of the lemma}\} \\ & t_n^{pre}. \end{aligned} \tag{6.147}$$

Because $t_1 \in [t_n^{pre}, t_n)$, $t_n^{pre} > t_r$ by Lemma 6.59, and $t_r \geq t_s$ by the definition of t_r in Definition 6.13, the following claim holds.

Claim 6.63.1. *$t_1 \in [t_s, t_n]$ and $t_n^{pre} \in [t_s, t_n]$.*

Continuing the proof of Lemma 6.63, we consider two cases.

Case 1: τ_i has no Pending Job at t_n^{pre}. In this case, all jobs of τ_i that run in $[t_n^{pre}, t_1)$ have $t_n^{pre} < r_{i,k} \leq y_{i,k} < y_{\max}$. Therefore, by the definition of $D_i^e(t_n^{pre}, y_{\max})$ in Definition 6.6, the total work from τ_i that runs in $[t_n^{pre}, t_1)$ is at most

$$\begin{aligned} & D_i^e(t_n^{pre}, y_{\max}) \\ \leq\ & \{\text{By Lemmas 6.32 and 6.34 with } t_0 = t_n^{pre}\text{, and because } t_n^{pre} > t_r \text{ by Lemma 6.59}\} \\ & D_i^C(t_n^{pre}, y_{\max}) \\ \leq\ & \{\text{By Lemma 6.35 with } t_0 = t_n^{pre} \text{ and } t_1 = y_{\max}\text{, by (6.147)}\} \\ & U_i^v \cdot (v(y_{\max}) - v(t_n^{pre})) + S_i \\ =\ & \{\text{By the definition of } y_{\max} \text{ in (6.146)}\} \\ & U_i^v \cdot (v(t_1) + Y_i - v(t_n^{pre})) + S_i \end{aligned}$$

$= \{\text{Rearranging}\}$

$U_i^v \cdot Y_i + S_i + U_i^v \cdot (v(t_1) - v(t_n^{pre}))$

$= \{\text{By the definition of } U_i^v \text{ in (6.27) and the definition of } S_i \text{ in (6.38)}\}$

$C_i \cdot \frac{Y_i}{T_i} + C_i \cdot \left(1 - \frac{Y_i}{T_i}\right) + U_i^v \cdot (v(t_1) - v(t_n^{pre}))$

$= \{\text{Simplifying}\}$

$C_i + U_i^v \cdot (v(t_1) - v(t_n^{pre}))$

$= \{\text{By Lemma 6.29 and Claim 6.63.1, and by the definition of } U_i^r \text{ in (6.28)}\}$

$C_i + U_i^r \cdot (t_1 - t_n^{pre})$

$\leq \{\text{Because } S_i \leq C_i \text{ by the definition of } S_i \text{ in (6.38), and because } U_i^r > 0 \text{ and } x_i^s(1) \geq 0 \text{ by the definition of } x_i^s(1) \text{ in (6.37)}\}$

$2C_i - S_i + U_i^r \cdot x_i^s(1) + U_i^r \cdot (t_1 - t_n^{pre})$.

Case 2: $\tau_{i,\ell}$ is the Earliest Pending Job of τ_i at t_n^{pre}. In this case, we use the following facts.

We will use Lemma 6.40 with $j = i$, $t_2 = t_n^{pre}$, $t_3 = y_{\max}$, $s_{ub} = s_r$, and $x_i^p\left(t_n^{pre}\right) = x_i^{p_r}\left(t_n^{pre}\right)$. Unlike when we used this lemma in Section 6.3.3, we are not trying to cancel out an $e_{i,k}$ term. Therefore, it suffices to use the general case considered in the lemma, rather than the more specific case. We have

$$\begin{aligned} t_2 &= t_n^{pre} \\ &> \{\text{By Lemma 6.59}\} \\ &\quad t_r. \end{aligned} \tag{6.148}$$

Furthermore, we have

$$\begin{aligned} t_2 &= t_n^{pre} \\ &\leq \{\text{By (6.147)}\} \\ &\quad y_{\max} \\ &= t_3. \end{aligned} \tag{6.149}$$

(6.77) holds by (6.148) and (6.149). Because $t_2 = t_n^{pre}$ and $j = i$, (6.78) holds by the case we are considering. Because $s_{ub} = s_r$, (6.79) holds by Property 6.6. We have that $\tau_{i,\ell}$ is pending at t_n^{pre} by the case we are considering and that $t_n^{pre} > t_r$ holds by Lemma 6.59. Thus, by Lemma 6.9 with $t_2 = t_n^{pre}$, $y_{i,\ell} \geq t_n^{pre}$. Thus, because $t_n^{pre} < t_n$ by the statement of the lemma (the interval $[t_n^{pre}, t_n)$ contains t_1), (6.80) holds by Property 6.6. (6.81) holds by (6.145). Thus, by Lemma 6.40,

$$e_{i,\ell}^r(t_n^{pre}) + D_i^e(b_{i,\ell}, y_{\max}) \leq C_i + U_i^v \cdot s_r \cdot x_i^{p_r}(t_n^{pre}) + U_i^v \cdot (v(y_{\max}) - v(t_n^{pre})). \tag{6.150}$$

We also have

$$\begin{aligned}
x_i^{p_r}(t_n^{pre}) &= \{\text{By the definition of } x_i^{p_r}\left(t_n^{pre}\right) \text{ in (6.73)}\} \\
&\quad x_i^s(s_r) + \Delta(t_n^{pre} - \rho) \\
&= \{\text{By the definition of } t_n^{pre} \text{ in (6.140)}\} \\
&\quad x_i^s(s_r) + \Delta(t_\delta) \\
&= \{\text{By Lemma 6.58}\} \\
&\quad x_i^s(s_r) + \delta \\
&\leq \{\text{By the definition of } \delta \text{ in (6.41)}\} \\
&\quad x_i^s(s_r) + x_i^s(1) - x_i^s(s_r) \\
&= \{\text{Simplifying}\} \\
&\quad x_i^s(1).
\end{aligned} \tag{6.151}$$

Continuing the analysis of Case 2, the work in $[t_n^{pre}, t_1)$ is at most $e_{i,\ell}^r(t_n^{pre})$ plus the work contributed by jobs $\tau_{i,k}$ with $b_{i,\ell} \leq r_{i,k} \leq y_{i,k} < y_{\max}$. By the definition of $D_i^e(b_{i,k}, y_{\max})$ in Definition 6.6, the total work from τ_i that runs in $[t_n^{pre}, t_1)$ is at most

$$\begin{aligned}
&e_{i,\ell}^r(t_n^{pre}) + D_i^e(b_{i,\ell}, y_{\max}) \\
&\leq \{\text{By (6.150) and the definition of } U_i^r \text{ in (6.28)}\} \\
&\quad C_i + U_i^r \cdot x_i^{p_r}(t_n^{pre}) + U_i^v \cdot (v(y_{\max}) - v(t_n^{pre})) \\
&= \{\text{By the definition of } y_{\max} \text{ in (6.146)}\}
\end{aligned}$$

$$
\begin{aligned}
& C_i + U_i^r \cdot x_i^{p_r}(t_n^{pre}) + U_i^v \cdot (v(t_1) + Y_i - v(t_n^{pre})) \\
&= \{\text{Rearranging}\} \\
& C_i + U_i^v \cdot Y_i + U_i^r \cdot x_i^{p_r}(t_n^{pre}) + U_i^v \cdot (v(t_1) - v(t_n^{pre})) \\
&= \{\text{By the definition of } U_i^v \text{ in (6.27)}\} \\
& C_i + C_i \cdot \frac{Y_i}{T_i} + U_i^r \cdot x_i^{p_r}(t_n^{pre}) + U_i^v \cdot (v(t_1) - v(t_n^{pre})) \\
&= \{\text{Rearranging}\} \\
& 2C_i - C_i \cdot \left(1 - \frac{Y_i}{T_i}\right) + U_i^r \cdot x_i^{p_r}(t_n^{pre}) + U_i^v \cdot (v(t_1) - v(t_n^{pre})) \\
&= \{\text{By the definition of } S_i \text{ in (6.38)}\} \\
& 2C_i - S_i + U_i^r \cdot x_i^{p_r}(t_n^{pre}) + U_i^v \cdot (v(t_1) - v(t_n^{pre})) \\
&= \{\text{By Lemma 6.29 and Claim 6.63.1, and by the definition of } U_i^r \text{ in (6.28)}\} \\
& 2C_i - S_i + U_i^r \cdot x_i^{p_r}(t_n^{pre}) + U_i^r \cdot (t_1 - t_n^{pre}) \\
&= \{\text{By (6.151)}\} \\
& 2C_i - S_i + U_i^r \cdot x_i^s(1) + U_i^r \cdot (t_1 - t_n^{pre}).
\end{aligned}
$$

□

We now combine these results to show that idleness will happen in a sufficiently long interval starting at t_n^{pre}.

Lemma 6.64. *Let*

$$F \triangleq \frac{\sum_{P_p \in P} \widehat{u_p} \sigma_p + \sum_{\tau_i \in \tau} (2C_i - S_i + U_i^r \cdot x_i^s(1))}{u_{tot} - \sum_{\tau_i \in \tau} U_i^r} \tag{6.152}$$

*(o*F*fset). F is well-defined. Moreover, if $t_1 \in (t_n^{pre} + F, t_n)$, then some CPU is idle for a nonzero interval of time in $[t_n^{pre}, t_1)$.*

Proof. We have

$$
\begin{aligned}
\sum_{\tau_i \in \tau} U_i^r &= \{\text{By the definition of } U_i^r \text{ in (6.28)}\} \\
& \sum_{\tau_i \in \tau} (s_r \cdot U_i^v) \\
&= \{\text{Factoring out } s_r\}
\end{aligned}
$$

$$
\begin{aligned}
& s_r \cdot \sum_{\tau_j \in \tau} U_i^v \\
\leq & \{\text{By Property 6.14}\} \\
& s_r \cdot u_{tot} \\
< & \{\text{By Property 6.6}\} \\
& u_{tot}.
\end{aligned}
$$

Therefore, $u_{tot} - \sum_{\tau_i \in \tau} U_i^r > 0$, and F is well-defined.

We show that the difference between CPUs available to level C and level-C work that completes in $[t_n^{pre}, t_1)$ is positive. Using Lemmas 6.62 and 6.63, this difference is at least

$$
\begin{aligned}
& u_{tot} \cdot (t_1 - t_n^{pre}) - \sum_{P_p \in P} \widehat{u_p} \sigma_p - \sum_{\tau_i \in \tau} (2C_i - S_i + U_i^r \cdot x_i^s(1) + U_i^r \cdot (t_1 - t_n^{pre})) \\
= & \{\text{Rearranging}\} \\
& \left(u_{tot} - \sum_{\tau_i \in \tau} U_i^r \right) \cdot (t_1 - t_n^{pre}) - \left(\sum_{P_p \in P} \widehat{u_p} \sigma_p + \sum_{\tau_i \in \tau} (2C_i - S_i + U_i^r \cdot x_i^s(1)) \right) \\
> & \{\text{By the statement of the lemma}\} \\
& \left(u_{tot} - \sum_{\tau_i \in \tau} U_i^r \right) \cdot (t_n^{pre} + F - t_n^{pre}) - \left(\sum_{P_p \in P} \widehat{u_p} \sigma_p + \sum_{\tau_i \in \tau} (2C_i - S_i + U_i^r \cdot x_i^s(1)) \right) \\
= & \{\text{By the definition of } F \text{ in (6.152)}\} \\
& 0.
\end{aligned}
$$

□

In order to convert between virtual and actual time, will need to use the following lemma, which is like Lemma 6.39, but uses a *lower* bound on $s(t)$, namely s_r, rather than an upper bound.

Lemma 6.65. *If $t_1 \geq t_0$, $t_0 \geq t_s$, and $t_1 \geq t_s$, then $v(t_1) - v(t_0) \geq s_r \cdot (t_1 - t_0)$.*

Proof. We have

$$
\begin{aligned}
v(t_1) - v(t_0) = & \{\text{By Property 6.2}\} \\
& \int_{t_0}^{t_1} s(t)\, dt \\
\geq & \{\text{By Properties 6.6 and 6.7}\}
\end{aligned}
$$

$$\int_{t_0}^{t_1} s_r \, dt$$

$$= \{\text{Rearranging}\}$$

$$s_r \cdot (t_1 - t_0).$$ □

We now use this result, combined with Lemma 6.61 above, to bound t_d.

Lemma 6.66. $t_d \leq t_n^{pre} + F$.

Proof. We use proof by contradiction. Suppose $t_d > t_n^{pre} + F$. Let t_1 be the midpoint between $t_n^{pre} + F$ and t_d. Then,

$$t_1 > t_n^{pre} + F \tag{6.153}$$

and

$$t_1 < t_d \tag{6.154}$$

$$\leq \{\text{By the definition of } t_n \text{ in Definition 6.21}\}$$

$$t_n. \tag{6.155}$$

Then, by Lemma 6.64, there is some time $t_0 \in [t_n^{pre}, t_1)$ such that at least one CPU is idle at t_0.

Note that

$$t_0 < t_1$$

$$< \{\text{By (6.154)}\}$$

$$t_d \tag{6.156}$$

and

$$t_0 \geq t_n^{pre}$$

$$> \{\text{By Lemma 6.59}\}$$

$$t_r. \tag{6.157}$$

We consider two cases, depending on whether $x_i\left(y_{i,k}\right) = x_i^s(1)$ is x-sufficient for all jobs $\tau_{i,k}$ pending at t_0.

Case 1: $x_i\left(y_{i,k}\right) = x_i^s(1)$ **is** x**-sufficient for all** $\tau_{i,k}$ **Pending at** t_0**.** In this case, we have

$$\begin{aligned} t_d &\leq \{\text{By (6.157), the case we are considering, and the definition of } t_d \text{ in Definition 6.20}\} \\ &\quad t_0 \\ &< \{\text{By (6.156)}\} \\ &\quad t_d. \end{aligned}$$

This is a contradiction.

Case 2: There is Some $\tau_{i,k}$ **Pending at** t_0 **for Which** $x_i\left(y_{i,k}\right) = x_i^s(1)$ **is not** x**-sufficient.** By Lemma 6.61 with $t_a = t_0$, we have

$$\begin{aligned} t_{i,k}^c &> t_n \\ &\geq \{\text{By the definition of } t_n \text{ in Definition 6.21}\} \\ &\quad t_d. \end{aligned} \tag{6.158}$$

Furthermore, because $\tau_{i,k}$ is pending at t_0, we have

$$\begin{aligned} r_{i,k} &\leq t_0 \\ &< \{\text{By (6.156)}\} \\ &\quad t_d. \end{aligned} \tag{6.159}$$

Thus, by (6.158) and (6.159), $\tau_{i,k}$ is pending at t_d. However, this contradicts the definition of t_d in Definition 6.20 by the case we are considering. □

Finally, we use the definition of t_d in Definition 6.20 and our bound on it in Lemma 6.66 in order to bound t_n.

Theorem 6.7. $t_n \leq t_n^{pre} + F + \max_{\tau_i \in \tau}\left(\frac{Y_i}{s_r} + x_i^s(1) + C_i\right)$.

Proof. If there are no jobs pending at t_d, then

$$\begin{aligned}
t_n &= \{\text{By the definition of } t_n \text{ in Definition 6.21}\} \\
&\quad t_d \\
&\leq \{\text{By Lemma 6.66}\} \\
&\quad t_n^{pre} + F \\
&\leq \{\text{Because } Y_i \geq 0,\ s_r > 0,\ x_i^s(1) \geq 0 \text{ by the definition of } x_i^s(1) \text{ in (6.37), and } C_i > 0\} \\
&\quad t_n^{pre} + F + \max_{\tau_i \in \tau} \left(\frac{Y_i}{s_r} + x_i^s(1) + C_i \right).
\end{aligned}$$

Otherwise, let $\tau_{i,k}$ be the pending job at t_d with the latest completion time, so that $t_n = t_{i,k}^c$ by Definition 6.21. Because $\tau_{i,k}$ is pending at t_d, by the definition of t_d in Definition 6.20, the following claim holds.

Claim 6.7.1. $x_i(y_{i,k}) = x_i^s(1)$ *is x-sufficient.*

We will establish that

$$y_{i,k} \leq t_d + \frac{Y_i}{s_r}. \tag{6.160}$$

If $y_{i,k} \leq t_d$, then (6.160) holds trivially. Thus, we assume $y_{i,k} > t_d$.

We will use Lemma 6.65. By the definition of t_d in Definition 6.20, $t_d \geq t_r$. Furthermore, by the definition of t_r in Definition 6.13, $t_r \geq t_s$. Thus,

$$t_d \geq t_s. \tag{6.161}$$

We have that $\tau_{i,k}$ is pending at t_d (by our choice of $\tau_{i,k}$) and that $t_d \geq t_r$ holds (by the definition of t_d in Definition 6.20). Therefore, by Lemma 6.9,

$$y_{i,k} \geq t_s. \tag{6.162}$$

Also, we have

$$\begin{aligned}
v(y_{i,k}) - v(t_d) &= \{\text{By the definition of } Y_i \text{ in (6.6)}\} \\
&\quad v(r_{i,k}) + Y_i - v(t_d) \\
&\leq \{\text{Because } \tau_{i,k} \text{ is pending at } t_d\}
\end{aligned}$$

$$\begin{aligned}
& v(t_d) + Y_i - v(t_d) \\
&= \{\text{Rearranging}\} \\
& Y_i. \qquad (6.163)
\end{aligned}$$

Recall that we are assuming that $y_{i,k} > t_d$. Thus, by Lemma 6.65 with $t_0 = t_d$, $t_1 = y_{i,k}$, and (6.161) and (6.162),

$$v(y_{i,k}) - v(t_d) \geq s_r \cdot (y_{i,k} - t_d). \qquad (6.164)$$

By (6.163) and (6.164), $s_r \cdot (y_{i,k} - t_d) \leq Y_i$. Rearranging, (6.160) holds.

Because $\tau_{i,k}$ is pending at t_d, by the definition of t_d in Definition 6.20, $x_i\left(y_{i,k}\right) = x_i^s(1)$ is x-sufficient. We will use Property 6.11, so we show that

$$\begin{aligned}
t_{i,k}^c &\geq \{\text{Because } \tau_{i,k} \text{ is pending at } t_d\} \\
& \quad t_d \\
& \geq \{\text{By the definition of } t_d \text{ in Definition 6.20}\} \\
& \quad t_r. \qquad (6.165)
\end{aligned}$$

Thus, we have

$$\begin{aligned}
t_{i,k}^c &\leq \{\text{By Claim 6.7.1 and the definition of } x\text{-sufficient in Definition 6.8}\} \\
& \quad y_{i,k} + x_i^s(1) + e_{i,k} \\
& \leq \{\text{By Property 6.11 and (6.165)}\} \\
& \quad y_{i,k} + x_i^s(1) + C_i \\
& \leq \{\text{By (6.160)}\} \\
& \quad t_d + \frac{Y_i}{s_r} + x_i^s(1) + C_i \\
& \leq \{\text{By Lemma 6.66}\} \\
& \quad t_n^{pre} + F + \frac{Y_i}{s_r} + x_i^s(1) + C_i \\
& \leq \{\text{By the definition of “max”}\}
\end{aligned}$$

A. $t_a < y_{i,0}$ (Lemma 6.67).

B. $t_a = y_{i,k}$ for some k and $t^c_{i,k} \leq y_{i,k} + e_{i,k}$ (Lemma 6.68).

C. $t_a \in (y_{i,k}, y_{i,k+1})$ for some k (Lemma 6.70).

D. $t_a = y_{i,k}$ for some k, $t^c_{i,k} > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is f-dominant for L_i (Lemma 6.71).

E. $t_a = y_{i,k}$ for some k, $t^c_{i,k} > y_{i,k} + e_{i,k}$, and $\tau_{i,k}$ is m-dominant for L_i (Lemma 6.73).

Figure 6.9: Cases considered when proving that $x_i(t_a) = x^s_i(1)$ is x-sufficient for $t_a \in [t_n, \infty)$

$$t^{pre}_n + F + \max_{\tau_i \in \tau} \left(\frac{Y_i}{s_r} + x^s_i(1) + C_i \right). \qquad \square$$

6.3.5 Proving that $x_i(t_a) = x^s_i(1)$ is x-sufficient for $t_a \in [t_n, \infty)$

In this subsection, we demonstrate that $x_i(t_a) = x^s_i(1)$ is x-sufficient for $t_a \in [t_n, \infty)$. Observe that unlike in Section 6.3.2, our choice of $x_i(t_a)$ does not depend on the specific value of t_a. We will consider the same cases that have been considered in previous sections, as now depicted in Figure 6.9. To make the induction easier, we actually reason about $[t_d, \infty)$ rather than $[t_n, \infty)$. However, because $t_d \leq t_n$ by the definition of t_n in Definition 6.21, our results also apply to $[t_n, \infty)$

In an analogous fashion to (6.52), we provide the equivalent for the analysis of $[t_d, \infty)$. We will use (6.166) for induction, just as we used (6.52) previously.

$$\text{For all } \tau_{j,\ell} \text{ with } y_{j,\ell} \in [t_d, t_a),\ x_i(y_{j,\ell}) = x^s_j(1) \text{ is } x\text{-sufficient.} \tag{6.166}$$

We first consider Case A in Figure 6.9, in which $t_a < y_{i,0}$.

Lemma 6.67. *If $t_a < y_{i,0}$, then $x_i(t_a) = x^s_i(1)$ is x-sufficient.*

Proof. If $t_a < y_{i,0}$, then by Theorem 6.1, which has the same precondition as this lemma, $x_i(t_a) = 0$ is x-sufficient. Furthermore, by the definition of $x^s_i(1)$ in (6.37), $x^s_i(1) \geq 0$. Therefore, by Property 6.4 with $c_0 = 0$ and $c_1 = x^s_i(1)$, $x_i(t_a) = x^s_i(1)$ is x-sufficient. □

The analysis of Case B in Figure 6.9 is simple, as was the case when analyzing the analogous case for $t_a \in [t_r, t_n)$.

Lemma 6.68. *If $t_a \in [t_d, \infty)$, $t_a = y_{i,k}$ for some k, and $t^c_{i,k} \le y_{i,k} + e_{i,k}$, then $x_i(t_a) = x^s_i(1)$ is x-sufficient.*

Proof. By Theorem 6.2, which only has preconditions implied by those of this lemma, $x_i(t_a) = 0$ is x-sufficient. Furthermore, by the definition of $x^s_i(1)$ in (6.37), $x^s_i(1) \ge 0$. Therefore, by Property 6.4 with $c_0 = 0$ and $c_1 = x^s_i(1)$, $x_i(t_a) = x^s_i(1)$ is x-sufficient. □

We will next consider Case C in Figure 6.9, in which $t_a \in (y_{i,k}, y_{i,k+1})$ for some k. For many of the results in this section, including our analysis of Case C, we will inductively use (6.166). It so happens, as we show in Lemma 6.69, that a similar result can actually be extended to jobs with PPs earlier than t_d, if considering only jobs *pending* at some $t_2 \in [t_d, \infty)$. We now demonstrate this fact.

Lemma 6.69. *If*

$$t_2 \in [t_d, \infty) \tag{6.167}$$

and

$$\text{For all } \tau_{j,\ell} \text{ with } y_{j,\ell} \in [t_d, t_2),\ x_i\left(y_{j,\ell}\right) = x^s_i(1) \text{ is x-sufficient,} \tag{6.168}$$

then $x_j\left(y_{j,\ell}\right) = x^s_j(1)$ is x-sufficient for all *jobs $\tau_{j,\ell}$ pending at t_2 with $y_{j,\ell} \in (-\infty, t_2)$.*

Proof. We consider an arbitrary $\tau_{j,\ell}$ with $y_{j,\ell} \in (-\infty, t_2)$. We consider two cases, depending on the value of $r_{j,\ell}$.

Case 1: $r_{j,\ell} \in (-\infty, t_d)$. In this case, because $\tau_{j,\ell}$ is pending at $t_2 \ge t_d$, $\tau_{j,\ell}$ is pending at t_d. Therefore, the lemma follows from the definition of t_d in Definition 6.20.

Case 2: $r_{i,\ell} \in [t_d, t_2)$. In this case, because $y_{j,\ell} \ge r_{j,\ell}$, $y_{j,\ell} \in [t_d, t_2)$. Thus, the lemma is true by (6.168). □

We now address Case C directly.

Lemma 6.70. *If $t_a \in [t_d, \infty)$, $t_a \in (y_{i,k}, y_{i,k+1})$ for some k, and* (6.166) *holds, then $x_i(t_a) = x^s_i(1)$ is x-sufficient.*

Proof. We consider two subcases, depending on whether $\tau_{i,k}$ is still pending at t_a.

Case 1: $\tau_{i,k}$ is no Longer Pending at t_a. In this case,

$$\begin{aligned} t^c_{i,k} &< t_a \\ &< \{\text{Because } x^s_i(1) \ge 0 \text{ by the definition of } x^s_i(1) \text{ in (6.37) and } e_{i,k} > 0\} \end{aligned}$$

$$t_a + x_i^s(1) + e_{i,k}.$$

By the definition of x-sufficient in Definition 6.8, $x_i(t_a) = x_i^s(1)$ is x-sufficient.

Case 2: $\tau_{i,k}$ is Pending at t_a. We will use the following fact.

We use Lemma 6.69 with $t_2 = t_a$. Because $t_2 = t_a$, (6.167) follows from the statement of the lemma and (6.168) follows from (6.166). Thus, by Lemma 6.69 with $j = i$ and $\ell = k$, $x_i(y_{i,k}) = x_i^s(1)$ is x-sufficient. $t_a \in (y_{i,k}, y_{i,k+1})$ holds by the statement of the lemma. Therefore, by Theorem 6.3, $x_i(t_a) = \max\{0,\ x_i^s(1) - (t_a - y_{i,k})\}$ is x-sufficient. Additionally,

$$\begin{aligned} x_i^s(1) \geq &\ \{\text{Because } x_i^s(1) > 0 \text{ and } t_a > y_{i,k}\} \\ &\max\{0,\ x_i^s(1) - (t_a - y_{i,k})\}. \end{aligned}$$

Therefore, by Property 6.4 with $c_0 = \max\{0, x_i^s(1) - (t_a - y_{i,k})\}$, and $c_1 = x_i^s(1)$, $x_i(t_a) = x_i^s(1)$ is x-sufficient.

□

We now address Case D, in which $t_a = y_{i,k}$ for some k and $\tau_{i,k}$ is f-dominant for L_i.

Lemma 6.71. *If $t_a \in [t_d, \infty)$, $t_a = y_{i,k}$ for some $\tau_{i,k}$, $t_{i,k}^c > y_{i,k} + e_{i,k}$, $\tau_{i,k}$ is f-dominant for L_i, and* (6.166) *holds, then $x_i(t_a) = x_i^s(1)$ is x-sufficient.*

Proof. We will use the following facts.

We will use Lemma 6.26. By the definition of t_d in Definition 6.20, $t_d \geq t_r$. Therefore, by the statement of the lemma, $t_a \in [t_r, \infty)$. The remaining preconditions of Lemma 6.26 are directly implied by those of this lemma. Thus, the following facts hold.

$$k > 0 \tag{6.169}$$

$$t_{i,k-1}^c > t_r \tag{6.170}$$

Furthermore, by the definition of f-dominant for L_i in Definition 6.10, $t_{i,k-1}^c > y_{i,k}$. Also,

$$\begin{aligned} r_{i,k-1} < &\ \{\text{By our task model}\} \\ &r_{i,k} \end{aligned}$$

$$\begin{aligned}&\leq \{\text{By our task model}\}\\ &\quad y_{i,k}.\end{aligned}$$

Therefore, the following claim holds.

Claim 6.71.1. *$\tau_{i,k-1}$ is pending at $y_{i,k}$.*

Lemma 6.28 describes the relationship between $y_{i,k-1}$ and $y_{i,k}$ in virtual time. We now bound the difference between the two in actual time.

$$\begin{aligned}(y_{i,k} - y_{i,k-1}) &\geq \{\text{By Lemma 6.39 with } s_{ub} = 1\}\\ &\quad v(y_{i,k}) - v(y_{i,k-1})\\ &\geq \{\text{By Lemma 6.28}\}\\ &\quad v(y_{i,k-1}) + T_i - v(y_{i,k-1})\\ &= \{\text{Rearranging}\}\\ &\quad T_i.\end{aligned}$$

Rearranging,

$$y_{i,k-1} \leq y_{i,k} - T_i. \tag{6.171}$$

We will use Lemma 6.69 with $t_2 = y_{i,k}$, $j = i$, and $\ell = k-1$. We have

$$\begin{aligned}t_2 &= y_{i,k}\\ &= \{\text{By the statement of the lemma}\}\\ &\quad t_a \qquad (6.172)\\ &\geq \{\text{By the statement of the lemma}\}\\ &\quad t_d.\end{aligned}$$

Therefore, (6.167) holds. (6.168) holds by (6.166) and (6.172). Thus, by Claim 6.71.1 and Lemma 6.69, $x_i(y_{i,k-1}) = x_i^s(1)$ is x-sufficient. Thus, by the definition of x-sufficient in Definition 6.8,

$$t^c_{i,k-1} \leq y_{i,k-1} + x_i^s(1) + e_{i,k-1}. \tag{6.173}$$

Continuing the proof of Lemma 6.71, we have

$$\begin{aligned}
x_i^s(1) &= \{\text{Rewriting}\} \\
&\quad y_{i,k-1} + x_i^s(1) + C_i - y_{i,k-1} - C_i \\
&\geq \{\text{By (6.170) and Property 6.11}\} \\
&\quad y_{i,k-1} + x_i^s(1) + e_{i,k-1} - y_{i,k-1} - C_i \\
&\geq \{\text{By (6.173)}\} \\
&\quad t^c_{i,k-1} - y_{i,k-1} - C_i \\
&\geq \{\text{By (6.171)}\} \\
&\quad t^c_{i,k-1} - y_{i,k} + T_i - C_i \\
&\geq \{\text{By the choice of } L_i \text{ in Definition 6.14}\} \\
&\quad t^c_{i,k-1} - y_{i,k} + A_i^{rn}(m - L_i - 1) - C_i \\
&\geq \{\text{By Lemma 6.10 with (6.169) and (6.170), which match (6.30) and (6.31)}\} \\
&\quad t^c_{i,k-1} - y_{i,k} + A_{i,k}(m - L_i - 1) - e_{i,k} \\
&= \{\text{By the definition of } x^f_{i,k} \text{ in (6.15)}\} \\
&\quad x^f_{i,k}.
\end{aligned}$$

Furthermore, by Theorem 6.4, which only has preconditions that are directly implied by those of this lemma, $x_i(t_a) = x_i(y_{i,k}) = x^f_{i,k}$ is x-sufficient. Therefore, by Property 6.4 with $c_0 = x^f_{i,k}$ and $c_1 = x_i^s(1)$, $x_i(t_a) = x_i^s(1)$ is x-sufficient. □

We finally consider Case E in Figure 6.9, in which $t_a = y_{i,k}$ for some k and $\tau_{i,k}$ is m-dominant for L_i. We will reuse many of the lemmas from our analysis of the same case with $t_a \in [t_r, t_n)$. However, we will show in

Lemma 6.72 below that we can use $x_i^p(t) = x_i^s(1)$ in place of $x_i^p(t) = x_i^{p_r}(t)$. The proof of Lemma 6.72, like the proof of Lemma 6.38, is based on Lemma 6.37, which considers $x_i(t)$ for certain values of t.

Lemma 6.72. *Let τ_j be arbitrary. If* (6.167) *and* (6.168) *hold, then $x_j^p(t_2) = x_j^s(1)$ is x^p-sufficient.*

Proof. We will prove the lemma directly from the definition of x^p-sufficient in Definition 6.19. We consider an arbitrary $\tau_{j,\ell}$ pending at t_2. Because $\tau_{j,\ell}$ is arbitrary, we simply need to show that $y_{j,\ell} \geq t_2 - (x_j^s(1) + e_{j,\ell}^c(t_2))$. We consider two cases, depending on the value of $y_{j,\ell}$.

Case 1: $y_{j,\ell} \in (-\infty, t_2)$**.** In this case, by Lemma 6.69 and our assumption that (6.167) and (6.168) hold, $x_j(y_{j,\ell}) = x_j^s(1)$ is x-sufficient. Therefore, by Lemma 6.37, $y_{j,\ell} \geq t_2 - (x_j^s(1) + e_{j,\ell}^c(t_2))$.

Case 2: $y_{j,\ell} \in [t_2, \infty)$**.** In this case,

$$\begin{aligned}
y_{j,\ell} &\geq t_2 \\
&\geq \{\text{Because } x_i^s(1) \geq 0 \text{ by the definition of } x_j^s(1) \text{ in (6.37), and because } e_{j,\ell}^c(t_2) \geq 0\} \\
&\quad t_2 - (x_j^s(1) + e_{j,\ell}^c(t_2))
\end{aligned}$$

Because $\tau_{j,\ell}$ was arbitrary, the lemma holds by the definition of x^p-sufficient in Definition 6.19. □

We now provide the analysis for Case E in Figure 6.9.

Lemma 6.73. *If $t_a = y_{i,k}$ for some k, $t_a \in [t_d, \infty)$, $t_{i,k}^c > y_{i,k} + e_{i,k}$, $\tau_{i,k}$ is m-dominant for L_i, and* (6.166) *holds, then $x_i(t_a) = x_i^s(1)$ is x-sufficient.*

Proof. Because $t_{i,k}^c > y_{i,k} + e_{i,k}$, $\tau_{i,k}$ is pending at $y_{i,k}$. Therefore, if $y_{i,k} = t_d$, the lemma follows from the definition of t_d in Definition 6.20. Thus, because $y_{i,k} = t_a \in [t_d, \infty)$, for the remainder of this proof we assume that

$$y_{i,k} > t_d. \tag{6.174}$$

We will use the following facts.

We first prove by contradiction that $t_{i,k}^b \geq t_d$. Suppose (for the purpose of contradiction) that $t_{i,k}^b < t_d$. Then, by (6.174), $t_d \in (t_{i,k}^b, y_{i,k})$. However, by the definition of $t_{i,k}^b$ in Definition 6.12, $[t_{i,k}^b, y_{i,k})$ is busy, and by the definition of t_d in Definition 6.20, some CPU is idle at t_d. This is a contradiction, so

$$t_{i,k}^b \geq t_d. \tag{6.175}$$

We also have

$$\begin{aligned} t_{i,k}^b &\leq \{\text{By the definition of } t_{i,k}^b \text{ in Definition 6.12}\} \\ &\quad y_{i,k} \\ &= \{\text{By the statement of the lemma}\} \\ &\quad t_a, \end{aligned} \tag{6.176}$$

$$\begin{aligned} t_{i,k}^b &\geq \{\text{By (6.175)}\} \\ &\quad t_d \\ &\geq \{\text{By the definition of } t_d \text{ in Definition 6.20}\} \\ &\quad t_r, \end{aligned} \tag{6.177}$$

and

$$\begin{aligned} y_{i,k} &> \{\text{By (6.174)}\} \\ &\quad t_d \\ &\geq \{\text{By the definition of } t_d \text{ in Definition 6.20}\} \\ &\quad t_r. \end{aligned} \tag{6.178}$$

We will use Lemma 6.72 with $t_2 = t_{i,k}^b$. (Although (6.167)–(6.168) are in Lemma 6.69, they are also required by Lemma 6.72.) (6.167) holds by (6.175). (6.168) holds by (6.166) and (6.176). Therefore, by Lemma 6.72,

$$\text{For each } \tau_i, x_j^p\left(t_{i,k}^b\right) = x_j^s(1) \text{ is } x^p\text{-sufficient.} \tag{6.179}$$

We will also use Lemma 6.72 with $t_2 = y_{i,k}$ and $j = i$. In this case, because $y_{i,k} = t_a$, (6.167) holds by the statement of the lemma, and (6.168) holds by (6.166). Therefore, by Lemma 6.72,

$$x_i^p\left(y_{i,k}\right) = x_i^s(1) \text{ is } x^p\text{-sufficient.} \tag{6.180}$$

We will use Lemma 6.51 with $s_{ub} = 1$ and $x_j^p\left(t_{i,k}^b\right) = x_j^s(1)$. (6.106) holds by the statement of the lemma. (6.107) holds by (6.177). Because $s_{ub} = 1$, (6.108) holds. By our definition of $s(t)$ in Section 6.1, $s(t) \leq 1$ holds for all t. Therefore, (6.109) holds. (6.110) holds by (6.179). Thus, by Lemma 6.51,

$$W_{i,k} \leq \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot x_i^s(1) - S_j) + \sum_{\tau_j \in \tau} S_j + \sum_{\tau_j \in \tau} U_j^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) + e_{i,k} - C_i. \tag{6.181}$$

We will also use Lemma 6.41 with $s_{ub} = 1$ and $x_i^p\left(y_{i,k}\right) = x_i^s(1)$. (6.84) holds by (6.178). (6.85) holds by the statement of the lemma. Because $s_{ub} = 1$, (6.86) holds. By our definition of $s(t)$ in Section 6.1, $s(t) \leq 1$ holds for all t. Therefore, (6.87) holds. (6.88) holds by (6.180). Therefore, by Lemma 6.41,

$$e_{i,k}^p \leq U_i^v \cdot x_i^s(1). \tag{6.182}$$

We will use Property 6.11, so we show that

$$\begin{aligned}
t_{i,k}^c >{} & \{\text{By the statement of the lemma}\} \\
& y_{i,k} + e_{i,k} \\
>{} & \{\text{Because } e_{i,k} > 0\} \\
& y_{i,k} \\
>{} & \{\text{By (6.178)}\} \\
& t_r.
\end{aligned} \tag{6.183}$$

Continuing the proof of Lemma 6.73, we have

$$\begin{aligned}
x_i^s(1) \geq{} & \{\text{By the definition of } x_i^s(1) \text{ in (6.37)}\} \\
& \frac{\sum_{m-1 \text{ largest}}(C_j + U_j^v \cdot x_j^s(1) - S_j) + \sum_{\tau_j \in \tau} S_j + (m - u_{tot} - 1)C_i + O^{rn} + L_i \cdot U_i^v \cdot x_i^s(1)}{u_{tot}}.
\end{aligned} \tag{6.184}$$

For simplicity, we separately consider a subset of this expression.

$$
\begin{aligned}
&\sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot x_j^s(1) - S_j) + \sum_{\tau_j \in \tau} S_j - C_i \\
&= \{\text{Rewriting}\} \\
&\quad \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot x_j^s(1) - S_j) + \sum_{\tau_j \in \tau} S_j + \sum_{\tau_j \in \tau} U_j^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) \\
&\quad - \sum_{\tau_j \in \tau} U_j^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) + e_{i,k} - C_i - e_{i,k} \\
&\geq \{\text{By (6.181)}\} \\
&\quad W_{i,k} - \sum_{\tau_j \in \tau} U_j^v \cdot (v(y_{i,k}) - v(t_{i,k}^b)) - e_{i,k} \\
&\geq \{\text{By Property 6.14}\} \\
&\quad W_{i,k} - u_{tot} \cdot (v(y_{i,k}) - v(t_{i,k}^b)) - e_{i,k} \\
&\geq \{\text{By Lemma 6.39 with } t_0 = t_{i,k}^b,\ t_1 = y_{i,k}, \text{ and } s_{ub} = 1;\ y_{i,k} \geq t_{i,k}^b \text{ by the definition of } t_{i,k}^b \text{ in Definition 6.12}\} \\
&\quad W_{i,k} - u_{tot} \cdot (y_{i,k} - t_{i,k}^b) - e_{i,k} \\
&= \{\text{By the definition of } R_{i,k} \text{ in (6.21)}\} \\
&\quad W_{i,k} - R_{i,k} - e_{i,k}. \qquad (6.185)
\end{aligned}
$$

Putting it all together,

$$
\begin{aligned}
x_i^s(1) &\geq \{\text{By (6.184) and (6.185)}\} \\
&\quad \frac{W_{i,k} - R_{i,k} + (m - u_{tot})C_i - e_{i,k} + O^{rn} + L_i \cdot U_i^v \cdot x_i^s(1)}{u_{tot}} \\
&\geq \{\text{By (6.182)}\} \\
&\quad \frac{W_{i,k} - R_{i,k} + (m - u_{tot})C_i - e_{i,k} + O^{rn} + L_i \cdot e_{i,k}^p}{u_{tot}} \\
&\geq \{\text{By (6.183) and Property 6.11}\} \\
&\quad \frac{W_{i,k} - R_{i,k} + (m - u_{tot} - 1)e_{i,k} + O^{rn} + L_i \cdot e_{i,k}^p}{u_{tot}} \\
&\geq \{\text{By Lemma 6.53 and (6.177); } t_{i,k}^c > y_{i,k} + e_{i,k} \text{ holds by the statement of the lemma}\} \\
&\quad \frac{W_{i,k} - R_{i,k} + (m - u_{tot} - 1)e_{i,k} + O_{i,k} + L_i \cdot e_{i,k}^p}{u_{tot}}
\end{aligned}
$$

$$= \{\text{By the definition of } x^m_{i,k} \text{ in (6.24)}\}$$

$$x^m_{i,k}. \tag{6.186}$$

Because $\tau_{i,k}$ is m-dominant for L_i, by Theorem 6.5, which only has preconditions implied by those of this lemma, $x_i(t_a) = x^m_{i,k}$ is x-sufficient. Thus, by (6.186) and Property 6.4 with $c_0 = x^m_{i,k}$ and $c_1 = x^s_i(1)$, $x_i(t_a) = x^s_i(1)$ is x-sufficient. □

We finally combine the lemmas previously proved in this subsection to show that $x_i(t_a) = x^s_i(1)$ is x-sufficient for arbitrary $t_a \in [t_n, \infty)$.

Theorem 6.8. *For arbitrary $t_a \in [t_d, \infty)$, $x_i(t_a) = x^s_i(1)$ is x-sufficient.*

Proof. The lemmas referenced in Figure 6.9 exhaustively consider all possible cases for t_a, in some cases assuming (6.166). For any given t_a, (6.166) can be established by inducting over all $y_{j,\ell} \in [t_r, t_a)$, using some lemma referenced in Figure 6.8 to establish that $x_j(y_{j,\ell}) = x^s_j(s_r) + \Delta(y_{j,\ell})$ is x-sufficient for each. The theorem then follows from the appropriate lemma referenced in Figure 6.8. □

We can also use Theorem 6.8 to determine response-time bounds in the absence of overload, as demonstrated in the next theorem. Recall that t_h is a time after which no overload occurs, as defined more precisely in Properties 6.8 and 6.9.

Theorem 6.9. *If $t_h = 0$, then for arbitrary $\tau_{i,k}$, $t^c_{i,k} \leq y_{i,k} + x^s_i(1) + C_i$.*

Proof. We shift the schedule one time unit later. Because neither jobs nor supply restriction execute within $[0,1)$, Properties 6.8 and 6.9 continue to hold. Because the earliest job release is now at time 1, there are no pending jobs at t_h. Therefore, by the definition of t_r in Definition 6.13,

$$t_r = 0, \tag{6.187}$$

and by the definition of t_d in Definition 6.20,

$$t_d = 0. \tag{6.188}$$

Each job cannot complete or have a PP before time 1 (where the original schedule started), so by (6.187),

$$t^c_{i,k} > t_r, \tag{6.189}$$

and by (6.188)

$$y_{i,k} > t_d. \tag{6.190}$$

Thus,

$$\begin{aligned} t^c_{i,k} \leq & \{\text{By Theorem 6.8, (6.190), and the definition of } x\text{-sufficient in Definition 6.8}\} \\ & y_{i,k} + x^s_i(1) + e_{i,k} \\ \leq & \{\text{By (6.189) and Property 6.11}\} \\ & y_{i,k} + x^s_i(1) + C_i. \end{aligned}$$

□

6.4 Implementation Description

We implemented our scheme by extending the existing MC^2 implementation that was described by Herman et al. (2012). Like that used in the job splitting experiments discussed in Chapter 5, this implementation is based on LITMUS$^{\text{RT}}$. Our implementation consists of two components: the scheduler, which is part of the kernel, and a *monitor program*, which runs in userspace. The kernel reports job releases and job completions to the monitor program and provides a system call that the monitor program can use to change the speed of the virtual clock. The speed of the virtual clock does not change between these calls. The kernel is responsible for implementing virtual time, ensuring that the SVO model's minimum-separation constraints are respected, and making scheduling decisions according to GEL-v scheduling. The monitor program is responsible for determining when virtual-clock speed changes should occur.

In a real system, level-C jobs will often run for less time than their respective level-C PWCETs. Therefore, it may be unnecessarily pessimistic to initiate overload response whenever a job overruns its level-C PWCET. For our implementation, we instead use the following definition.

Definition 6.22. τ_i has a nonnegative *response-time tolerance*, denoted ξ_i, relative to each job's PP. A task *meets* its response-time tolerance if $t^c_{i,k} \leq y_{i,k} + \xi_i$, and *misses* it otherwise.

We slow down the virtual clock only after some job misses its response-time tolerance. Ideally, response-time tolerances should be determined based on the analytical upper bounds of job response times presented in Section 6.3, in order to guarantee that the virtual clock is never slowed down in the absence of overload. Specifically, for each τ_i, we use $\xi_i = x^s_i(1) + C_i$, by Theorem 6.9. However, smaller response-time tolerances

```
Function initialize()
    last_act := now();
    last_virt := 0;
    speed := 1;

Function act_to_virt(act)
    return last_virt + (act − last_act) · speed;

Function virt_to_act(virt)
    return last_act + (virt − last_virt)/speed;

Function schedule_pending_release($\tau_{i,k}$, $v(r_{i,k})$)
    Set release timer to fire at virt_to_act($v(r_{i,k})$);

Function job_release($\tau_{i,k}$)
    $r_{i,k}$ := now();
    $v(y_{i,k})$ := act_to_virt($r_{i,k}$) + $Y_i$;
    $y_{i,k} := \bot$;

Function job_complete($\tau_{i,k}$)
    virt := act_to_virt(now());
    if $y_{i,k} = \bot$ and $v(y_{i,k}) <$ virt then
        $y_{i,k}$ := virt_to_act($v(y_{i,k})$);
    Report $\tau_{i,k}$, $r_{i,k}$, $y_{i,k}$, now(), and whether the level-C ready queue is empty to the monitor program;

Function change_speed(new_speed)
    act := now();
    virt := act_to_virt(act);
    foreach $\tau_{i,k}$ such that $y_{i,k} = \bot$ and $v(y_{i,k}) <$ virt do
        $y_{i,k}$ := virt_to_act($v(y_{i,k})$);
    last_act := act;
    last_virt := virt;
    speed := new_speed;
    foreach $\tau_{i,k}$ such that a pending release has been scheduled for virtual time $v(r_{i,k})$ do
        Reset release timer to fire at virt_to_act($v(r_{i,k})$);
```

Algorithm 6.1: In-kernel functionality used to handle virtual time.

could be used, at the possible expense of returning the virtual-clock speed to normal prematurely and having to slow it down again.

For illustration, Figure 6.1(c) from Section 6.1 actually depicts a system that uses a response-time tolerance of three for each task. Thus, we do not slow down virtual time until some job's completion time is greater than three units of actual time after its PP. At time 18, $\tau_{3,4}$ completes exactly three units after its PP, which is barely within its tolerance, so the virtual clock is not slowed down. However, at time 19, $\tau_{1,3}$ completes four units after its PP, which exceeds the response-time tolerance. Therefore, we slow down the virtual clock at time 19. We define *normal behavior* for a system as the situation in which all jobs meet their response-time tolerances.

Within the kernel, the primary change that we made compared to the prior MC^2 implementation was the use of virtual time at level C. No changes at levels A or B were required. Psuedocode for the changed functionality is provided in Algorithm 6.1. `now()` is a function that always returns the current actual time.

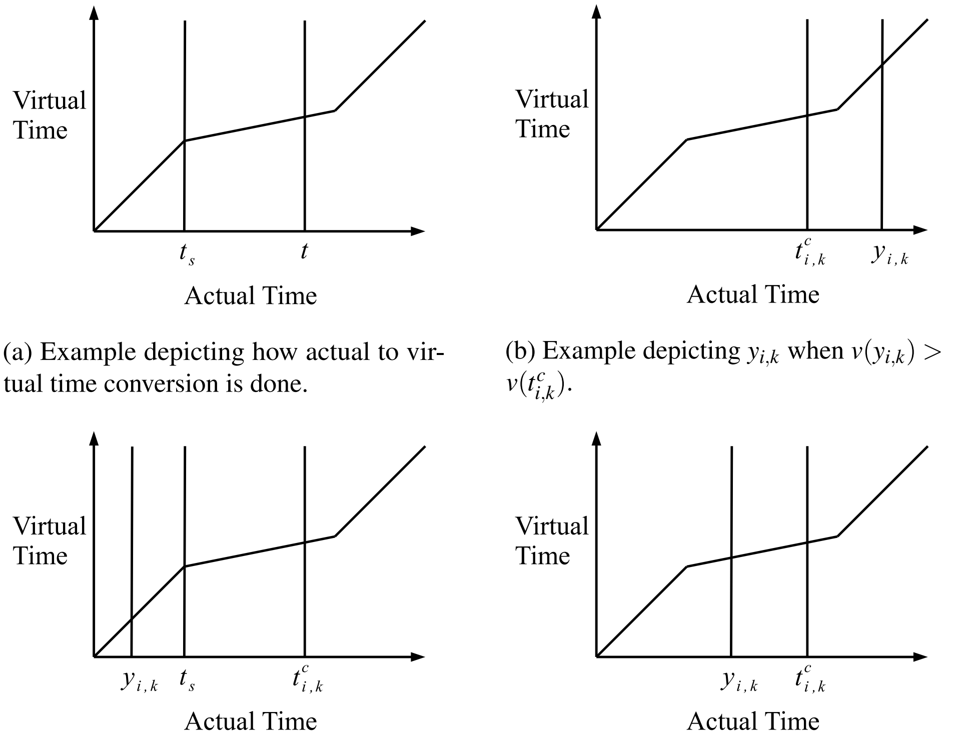
Because the virtual-clock speed is constant between discrete changes, virtual time is a piecewise linear function of actual time, as depicted in Figure 6.10(a), where t_s (*s*peed change) is the latest speed change before arbitrary time t. (This is not necessarily the same t_s as in Section 6.3.) The kernel keeps track of the most recent such actual time as $\mathsf{last\_act}$, the corresponding virtual time as $\mathsf{last\_virt}$, and the current speed of virtual time as speed. These values are initialized in `initialize()` and updated in `change_speed()`.

The convenience function `act_to_virt()` converts an actual time to a virtual time, assuming that $\mathsf{act} > \mathsf{last\_act}$ and that there is no virtual-clock speed change between $\mathsf{last\_act}$ and act. By (6.4), the virtual-clock speed at t is the slope of the line graphed in Figure 6.10(a) with $t_s = \mathsf{last\_act}$, resulting in the simple calculation performed in that function. Similarly, the convenience function `virt_to_act()` converts a virtual time to an actual time, assuming that $\mathsf{virt} > \mathsf{last\_virt}$ and that there is no virtual-clock speed change between $\mathsf{last\_act}$ and `virt_to_act(`virt`)`.

The kernel invokes `schedule_pending_release()` in order to set the release timer for a level-C job. This function uses `virt_to_act(`$v(r_{i,k})$`)` to determine when the timer should fire. This time could be incorrect if the virtual-clock speed is changed before the timer fires, but in that case `change_speed()` will update the timer to fire at the correct time.

When a job release actually occurs, `job_release()` is called. This function determines the scheduling priority of $\tau_{i,k}$, which is simply the virtual time $v(y_{i,k})$ because the actual time $y_{i,k}$ is not known until $y_{i,k}$ occurs (because the virtual-clock speed may change). However, recall that the definition of "response-time tolerance" in Definition 6.22 is based on the actual time $y_{i,k}$. Therefore, it will generally be necessary for the kernel to determine $y_{i,k}$ and return it to the monitor program. Initially, the kernel uses the placeholder $\perp$, to indicate that $y_{i,k}$ has not yet occurred. There are three cases for when $y_{i,k}$ could occur relative to $t^c_{i,k}$, as depicted in Figure 6.10(b)–(d).

If $t^c_{i,k} \le y_{i,k}$, as depicted in Figure 6.10(b), then $\tau_{i,k}$ meets its response-time tolerance (which was defined in Definition 6.22 to be nonnegative) by definition. Therefore, it is sufficient to return $\perp$ to the monitor program in this situation.



(a) Example depicting how actual to virtual time conversion is done.

(b) Example depicting $y_{i,k}$ when $v(y_{i,k}) > v(t^c_{i,k})$.

(c) Example depicting $y_{i,k}$ when $v(y_{i,k}) < v(t^c_{i,k})$ and at least one speed change occured between $y_{i,k}$ and $t^c_{i,k}$.

(d) Example depicting $y_{i,k}$ when $v(y_{i,k}) < v(t^c_{i,k})$ and no speed change occured between $y_{i,k}$ and $t^c_{i,k}$.

Figure 6.10: Examples illustrating virtual time computations in the kernel.

If $t^c_{i,k} > y_{i,k}$ and the speed of the virtual clock changes at least once between $y_{i,k}$ and $t^c_{i,k}$, then this scenario is depicted in Figure 6.10(c), where t_s now refers to the first virtual-clock speed change *after* $y_{i,k}$. In this case, $y_{i,k}$ is computed when change_speed() is called at time t_s.

If $t^c_{i,k} > y_{i,k}$ and the speed of the virtual clock does not change between $y_{i,k}$ and $t^c_{i,k}$, then this scenario is depicted in Figure 6.10(d). In this case, $y_{i,k}$ is computed when job_complete($\tau_{i,k}$) is called.

When any $\tau_{i,k}$ completes, job_complete() performs the just-mentioned check and notifies the monitor program.

When the monitor program requests a virtual-clock speed change, change_speed() is called. This function performs the updates mentioned above and also updates last_act, last_virt, and speed so that virt_to_act() and act_to_virt() remain correct.

The purpose of the userspace monitor program is both to detect when response-time tolerances are missed, and to detect when the system has returned to normal behavior. By Theorem 6.8, once we have

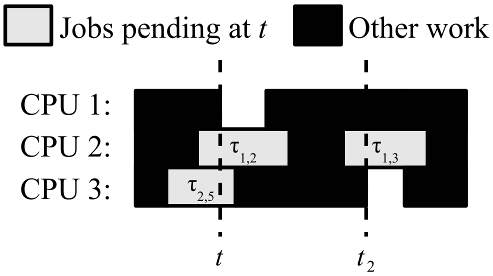


Figure 6.11: Illustration of “idle normal instant.” If all jobs pending at t meet their response-time tolerances, then t is an idle normal instant. t_2 is also referenced below.

detected t_n (see Definition 6.21, which occurs when the last job pending at t_d (see Definition 6.20) completes, the system has returned to normal behavior, and the virtual clock can be returned to normal speed. Therefore, we define an “idle normal instant ” such that t_d is the earliest idle normal instant when each $\xi_i = x_i^s(1) + C_i$, if each level-C job runs for its level-C PWCET. Definition 6.23 is illustrated in Figure 6.11.

Definition 6.23. Arbitrary time t is an *idle normal instant* if some processor is idle at t and all jobs pending at t meet their (normal) response-time tolerances.

The general structure of a userspace monitor program is presented in Algorithm 6.2. A significant portion of the code is intended to detect the earliest possible idle normal instant. We define the following definition, which is closely related to the definition of “idle normal instant ” in Definition 6.23.

Definition 6.24. t is a *candidate idle instant* at time $t_2 \geq t$ if some processor is idle at t and any job pending at t either meets its response-time tolerance or is still pending at t_2.

In Figure 6.11, t is a candidate idle instant at t_2 even if $\tau_{1,3}$ misses its response-time tolerance, as long as $\tau_{1,2}$ and $\tau_{2,5}$ meet their response-time tolerances.

The following theorem shows that we may consider only one candidate idle instant at any given time and still find the earliest idle normal instant. In Figure 6.11, t_2 was selected as a time when a processor becomes idle, in order to illustrate this theorem.

Theorem 6.10. *If t is a candidate idle instant at t_2 and t_2 is an idle normal instant, then t is an idle normal instant.*

Proof. Because t is a candidate idle instant, by Definition 6.24, every job pending at t that is no longer pending at t_2 meets its response-time tolerance. Furthermore, because t_2 is an idle normal instant, by Definition 6.23, every job that is still pending at t_2 meets its response-time tolerance. Therefore, every job pending at t meets its response-time tolerance. Furthermore, because t is a candidate idle instant, by Definition 6.24, some processor is idle at t. Therefore, by Definition 6.23, t is an idle normal instant. □

In order to detect an idle normal instant, we maintain as idle_cand the earliest candidate idle instant and as pend_idle_cand the set of incomplete jobs pending at idle_cand. If there is no current candidate idle instant, then the placeholder $\perp$ is used instead. So that the monitor program can determine pend_idle_cand when a candidate idle instant is detected, it always maintains as pend_now the set of jobs currently pending. A job is added to pend_now whenever it is released, in `on_job_release()`. A job is removed from pend_now as soon as it has completed, in `on_job_complete()`.

In the definition of `job_complete()` in Algorithm 6.1, the kernel reports to the monitor program whether the ready queue is empty. The purpose for this reporting is that if the ready queue is empty, the processor that just completed $\tau_{i,k}$ has become idle. This fact is exploited in both `init_recovery()` and `on_job_complete()` in order to detect candidate idle instants.

The function `init_recovery()` initializes *recovery mode*, the process of finding an idle normal instant. Because recovery mode is always initiated as a result of a job missing its response-time tolerance, and such a miss is detected when the job completes, there is a relevant job completion time comp_time. As discussed above, if the ready queue was empty at comp_time, as indicated by queue_empty, then a processor became idle at comp_time. Therefore, by the definition of "candidate idle instant " in Definition 6.24, comp_time is a candidate idle instant. This case is handled in Lines 2–4. Otherwise, no candidate idle instant has yet been detected, as handled in Lines 5–7.

As discussed above, the function `on_job_release()` simply updates the set pend_now of currently pending jobs.

The function `on_job_complete()` first, in Line 9, updates pend_now as discussed above. Then, Lines 10–11 consider a response-time tolerance miss, regardless of whether the monitor program is currently in recovery mode. If such a miss occurs, then the function `handle_miss()` is called. The particular implementation of this function differs among the monitor programs we consider, and is discussed later. Lines 12–23 execute only when the monitor program is in recovery mode. Lines 12–17 execute if there is already a candidate idle instant under consideration. Lines 13–15 execute if $\tau_{i,k}$ has missed its response-time tolerance, in which case any prior candidate idle instant is no longer a candidate idle instant. On the other hand, Lines 16–17 execute when $\tau_{i,k}$ has met its response-time tolerance. In this case, we remove $\tau_{i,k}$ from the set pend_idle_cand of still-pending jobs that were pending at idle_cand. Lines 18–20 consider the case that comp_time has become the earliest candidate idle instant. This could happen either because a processor just became idle, or because a previous candidate idle instant was just discarded in Lines 13–15 while a processor was idle. In either case, we start considering comp_time as a candidate idle instant. Finally, in Lines 21–23 we consider the case that there is an existing candidate idle instant, but the set pend_idle_cand is empty. Whenever this situation occurs, either because the last job in pend_idle_cand was removed in Line 17 or because the set of pending jobs considered in Line 20 was empty, the system exits recovery mode.

Our first userspace monitor program, SIMPLE, is depicted in Algorithm 6.3. It is given the response-time tolerances desired for the tasks and a virtual time speed $s(t)$ used for overload recovery. When a response-time tolerance miss is detected while the system is not in recovery mode, it simply slows down the virtual clock and starts recovery mode.

Our second userspace monitor program, ADAPTIVE, is depicted in Algorithm 6.4. It allows a value of $s(t)$ to be determined at runtime, selecting a smaller value for a more significant response-time tolerance miss. This minimizes the impact on the system when only a minor response-time tolerance miss has occured, but provides a more drastic response when a larger miss has occured. The monitor accepts an *aggressiveness factor a* in addition to the set of response-time tolerances, providing additional tuning. Once a response-time tolerance violation is detected, the monitor maintains the invariant that $s(t) = a \cdot \min((Y_i + \xi_i)/R_{i,k})$, where the min is over all jobs with $t^c_{i,k}$ after recovery mode last started. Thus, it chooses the speed based on the largest observed response time since recovery mode started.

6.5 Experiments

When a designer provisions an MC system, he or she should select level-C PWCETs that will be infrequently violated. Therefore, in the most common cases, overload conditions should be inherently transient, and it should be possible to return the system to normal operation relatively quickly. Therefore, our experiments consist of transient overloads rather than continuous overloads.

We ran experiments on a system with one quad-core 920-i7 CPU at 2.67 GHz, with 4GB of RAM. We generated 20 task sets, using a methodology similar to that used by Herman et al. (2012), which used task systems designed to mimic avionics. As in (Herman et al., 2012), we assumed that each task's level-B PWCET is ten times its level-C PWCET, and that its level-A PWCET is twenty times its level-C PWCET.

```
Function init_recovery(comp_time, queue_empty)
    recovery_mode := true;
    if queue_empty then
        idle_cand := comp_time;
        pend_idle_cand := pend_now;
    else
        idle_cand := $\bot$;
        pend_idle_cand := $\{\}$;

Function on_job_release($\tau_{i,k}$)
    Add $\tau_{i,k}$ to pend_now;

Function on_job_complete($\tau_{i,k}$, $r_{i,k}$, $y_{i,k}$, comp_time, queue_empty)
    Remove $\tau_{i,k}$ from pend_now;
    if comp_time $- y_{i,k} > \xi_i$ then
        handle_miss($\tau_{i,k}$, $r_{i,k}$, $y_{i,k}$, comp_time, queue_empty);
    if recovery_mode and idle_cand $\neq \bot$ then
        if comp_time $- y_{i,k} > \xi_i$ then
            idle_cand := $\bot$;
            pend_idle_cand := $\{\}$;
        else
            Remove $\tau_{i,k}$ from pend_idle_cand;
    if recovery_mode and idle_cand $= \bot$ and queue_empty then
        idle_cand := comp_time;
        pend_idle_cand := pend_now;
    if recovery_mode and idle_cand $\neq \bot$ and pend_idle_cand $= \{\}$ then
        change_speed(1);
        recovery_mode := false;
```

Algorithm 6.2: Userspace monitor algorithms common to SIMPLE and ADAPTIVE.

```
Function handle_miss($\tau_{i,k}$, $r_{i,k}$, $y_{i,k}$, comp_time, queue_empty)
    if not recovery_mode then
        change_speed($s(t)$);
        init_recovery(comp_time, queue_empty);
```

Algorithm 6.3: Specific userspace implementation for SIMPLE.

```
Function handle_miss($\tau_{i,k}$, $r_{i,k}$, $y_{i,k}$, comp_time, queue_empty)
    if not recovery_mode then
        current_speed := 1;
        init_recovery(comp_time, queue_empty);
    new_speed := $a \cdot (Y_i + \xi_i)/($comp_time $- r_{i,k})$;
    if new_speed < current_speed then
        change_speed(new_speed);
        current_speed := new_speed;
```

Algorithm 6.4: Specific userspace implementation for ADAPTIVE.

	Level A	Level B	Level C
Level A	100%	50%	5%
Level B	0%	50%	5%
Level C	0%	0%	65%

(a) Percentage of the system occupied by the level specified in each row, when all jobs execute for the PWCET specified in the column.

	Level A	Level B	Level C
Level A	$(0.1,0.4)$	$(0.2,0.8)$	$(2.0,8.0)$
Level B	$(0.05,0.2)$	$(0.1,0.4)$	$(1.0,4.0)$
Level C	$(0.005,0.02)$	$(0.01,0.04)$	$(0.1,0.4)$

(b) Uniform utilization distribution for a task at the level specified in each row, when measured using the PWCET specified in the column.

Level A	Randomly from $\{25\text{ ms}, 50\text{ ms}, 100\text{ ms}\}$.
Level B	Random multiple of the largest level-A period on the same CPU, capped at 300 ms.
Level C	Random multiple of 5 ms between 10 ms and 100 ms, inclusive.

(c) Selection of periods for tasks at each level, which is independent of the level of the PWCETs.

Figure 6.12: Parameters used for experiments.

The parameters we used are specified in Figure 6.12. We generated task systems where levels A and B each occupy 5% of the system's processor capacity and level C occupies 65% of the system's capacity, *assuming that all jobs at all levels execute for their level-C PWCETs.* As a property of the relative sizes of the PWCETs at different levels, and as depicted in Figure 6.12(a), when all tasks execute for their level-B PWCETs or above, there is not guaranteed to be any capacity left for level C.

At levels A and B, we generated tasks on one CPU at a time, using 5% of each CPU's capacity for level A (assuming level-C execution times) and 5% for level B (again assuming level-C execution times), as depicted in Figure 6.12(a). For level-A tasks, we selected periods randomly from the set $\{25\text{ ms}, 50\text{ ms}, 100\text{ ms}\}$, and for level-B tasks, we selected random multiples of the largest level-A period on the same CPU, capped at 300 ms, as depicted in Figure 6.12(c). We then selected, for each task, a utilization (at its own criticality level) uniformly from $(0.1, 0.4)$. This is the "uniform medium" distribution from prior work, e.g., (Brandenburg, 2011). For utilization at level C, the resulting choice is scaled by $1/20$ for level-A tasks and $1/10$ for level-B tasks, as depicted in Figure 6.12(b). When a task would not fit within the allocated capacity for its criticality level, its utilization was scaled down to fit. Each task was then assigned a level-C PWCET based on multiplying its level-C utilization by its period.

At level C, we selected periods that were multiples of 5 ms between 10 ms and 100 ms, inclusive, as depicted in Figure 6.12(c). We used uniform medium utilizations, as at levels A and B, and as depicted in Figure 6.12(b). As we did with levels A and B, we scaled down the utilization of the last task to fit. Y_i was selected for each level-C task using G-FL, which provides better response time bounds than G-EDF, as discussed in Chapter 3. To determine response-time tolerances, we used the analytical bounds described in Section 6.3, assigning to each task $\xi_i = x_i^s(1) + C_i$.

We tested the following overload scenarios:

- (SHORT) - All jobs at levels A, B, and C execute for their level-B PWCETs for 500 ms, and then execute for their level-C PWCETs afterward.
- (LONG) - All jobs at levels A, B, and C execute for their level-B PWCETs for one second, and then execute for their level-C PWCETs afterward.
- (DOUBLE) - All jobs at levels A, B, and C execute for their level-B PWCETs for 500 ms, execute for their level-C PWCETs for one second, execute for their level-B PWCETs for another 500 ms, and then execute for their level-C PWCETs afterward.

As can be seen from Figure 6.12(a), these represent particularly pessimistic scenarios in which all CPUs are occupied by level-A and -B work for almost all of the time during the overload.

For each overload scenario, we used SIMPLE with $s(t)$ choices from 0.2 to 1 in increments of 0.2. The choice of $s(t) = 1$ does not use our overload management techniques at all and provides a baseline for comparison. We also used ADAPTIVE with a choices from 0.2 to 1.0 in increments of 0.2. We then recorded the minimum virtual-time speed (to analyze the behavior of ADAPTIVE) and the amount of time from when the last overload stopped until the virtual-time clock was returned to normal. We then averaged each result over all generated task sets.

In Figure 6.13, we depict the average dissipation time using SIMPLE with respect to the choice of $s(t)$ during recovery. Additionally, we depict error bars for 95% confidence intervals. Under LONG, dissipation times are approximately twice as long as under SHORT. This is to be expected, because overhead occurs for twice as long. Under DOUBLE, dissipation times are bigger than under SHORT for $s(t) = 1$, but nearly identical for smaller choices of $s(t)$. This occurs because dissipation time is measured from the end of the *second* (and final) interval during which overload occurs. For sufficiently small choices of $s(t)$, the system usually recovers completely before the second interval of overload starts, and that interval is the same length as in SHORT. In any case, a reduction of at least 50% of the dissipation time can be achieved with a choice of $s(t) = 0.6$, and with that choice, the dissipation time is less than twice the length of the interval during which overload occurs. Smaller choices of $s(t)$ have diminishing returns, with only a small improvement in

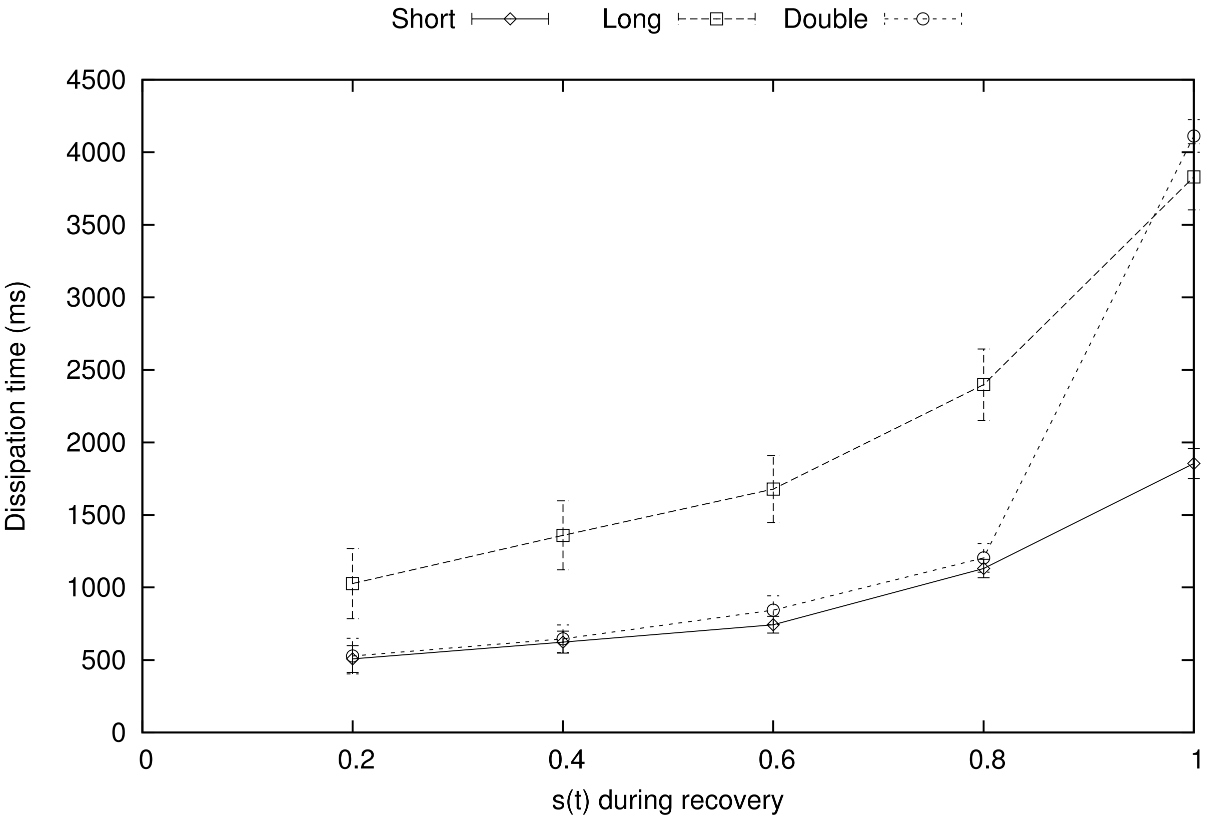


Figure 6.13: Dissipation time for SIMPLE

dissipation time. Such a small improvement is likely outweighed by the larger impact on job releases from selecting a smaller $s(t)$.

In Figure 6.14, we depict the average dissipation time using ADAPTIVE with respect to the aggressiveness factor. As before, we depict error bars for 95% confidence intervals. There is significant variance in the initial choice of $s(t)$ by ADAPTIVE, depending on which level-C jobs complete first after the overload starts, resulting in the larger confidence intervals. This effect is particularly pronounced in the case of DOUBLE. By comparing Figures 6.13 and 6.14, we see that ADAPTIVE significantly reduces the dependency of dissipation time on the length of the overload interval. Furthermore, dissipation times are often significantly smaller under ADAPTIVE than under SIMPLE.

However, in order to fully evaluate ADAPTIVE, we must consider the minimum $s(t)$ value it chooses. Figure 6.15 depicts the average of this choice with respect to the aggressiveness value, in addition to 95% confidence intervals. Here, we see that ADAPTIVE achieves smaller dissipation times than SIMPLE by choosing significantly slower virtual-clock speeds. Thus, jobs are released at a drastically lower frequency

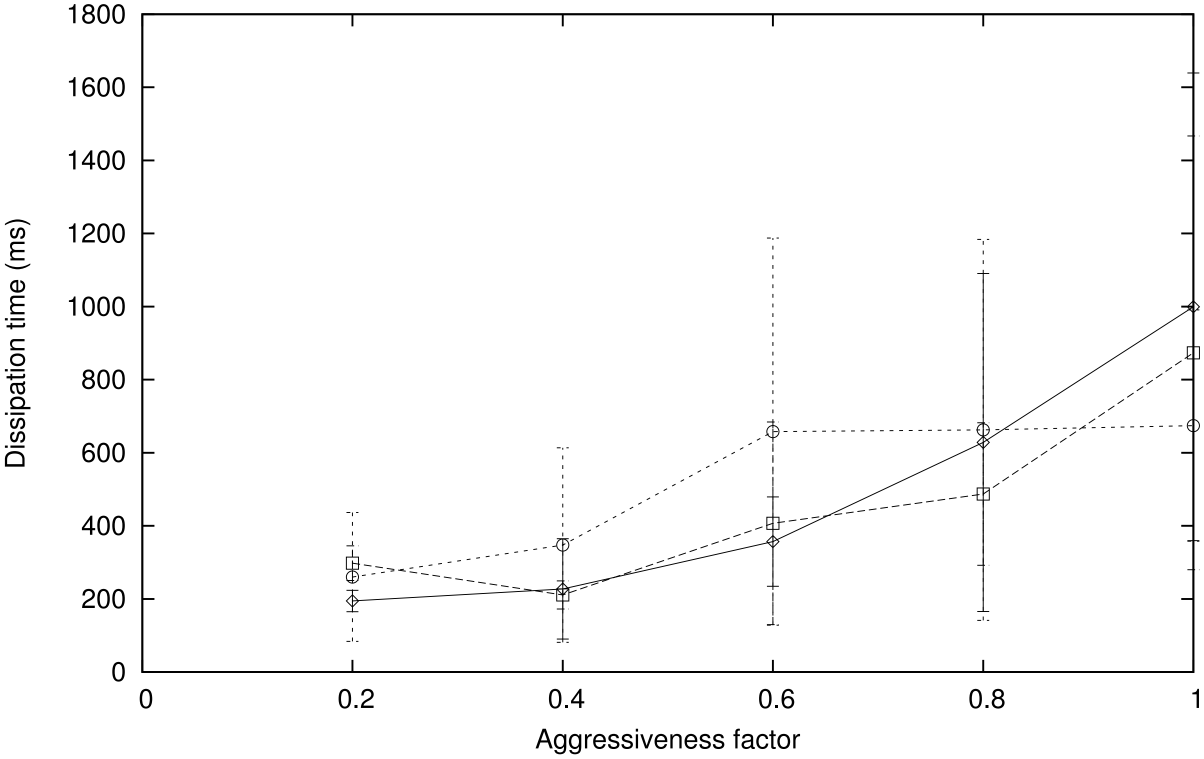


Figure 6.14: Dissipation time for ADAPTIVE

during the recovery period. Therefore, under the highly pessimistic scenarios we considered, SIMPLE is a better choice than ADAPTIVE.

As discussed above, level-C tasks run very little during the overload, so jobs pending at the end of the overload dominate other jobs in producing the largest response times. Because ADAPTIVE usually results in complete recovery from overload before the second overload interval, this causes nearly identical minimum choices of $s(t)$ between SHORT and DOUBLE. Similarly, because the overload interval is twice as long under LONG than under SHORT, the minimum choice of $s(t)$ is about half under LONG compared to SHORT.

In summary, the best choice of monitor under the tested conditions was SIMPLE with $s(t) = 0.6$, although $s(t) = 0.8$ could be a good choice if it is preferable to have a smaller impact on new releases with a longer dissipation time.

We also measured the same overheads considered by Herman et al. (2012) both with and without our virtual time mechanism present, and considering both average and maximum observed overheads. For most overheads considered, there was no significant difference from the virtual time mechanism. However, there was variance in the scheduling overheads, as depicted in Figure 6.16. For average-case overheads, the

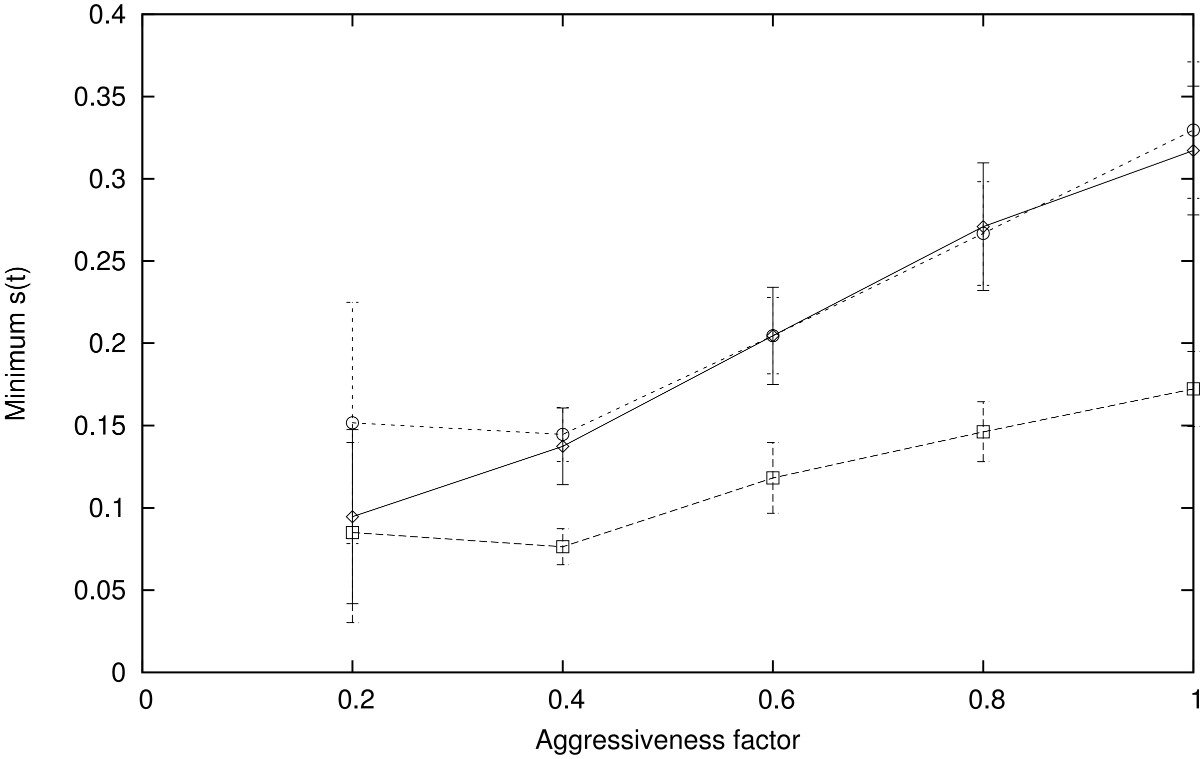


Figure 6.15: Minimum $s(t)$ for ADAPTIVE

introduction of virtual time increased the scheduling time by about 40%, while for worst-case overheads, the introduction of virtual time approximately doubled the scheduling time. Because level C is SRT, the average-case overheads are more relevant, and the cost of adding the virtual time mechanism is small. Furthermore, the userspace monitor program had an effective CPU utilization of approximately 0.1, less than a typical task.

6.6 Conclusion

In this chapter, we addressed the problem of scheduling under MC^2 when a transient overload occurs. We provided methods to analyze response times at level C within MC^2 and proposed a virtual-time mechanism to deal with overload conditions caused by jobs exceeding their level-C PWCETs.

We provided analytical dissipation bounds on the amount of time required to recover from such an overload. We then presented an implementation of our mechanism and provided experiments to demonstrate that it can effectively provide recovery from unexpected overload scenarios. In our experiments, dissipation

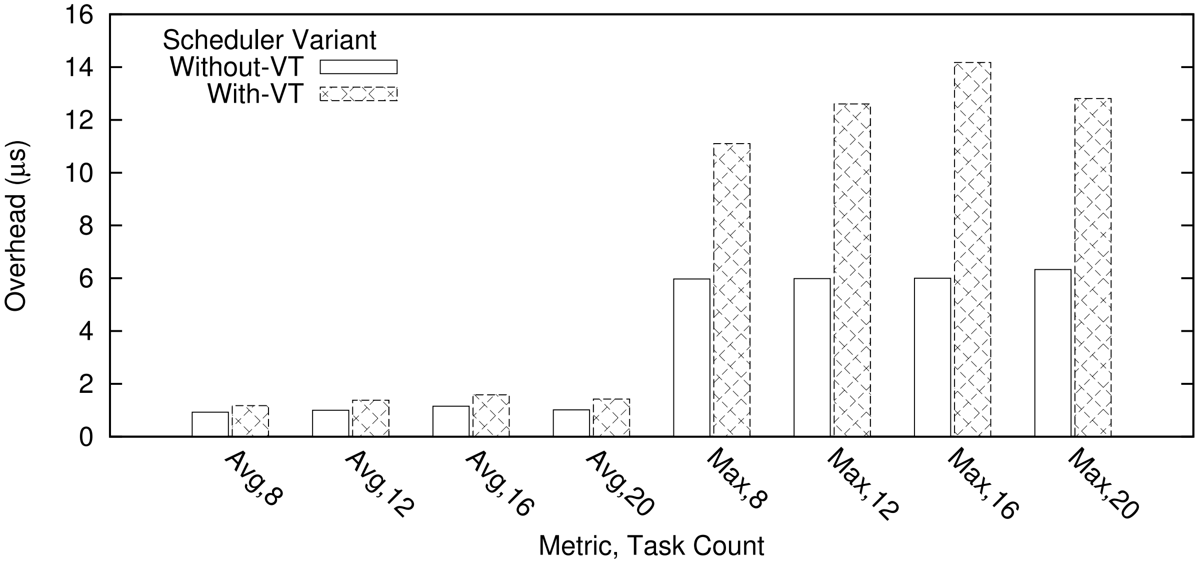


Figure 6.16: Scheduling overhead measurements

times could be brought within twice the length of a pessimistic overload scenario by only moderately affecting the time between job releases, and our scheme created little additional overhead.

CHAPTER 7: CONCLUSION

Prior work has established that G-EDF is a useful scheduler for multiprocessor real-time systems in which bounded lateness is acceptable, and has described how to compute lateness bounds. In this dissertation, we have demonstrated that the broader class of GEL schedulers can be used and modified in order to provide improved performance in the form of smaller lateness bounds. Furthermore, prior work on the MC^2 scheduling framework has not accounted for overload caused by jobs running for longer than their level-C PWCETs. We have provided a mechanism to recover from such overloads, in addition to bounds on the time required for the system to return to normal in the absence of continued overload.

7.1 Summary of Results

In this dissertation, we have supported the thesis that *G-EDF can be modified to support smaller lateness bounds than previous work allows, with more flexibility to specify desired lateness criteria. Furthermore, such modifications do not violate the assumptions required for multiprocessor locking protocols, and the modified scheduler is easier to implement and/or has lower overheads than known HRT-optimal schedulers. In addition, recovery from overloads caused by tasks in MC^2 overrunning their level-C PWCETs can be facilitated by modifying the scheduler to delay job releases dynamically.*

To support this thesis, we have provided improved analysis of GEL schedulers and proposed modifications to G-EDF. We now describe these contributions.

Compliant-Vector Analysis. In Chapter 3, we have proposed CVA as an improved method to compute tardiness bounds under GEL schedulers. Compared to the initial work of Devi and Anderson (2008), we have provided more finely tuned bounds for different tasks within the same system and allow PPs that differ from deadlines, resulting in response-time bounds of the form $Y_i + x_i + C_i$ instead of $D_i + x + C_i$. As a result of this change, response-time bounds may be less than relative deadlines. Under such circumstances, tardiness is simply zero, but lateness is negative. Therefore, we provide better bounds by considering lateness bounds rather than tardiness bounds. Lateness bounds can be computed using an LP.

Global-Fair-Lateness Scheduling. Also in Chapter 3, we have proposed G-FL as a scheduler that provably provides under CVA the best maximum lateness bound under a task system. This property is achieved by ensuring that all tasks in the system have the same lateness bounds. G-FL is simply defined with $Y_i = D_i - \frac{m-1}{m}C_i$, resulting in a simple implementation.

LP Techniques for Assigning PPs. Our final contribution in Chapter 3 has been to propose methods to use an LP to select GEL schedulers to achieve lateness properties that differ from those provided by G-FL. The same LP used to compute lateness bounds for a given CVA scheduler can be extended to include relative PPs as variables and linear functions of lateness as the objective function to be minimized. With such a technique we have shown, for example, that lateness bounds for some tasks can be further reduced from G-FL (although the *maximum* lateness bound in the system cannot be reduced) and that we can minimize the maximum lateness bound *proportional* to a task's relative deadline.

Removing Intra-Task Precedence Constraints. In Chapter 4, we have shown that allowing multiple jobs of the same task to run in parallel can provide several benefits. When only one job of each task can run at a time, no task's utilization can exceed one, or that task's tardiness may grow without bound. However, by allowing multiple jobs of each task to run in parallel, it is only necessary that the entire system not be overutilized. Furthermore, even when allowing multiple jobs of the same task is not necessary for schedulability, doing so can result in smaller lateness bounds. We have provided a method to compute such bounds.

Job Splitting. In Chapter 5, we have considered a technique to split jobs in order to further reduce lateness bounds. With our technique, each job of a task is split into a per-task constant number of subjobs. It is not necessary for the developer to explicitly split the job in source code; rather, the operating system performs the splitting by enforcing subjob budgets. In the absence of overheads and critical sections, such splitting could ensure lateness bounds arbitrarily close to zero. We have provided an experimental study, consisting of both an implementation and a schedulability component, demonstrating the improvements that can be obtained while considering overheads in a real system. We have also provided a method to handle critical sections that is compatible with existing locking protocols, and we have considered the effects of critical sections on achievable lateness bounds.

General Analysis of Level C of MC^2. In Chapter 6, we have provided improved analysis for level C of MC^2. While Leontyev and Anderson (2010) provided restricted supply analysis that can be applied in such a setting, we have provided three improvements. First, we have tightened the analysis in the case that all processors are mostly, but not fully, available to level C. Such a situation is the common case for MC^2. Second, we have provided analysis that is general enough to apply *even when some jobs exceed their level-C PWCETs*. Finally, our analysis explicitly accounts for the virtual time mechanism we describe in the next paragraph.

Virtual Time at Level C of MC^2. If an overload occurs in which jobs exceed their level-C PWCETs, it is possible that the system may never recover or may recover slowly, even in the absence of continued overload. In order to resolve this problem, we have also provided, in Chapter 6, a mechanism to recover from such overloads. Instead of defining job PPs and task minimum separation times with respect to the actual time, we define PPs and minimum separation times with respect to *virtual* time. In the absence of overload, virtual time and actual time are identical, but the system can recover from overload by causing virtual time to elapse more slowly than actual time. This reduces the effective amount of work at level C, assisting the system in recovering from the overload.

Dissipation Time and Bounds at Level C of MC^2. Additionally in Chapter 6, we have provided a condition that ensures that the system has "returned to normal," and that virtual time can safely proceed at the same speed as actual time after this condition is satisfied. We have provided two forms of analysis of the dissipation time, or the time required for the system to reach this condition once an overload has completed. First, we have provided an experimental study of the dissipation time on a real system. Second, we have provided theoretical dissipation bounds, or bounds on dissipation time.

7.2 Other Related Work

In this section, we briefly describe our other contributions to the field of real-time systems that have not been included in this dissertation.

Original MC^2 Proposal. In (Mollison et al., 2010), we proposed MC^2 in its original form with five criticality levels. In that form, an additional SRT level D is also present, is scheduled under G-EDF, and is prioritized below level C. Otherwise, that form of MC^2 is identical to that reviewed in Chapter 1. In this dissertation, we have focused on the newer MC^2 model and have considered only level C.

Scheduling in Google Earth. In (Erickson et al., 2012), we considered scheduling within the virtual globe software Google Earth. We considered scheduling techniques to reduce the problem of *stutter*, where the display cannot update to a new image at the correct time, preventing smooth animation.

This work involved predicting how long certain jobs within Google Earth would run. However, because Google Earth runs on a wide variety of software and hardware platforms, it is not possible to determine job execution times *a priori*. Thus, we proposed a system to estimate job execution times online. We also implemented our system in Google Earth, and it has been included in the official Google Earth release since version 6.2.

Scheduling Automotive-Inspired Dataflows. In (Elliott et al., 2014), we considered the multicore scheduling of DAG-based systems with producer/consumer dependencies. In such systems, dependencies are specified using a DAG, and each job must wait to start until all its corresponding jobs in its predecessor tasks have finished. We extended prior work (Liu and Anderson, 2010, 2011) by adding additional constraints to reduce pessimism, using C-FL (the clustered variant of G-FL) in place of C-EDF (the clustered variant of G-EDF), using the job splitting technique proposed in Chapter 5, and intelligently assigning tasks to clusters of processors to reduce cache-related overhead. We performed an implementation study to analyze response times.

Optimal Soft Real-Time Semi-Partitioned Scheduling. In (Anderson et al., 2014), we proposed the *EDF-os* algorithm for SRT scheduling. EDF-os is derived from EDF-fm (Anderson et al., 2008), which is a *semi-partitioned* scheduler that does not allow most tasks to migrate between processors, but allows certain tasks to migrate at job boundaries. EDF-fm required restrictions on per-task utilizations. In EDF-os, we provided several modifications that eliminated the need for such restrictions. We also provided both lateness bounds for EDF-os and an implementation study comparing EDF-os to several other algorithms. EDF-os can often provide smaller lateness bounds than G-FL, at the expense of not allowing the task system to be modified at runtime.

7.3 Future Work

We now discuss some future work that could improve the results presented in this dissertation.

Tighter Lateness Bounds. While the lateness bounds provided in Chapter 3 are tighter than those provided in previous work, we still do not believe that they are tight. There are two pessimistic assumptions that may be able to be lifted. For one, we assume that jobs do not begin execution before their PPs, in order to avoid some complex edge cases in our analysis. Additionally, we assume that each task that carries work into the beginning of a busy interval has as much lateness as possible. It may be possible to prove that this cannot be the case.

Selection of GEL Schedulers in the Absence of Intra-Task Precedence Constraints. As discussed in Chapter 4, the LP techniques used in Chapter 3 cannot immediately be applied in the absence of intra-task precedence constraints. Therefore, although our analysis is general enough to apply to arbitrary GEL schedulers, we do not specify either a scheduler like G-FL that provably has particular desired properties, or a method to choose a GEL scheduler to minimize particular lateness criteria. The same insights used to compute the response-time bounds in Chapter 4 may also be useful to achieve these goals using a method other than an LP.

Splitting Jobs with Other Locking Analysis. In Chapter 5, we consider only one mutex queue spin-based locking protocol, where a CPU waiting to acquire a lock consumes CPU until the lock is available. However, the technique we use could also be applied to other forms of locking protocols. We leave such extensions as future work.

More Flexible Mechanisms for Overload Management at Level C in MC^2. The virtual time mechanism proposed in Chapter 6 equally penalizes all level-C tasks when recovering from overload. This technique makes sense in a mixed-criticality setting, because tasks of greater importance than others could be moved to level A or level B. However, a finer-grained notion of importance may be preferable: some tasks may not require the full rigor of level-A or level-B analysis, but may be more important in an overload than other level-C tasks. Alternative mechanisms that allow different tasks to be treated differently would be useful in such situations.

APPENDIX A: NOTATION USED IN CHAPTER 3

b	LP auxilliary variable used while computing $G(\vec{x},\vec{Y})$
C_i	Worst-case execution time for τ_i
$C_{\max}$	Maximum C_i for τ
D_i	Deadline for τ_i
G	LP variable corresponding to $G(\vec{x},\vec{Y})$
$G(\vec{x},\vec{Y})$	Extra demand from certain analyzed tasks (see (3.12))
H	Set of jobs with priority at least that of $\tau_{i,k}$
I_i	Proportional lateness bound for τ_i
$I_{\max}$	Maximum proportional lateness bound for τ
L_i	Lateness bound for τ_i
$L_{\max}$	Maximum lateness bound of G-FL for τ
m	Processor count
R_i	Response time bound for τ_i
s	LP variable corresponding to $G(\vec{x},\vec{Y})+S(\vec{Y})$
$S_i(Y_i)$	Extra demand term for τ_i due to $D_i < T_i$ (see (3.8))
S_{sum}	LP variable corresponding to $S(\vec{Y})$
$S(\vec{Y})$	Sum of $S_i(Y_i)$ terms over task system (see (3.9))
t_b	End of last idle interval before $y_{i,k}$
t_d	Beginning of last idle interval before $y_{i,k}$
T_i	Minimum separation time for τ_i
Y_i	Relative PP for τ_i
U_i	Utilization for τ_i
U^+	Ceiling of utilization sum for τ
$W_j(t)$	Total remaining work at time t for jobs of τ_j in H
$W(t)$	Total remaining work at time t for jobs in H
x_i	Component of response time bound $Y_i+x_i+C_i$ for τ_i (see (3.26))

$y_{i,k}$	PP of $\tau_{i,k}$
z_i	LP auxilliary variable used while computing $G(\vec{x},\vec{Y})$
δ	Constant used to describe already completed work or early completion
τ	Task system
τ_i	Task i
$\tau_{i,k}$	Arbitrary job under analysis
$\tau_{j,\ell}$	Other arbitrary job

APPENDIX B: PROOFS OF LEMMAS IN CHAPTER 3

Lemma 3.1. *If $\Delta \geq Y_i$, then*

$$DBF(\tau_i, Y_i, \Delta) \leq U_i\Delta + C_i \cdot \left(1 - \frac{Y_i}{T_i}\right).$$

Proof. As first demonstrated in Baruah et al. (1990), for arbitrary-deadline G-EDF,

$$\text{DBF}(\tau_i, \Delta) = C_i \times \max\left\{0, \left\lfloor \frac{\Delta - D_i}{T_i} \right\rfloor + 1\right\}.$$

Therefore, for arbitrary GEL schedulers, we instead define

$$\text{DBF}(\tau_i, Y_i, \Delta) = C_i \times \max\left\{0, \left\lfloor \frac{\Delta - Y_i}{T_i} \right\rfloor + 1\right\}. \tag{B.1}$$

If $\Delta \geq Y_i$, then

$$\begin{aligned}
U_i\Delta + C_i\left(1 - \frac{Y_i}{T_i}\right) \geq\ & \{\text{By the definition of the floor function}\} \\
& U_i\left(T_i\left\lfloor \frac{\Delta - Y_i}{T_i}\right\rfloor + Y_i\right) + C_i\left(1 - \frac{Y_i}{T_i}\right) \\
=\ & \{\text{Rearranging, and because } U_i = \tfrac{C_i}{T_i}\} \\
& C_i\left\lfloor \frac{\Delta - Y_i}{T_i}\right\rfloor + \frac{C_iY_i}{T_i} + C_i - \frac{C_iY_i}{T_i} \\
=\ & \{\text{Cancelling and rearranging}\} \\
& C_i\left(\left\lfloor \frac{\Delta - Y_i}{T_i}\right\rfloor + 1\right) \\
=\ & \{\text{By (B.1)}\} \\
& \text{DBF}(\tau_i, Y_i, \Delta)
\end{aligned}$$

□

Lemma 3.2. $\forall \Delta \geq 0, DBF(\tau_i, Y_i, \Delta) \leq U_i\Delta + S_i(Y_i)$.

Proof. We consider two cases.

We first consider $\Delta \in [0, Y_i)$. In this case, $\Delta - Y_i < 0$, so $\left\lfloor \frac{\Delta - Y_i}{T_i} \right\rfloor + 1 \leq 0$. Therefore, by the definition of $S_i(Y_i)$ in (3.8) and (B.1), $\text{DBF}(\tau_i, Y_i, \Delta) = 0 \leq U_i\Delta + S_i(Y_i)$.

We then consider $\Delta \geq Y_i$. By the definition of $S_i(Y_i)$ in (3.8),

$$U_i\Delta + S_i(Y_i) \geq U_i\Delta + C_i\left(1 - \frac{Y_i}{T_i}\right).$$

Therefore, by Lemma 3.1,

$$\mathrm{DBF}(\tau_i, Y_i, \Delta) \leq U_i\Delta + S_i(Y_i).$$ □

Lemma 3.5. *If $\vec{x}$ is compliant and each job $\tau_{j,\ell}$ with higher priority than $\tau_{i,k}$ finishes with response time no greater than $Y_j + x_j + C_j$, then $\tau_{i,k}$ finishes with response time no greater than $Y_i + x_i + C_i$.*

Proof. Let δ denote the sum of the amount of execution of $\tau_{i,k}$ before $y_{i,k}$ and the amount of time by which it finishes early, so that $C_i - \delta$ units remain at time $y_{i,k}$. By Lemma 3.4, the remaining work at time $y_{i,k}$ for all jobs in H (including $\tau_{i,k}$ itself) is at most $G(\vec{x}, \vec{Y}) + S(\vec{Y})$. Therefore, there can be at most $G(\vec{x}, \vec{Y}) + S(\vec{Y}) - (C_i - \delta)$ units of competing work from other tasks. (This expression actually bounds all competing work, but is sufficient as an upper bound for work from other tasks.) Because each $Y_i \geq 0$, no job releases after $y_{i,k}$ affect the scheduling of $\tau_{i,k}$. As depicted in Figure B.1, a CPU becomes available for τ_i at the earliest time that any CPU completes its competing work. Therefore, in the worst case, all CPUs execute competing work with maximum parallelism, and a CPU becomes available for τ_i at time

$$t_c = y_{i,k} + \frac{G(\vec{X}, \vec{Y}) + S(\vec{Y}) - C_i + \delta}{m}. \tag{B.2}$$

We consider two cases, depicted in Figure B.2:

Case 1 (Figure B.2(a)). If some predecessor of $\tau_{i,k}$ is running at time t_c, we note that the immediate predecessor of $\tau_{i,k}$ is $\tau_{i,k-1}$. Because all competing work from other tasks has completed, $\tau_{i,k}$ can begin execution immediately upon completion of $\tau_{i,k-1}$. Therefore, because $\tau_{i,k-1}$ has a response time no greater than the assumed response bound in (3.10), and $\tau_{i,k}$ must have been released at least T_i units after $\tau_{i,k-1}$, $\tau_{i,k-1}$ must complete no later than $Y_i + x_i + C_i - T_i$ units after the release of $\tau_{i,k}$. Because we assumed $C_i \leq T_i$, $\tau_{i,k}$ must complete within $Y_i + x_i + C_i - T_i + C_i \leq Y_i + x_i + C_i$ units of its release, and the lemma is proven.

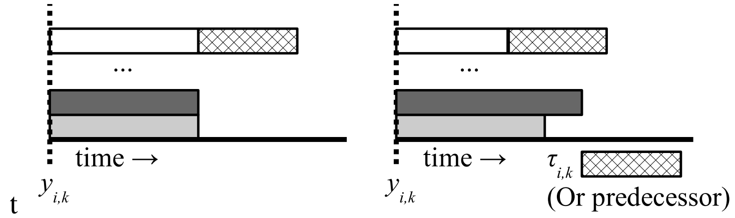


Figure B.1: Possible competing work schedules after $y_{i,k}$.

Case 2 (Figure B.2(b)). If no predecessor of $\tau_{i,k}$ is running at time t_c, $\tau_{i,k}$ is eligible for execution at time t_c unless it has already completed. At most $C_i - \delta$ units of execution remain, so it completes by time

$$
\begin{aligned}
y_{i,k} + \frac{G(\vec{x},\vec{Y}) + S(\vec{Y}) - C_i + \delta}{m} + C_i - \delta \leq{}& \{\text{Because } \delta > \tfrac{\delta}{m}\} \\
& y_{i,k} + \frac{G(\vec{x},\vec{Y}) + S(\vec{Y}) - C_i}{m} + C_i \\
\leq{}& \{\text{By Definition 3.1}\} \\
& y_{i,k} + x_i + C_i,
\end{aligned}
$$

and the lemma is proven. □

Lemma 3.6. *$\vec{x}$ is a minimum near-compliant vector if and only if*

$$\forall i, x_i = \frac{G(\vec{x},\vec{Y}) + S(\vec{Y}) - C_i}{m}. \tag{3.26}$$

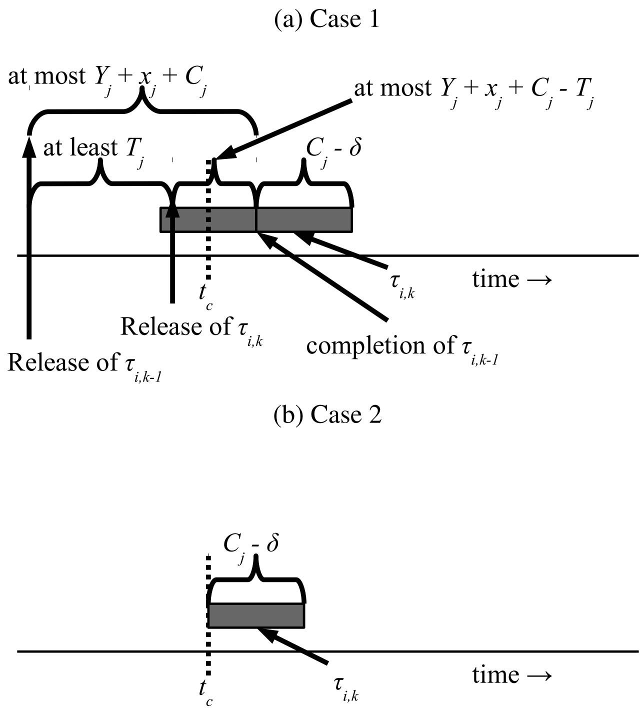
Proof. We consider two cases, depending on whether (3.26) holds.

Case 1: (3.26) Holds. If (3.26) holds, then by Definition 3.1, $\vec{x}$ is near-compliant. Let $\delta > 0$ and $\vec{x'}$ be such that, for some j,

$$x'_j = x_j - \delta, \tag{B.3}$$

and for $i \neq j$, $x'_i = x_i$. Then, by the definition of $G(\vec{x},\vec{Y})$ in (3.12),

$$G(\vec{x'},\vec{Y}) \geq G(\vec{x},\vec{Y}) - U_j\delta. \tag{B.4}$$


(a) Case 1

(b) Case 2

Figure B.2: Cases considered in Lemma 3.5

Therefore,

$$
\begin{aligned}
\frac{G(\vec{x'},\vec{Y})+S(\vec{Y})-C_j}{m} &\geq \{\text{By (B.4)}\} \\
&\quad \frac{G(\vec{x},\vec{Y})-U_j\delta+S(\vec{Y})-C_j}{m} \\
&= \{\text{Rearranging}\} \\
&\quad \frac{G(\vec{x},\vec{Y})+S(\vec{Y})-C_j}{m}-\frac{U_j\delta}{m} \\
&= \{\text{By (3.26) with } i=j\} \\
&\quad x_j-\frac{U_j\delta}{m} \\
&> \{\text{Because } U_j>0,\ m>1, \text{ and } \delta>0, \text{ and by (B.3)}\} \\
&\quad x'_j
\end{aligned}
$$

Therefore, by Definition 3.1, $\vec{x'}$ is not near-compliant. Since j and δ were arbitrary, by Definition 3.2, $\vec{x}$ is a minimum near-compliant vector.

Case 2: (3.26) Does not Hold. Suppose (3.26) does not hold. If, for some j, $x_j < \frac{G(\vec{x},\vec{Y})+S(\vec{Y})-C_j}{m}$, then by Definition 3.1, $\vec{x}$ is not near-compliant. Thus, if $\vec{x}$ is near-compliant, then for some j,

$$x_j > \frac{G(\vec{x},\vec{Y})+S(\vec{Y})-C_j}{m}. \tag{B.5}$$

If there is more than one such j, we choose one arbitrarily. We define $\vec{x'}$ such that

$$x'_j = \frac{G(\vec{x},\vec{Y})+S(\vec{Y})-C_j}{m}, \tag{B.6}$$

and for $i \neq j$,

$$x'_i = x_i. \tag{B.7}$$

By the definition of $G(\vec{x},\vec{Y})$ in (3.12) and (B.5)–(B.7),

$$G(\vec{x'},\vec{Y}) \leq G(\vec{x},\vec{Y}). \tag{B.8}$$

Therefore,

$$\begin{aligned} x'_j &= \{\text{By (B.6)}\} \\ &\quad \frac{G(\vec{x},\vec{Y})+S(\vec{Y})-C_i}{m} \\ &\geq \{\text{By (B.8)}\} \\ &\quad \frac{G(\vec{x'},\vec{Y})+S(\vec{Y})-C_i}{m}, \end{aligned}$$

and, for $i \neq j$,

$$\begin{aligned} x'_i &= \{\text{By (B.7)}\} \\ &\quad x_i \\ &\geq \{\text{By Definition 3.1}\} \\ &\quad \frac{G(\vec{x},\vec{Y})+S(\vec{Y})-C_i}{m} \\ &\geq \{\text{By (B.8)}\} \end{aligned}$$

$$\frac{G(\vec{x'},\vec{Y})+S(\vec{Y})-C_i}{m}.$$

Thus, $\vec{x'}$ is near-compliant, so by Definition 3.2, $\vec{x}$ is not a *minimum* near-compliant vector. □

Lemma 3.7. *If the minimum near-compliant vector for the task system exists, it is unique.*

Proof. We use proof by contradiction. Assume that two distinct minimum near-compliant vectors exist for the task system, $\vec{x}$ with

$$\forall i, x_i = \frac{s-C_i}{m}, \tag{B.9}$$

and $\vec{x}'$ with

$$\forall i, x_i' = \frac{s'-C_i}{m}. \tag{B.10}$$

Without loss of generality, assume $s' < s$.

We show

$$\begin{aligned}
s = &\{\text{By Corollary 3.1}\}\\
&G(\vec{x},\vec{Y})+S(\vec{Y})\\
&=\{\text{By the definition of } G(\vec{x},\vec{Y}) \text{ in (3.12)}\}\\
&\sum_{U^+-1 \text{ largest}} (x_iU_i+C_i-S_i(Y_i))+S(\vec{Y})\\
&=\{\text{By (B.9)}\}\\
&\sum_{U^+-1 \text{ largest}} \left(\left(\frac{s-C_i}{m}\right)U_i+C_i-S_i(Y_i)\right)+S(\vec{Y})\\
&=\{\text{Rearranging}\}\\
&\sum_{U^+-1 \text{ largest}} \left(\left(\frac{s'-C_i}{m}\right)U_i+\left(\frac{s-s'}{m}\right)U_i+C_i-S_i(Y_i)\right)+S(\vec{Y})\\
&<\{\text{Because } U^+\le m \text{ and } U_i\le 1 \text{ holds for each } i\}\\
&s-s'+\sum_{U^+-1 \text{ largest}} \left(\left(\frac{s'-C_i}{m}\right)U_i+C_i-S_i(Y_i)\right)+S(\vec{Y})\\
&=\{\text{By (B.10)}\}\\
&s-s'+\sum_{U^+-1 \text{ largest}} (x_i'U_i+C_i-S_i(Y_i))+S(\vec{Y})
\end{aligned}$$

$$
\begin{aligned}
&= \{\text{By the definition of } G(\vec{x},\vec{Y}) \text{ in (3.12))}\} \\
&\quad s - s' + G(\vec{x}',\vec{Y}) + S(\vec{Y}) \\
&= \{\text{By Corollary 3.1}\} \\
&\quad s - s' + s' \\
&= \{\text{Cancelling}\} \\
&\quad s. \qquad \text{(B.11)}
\end{aligned}
$$

(B.11) is a contradiction, so the lemma is proven. □

Lemma 3.9. *For any assignment of variables satisfying Constraint Sets 3.1–3.4,*

$$G + S_{\text{sum}} \geq G(\vec{x},\vec{Y}) + S(\vec{Y}).$$

Proof.

$$
\begin{aligned}
G + S_{\text{sum}} &\geq \{\text{By (3.31) and Constraint Set 3.4}\} \\
&\quad \sum_{U^+-1 \text{ largest}} (x_i U_i + C_i - S_i) + \sum_{\tau_i \in \tau} S_i \\
&\geq \{\text{By (3.30); observe that each } S_i \text{ appearing in the first summation also appears in the second}\} \\
&\quad \sum_{U^+-1 \text{ largest}} (x_i U_i + C_i - S_i(Y_i)) + \sum_{\tau_i \in \tau} S_i(Y_i) \\
&= \{\text{By the definition of } S(\vec{Y}) \text{ in (3.9), and the definition of } G(\vec{x},\vec{Y}) \text{ in (3.12)}\} \\
&\quad G(\vec{x},\vec{Y}) + S(\vec{Y}).
\end{aligned}
$$

□

Lemma 3.10. *For any assignment of variables satisfying Constraint Sets 3.1–3.5, $\vec{x}$ is a near-compliant vector.*

Proof. Consider an arbitrary assignment of variables satisfying Constraint Sets 3.1–3.5. For arbitrary i,

$$
\begin{aligned}
x_i &= \{\text{By Constraint Set 3.1}\} \\
&\quad \frac{s - C_i}{m}
\end{aligned}
$$

$$\begin{aligned}
&\geq \{\text{By Constraint Set 3.5}\}\\
&\quad \frac{G+S_{\text{sum}}-C_i}{m}\\
&\geq \{\text{By Lemma 3.9}\}\\
&\quad \frac{G(\vec{x},\vec{Y})+S(\vec{Y})-C_i}{m}. \qquad \text{(B.12)}
\end{aligned}$$

By (B.12) and Definition 3.1, $\vec{x}$ is a near-compliant vector. □

Lemma 3.11. *If $s<0$, then Constraint Sets 3.1–3.5 are infeasible.*

Proof. We use proof by contradiction. Assume Constraint Sets 3.1–3.5 are satisfied by some assignment of variables with

$$s<0. \qquad \text{(B.13)}$$

Denote as θ the set of U^+-1 tasks that contribute to $G(\vec{x},\vec{Y})$, so that by the definition of $G(\vec{x},\vec{Y})$ in (3.12),

$$G(\vec{x},\vec{Y}) = \sum_{\tau_i\in\theta}(x_iU_i+C_i-S_i(Y_i)). \qquad \text{(B.14)}$$

We have

$$\begin{aligned}
s &\geq \{\text{By Constraint Set 3.5}\}\\
&\quad G+S_{\text{sum}}\\
&\geq \{\text{By Lemma 3.9}\}\\
&\quad G(\vec{x},\vec{Y})+S(\vec{Y})\\
&= \{\text{By the definition of } S(\vec{Y}) \text{ in (3.9) and (B.14)}\}\\
&\quad \sum_{\tau_i\in\theta}(x_iU_i+C_i-S_i(Y_i))+\sum_{\tau_i\in\tau}S_i(Y_i)\\
&= \{\text{By Constraint Set 3.1}\}\\
&\quad \sum_{\tau_i\in\theta}\left(\left(\frac{s-C_i}{m}\right)U_i+C_i-S_i(Y_i)\right)+\sum_{\tau_i\in\tau}S_i(Y_i)\\
&= \{\text{Rewriting}\}\\
&\quad \sum_{\tau_i\in\theta}\left(\frac{s}{m}\cdot U_i\right)+\sum_{\tau_i\in\theta}\left(\left(\frac{-C_i}{m}\right)U_i+C_i-S_i(Y_i)\right)+\sum_{\tau_i\in\tau}S_i(Y_i)
\end{aligned}$$

$>$ {By (B.13), because θ has $U^+ - 1 < m$ tasks, and $0 < U_i \leq 1$ holds for each U_i}

$$s + \sum_{\tau_i \in \theta} \left(\left(\frac{-C_i}{m} \right) U_i + C_i - S_i(Y_i) \right) + \sum_{\tau_i \in \tau} S_i(Y_i)$$

$=$ {Rearranging}

$$s + \sum_{\tau_i \in \theta} \left(\left(\frac{C_i(m - U_i)}{m} \right) - S_i(Y_i) \right) + \sum_{\tau_i \in \tau} S_i(Y_i)$$

$\geq$ {Because each $S_i(Y_i) \geq 0$ by the definition of $S_i(Y_i)$ in (3.8); observe that each $S_i(Y_i)$ in the first summation also appears in the second}

$$s + \sum_{\tau_i \in \theta} \frac{C_i(m - U_i)}{m}$$

$>$ {Because $U_i \leq 1 < m$}

$$s.$$

This is a contradiction. □

Lemma 3.12. *Constraint Sets 3.1–3.5 are feasible.*

Proof. We present an assignment of variables that satisfies Constraint Sets 3.1–3.5. Let $C_{\max}$ denote the largest value of C_i in the system. Let

$$s = m^2 C_{\max} + mS(\vec{Y}), \tag{B.15}$$

$$\forall i, x_i = \frac{s - C_i}{m}, \tag{B.16}$$

$$\forall i, S_i = S_i(Y_i), \tag{B.17}$$

$$G = G(\vec{x}, \vec{Y}), \tag{B.18}$$

$$b = (U^+ - 1)^{th} \text{ largest value of } x_i U_i + C_i - S_i, \tag{B.19}$$

$$\forall i, z_i = \max\{0,\ x_i U_i + C_i - S_i - b\}, \tag{B.20}$$

$$S_{\text{sum}} = S(\vec{Y}). \tag{B.21}$$

Constraint Set 3.1 holds by (B.16).

Constraint Set 3.2 holds by (B.17).

Constraint Set 3.3 holds by (B.18)–(B.20).

Constraint Set 3.4 holds by (B.17) and (B.21).

To see that Constraint Set 3.5 holds, note that

$$
\begin{aligned}
G + S_{\text{sum}} &= \{\text{By (B.18) and (B.21)}\}\\
&\quad G(\vec{x},\vec{Y}) + S(\vec{Y})\\
&= \{\text{By the definition of } G(\vec{x},\vec{Y}) \text{ in (3.12)}\}\\
&\quad \sum_{U^+-1 \text{ largest}} (x_i U_i + C_i - S_i(Y_i)) + S(\vec{Y})\\
&= \{\text{By (B.15)–(B.16)}\}\\
&\quad \sum_{U^+-1 \text{ largest}} \left(\left(\frac{m^2 C_{\max} + mS(\vec{Y}) - C_i}{m}\right) U_i + C_i - S_i(Y_i)\right) + S(\vec{Y})\\
&< \{\text{Because } U^+ \leq m,\ \forall i, C_i \leq C_{\max}, \text{ and each } U_i \leq 1\}\\
&\quad \sum_{m-1 \text{ largest}} (mC_{\max} + S(\vec{Y}) + C_{\max}) + S(\vec{Y})\\
&= \{\text{Rearranging}\}\\
&\quad (m^2-1)C_{\max} + mS(\vec{Y})\\
&< \{\text{By (B.15), because } C_{\max} > 0\}\\
&\quad s.
\end{aligned}
$$

□

Lemma 3.14. *If $U^+ > 1$ and $s < C_{\max}$, then Constraint Sets 3.1–3.5 are infeasible.*

Proof. We use proof by contradiction. Assume $U^+ > 1$ and Constraint Sets 3.1–3.5 are satisfied by some assignment of variables with

$$s < C_{\max}. \tag{B.22}$$

We denote as ϕ the set of $U^+ - 1$ tasks with the largest values of C_i. Because ϕ has $U^+ - 1$ tasks, by the definition of $G(\vec{x},\vec{Y})$ in (3.12),

$$G(\vec{x},\vec{Y}) \geq \sum_{\tau_i \in \phi} (x_i U_i + C_i - S_i(Y_i)). \tag{B.23}$$

$$s \geq \{\text{By Constraint Set 3.5}\}$$

$$G + S_{\text{sum}}$$

$\geq$ {By Lemma 3.9}

$$G(\vec{x}, \vec{Y}) + S(\vec{Y})$$

$\geq$ {By the definition of $S(\vec{Y})$ in (3.9) and (B.23)}

$$\sum_{\tau_i \in \phi} (x_i U_i + C_i - S_i(Y_i)) + \sum_{\tau_i \in \tau} S_i(Y_i)$$

$=$ {By Constraint Set 3.1}

$$\sum_{\tau_i \in \phi} \left(\left(\frac{s - C_i}{m} \right) U_i + C_i - S_i(Y_i) \right) + \sum_{\tau_i \in \tau} S_i(Y_i)$$

$=$ {Rewriting}

$$\sum_{\tau_i \in \phi} \left(\left(\frac{s - C_{\max} + C_{\max} - C_i}{m} \right) U_i + C_i - S_i(Y_i) \right) + \sum_{\tau_i \in \tau} S_i(Y_i)$$

$=$ {Rewriting}

$$\sum_{\tau_i \in \phi} \left(\frac{s - C_{\max}}{m} \cdot U_i \right) + \sum_{\tau_i \in \phi} \left(\left(\frac{C_{\max} - C_i}{m} \right) U_i + C_i - S_i(Y_i) \right) + \sum_{\tau_i \in \tau} S_i(Y_i)$$

$>$ {By (B.22), because ϕ has $U^+ - 1 < m$ tasks, and $0 < U_i \leq 1$ holds for each U_i}

$$s - C_{\max} + \sum_{\tau_i \in \phi} \left(\left(\frac{C_{\max} - C_i}{m} \right) U_i + C_i - S_i(Y_i) \right) + \sum_{\tau_i \in \tau} S_i(Y_i)$$

$\geq$ {Because $C_{\max} \geq C_i$ for all i}

$$s - C_{\max} + \sum_{\tau_i \in \phi} (C_i - S_i(Y_i)) + \sum_{\tau_i \in \tau} S_i(Y_i)$$

$\geq$ {Because each $S_i(Y_i) \geq 0$ by the definition of $S_i(Y_i)$ in (3.8); observe that each $S_i(Y_i)$ in the first summation also appears in the second.}

$$s - C_{\max} + \sum_{\tau_i \in \phi} C_i$$

$\geq$ {By the definition of ϕ, which contains the $U^+ - 1 > 0$ tasks with the largest values of C_i}

$$s.$$

This is a contradiction. □

APPENDIX C: NOTATION USED IN CHAPTER 6

$A_{i,k}(v)$	Function used to account for completion time in the Few Tasks Case (see (6.13))
$A_i^{rn}(v)$	Upper bound on $A_{i,k}(v)$ for $[t_r, \infty)$ (see (6.29))
$b_{i,k}$	Earliest time such that $v(b_{i,k}) = v(r_{i,k}) + T_i$ (see Definition 6.5)
C_i	Worst-case execution time of τ_i in the absence of overload (see Property 6.11)
$D_i^C(t_0,t_1)$	Upper bound on $D_i^e(t_0,t_1)$ when $t_0 > t_r$ (see Definition 6.18)
$D_i^e(t_0,t_1)$	Total execution cost from jobs $\tau_{i,k}$ with $t_0 \le r_{i,k} \le y_{i,k} \le t_1$ (see Definition 6.6)
$e_{i,k}$	Actual execution of $\tau_{i,k}$ (see Definition 6.1)
$e_{i,k}^c(t)$	Work completed by $\tau_{i,k}$ before actual time t (see Definition 6.2)
$e_{i,k}^p$	Work after $y_{i,k}$ for jobs of τ_i prior to $\tau_{i,k}$ (see Definition 6.9)
$e_{i,k}^r(t)$	Work remaining for $\tau_{i,k}$ after actual time t (see Definition 6.3)
F	Constant that guarantees idleness in a certain interval (see (6.152))
L	Arbitrary integer parameter with $0 \le L < m$
L_i	Selection of L for τ_i in Section 6.3 (see Definition 6.14)
m	Number of CPUs in the system
n	Number of level-C tasks in the system
$O_{i,k}$	Term to account for supply restriction overload (see (6.20))
$O_{i,k}^o$	Term used to bound the contribution to $O_{i,k}$ from $[0,t_r)$ (see (6.44))
O^{rn}	Term used to bound the contribution to $O_{i,k}$ from $[t_r, \infty)$ (see (6.39))
$o_p(t_0,t_1)$	Supply restriction overload over $[t_0,t_1)$ (see (6.8))
P	Set of all processors
P_p	Processor p
q	Base of the exponential function in $\Delta^e(t)$ (see (6.49))
$r_{i,k}$	Release time of $\tau_{i,k}$
$R_{i,k}$	Term to account for work completed in $[t_{i,k}^b, y_{i,k})$ (see (6.21))
$R_{i,k}^o$	Term to account for the contribution of $[0,t_r)$ to $R_{i,k}$ (see (6.43))
S_i	Term used to account for $Y_i < T_i$ (see (6.38))

$s(t)$	Speed of virtual time at actual time t
s_r	Speed of virtual time during recovery interval (see Property 6.6)
T_i	Minimum separation time between jobs of τ_i in virtual time (see (6.5))
t_a	Actual time under immediate analysis
$t_{i,k}^c$	Completion time of $\tau_{i,k}$
t_d	Time at which some processor is idle and $x_j\left(y_{j,\ell}\right) = x_j^s(1)$ is x-sufficient for all pending $\tau_{j,\ell}$ (see Definition 6.20)
t_e	Time at which $\Delta(t)$ switches from linear to exponential (see (6.48))
t_n	Time at which the virtual clock can return to normal speed, because all jobs pending at t_d are complete (see Definition 6.21)
t_n^{pre}	Time after which $x_i\left(y_{i,k}\right) = x_i^s(1)$ is x-sufficient for all pending jobs (see (6.140))
t_r	Time at start of recovery interval $[t_r, t_n)$ (see Definition 6.13)
t_s	Time at which virtual clock actually slows to stable value
t_δ	Time at which $\Delta(t_\delta) = \delta$ (see (6.134))
$\widehat{u_p}$	Nominal utilization (of availability) of P_p
u_{tot}	Sum of nominal utilizations over all processors (see (6.10))
U_i^r	Utilization of τ_i in $[t_r, t_n)$ (see (6.28))
U_i^v	Utilization of τ_i in virtual time (see (6.27))
$v(t)$	Virtual time corresponding to actual time t (see (6.4))
$W_{i,k}$	Term to account for work (see (6.17))
$W_{i,k}^o$	Term to account for the contribution of $[0, t_r)$ to $W_{i,k}$ (see (6.42))
$W_{i,k}^r$	Term to account for part of $W_{i,k}$ when $t_{i,k}^b \in [t_r, t_n)$ (see (6.112))
$x_i(t)$	Upper bound described in Definition 6.8
$\dot{x}_i(t)$	Upper bound described in Definition 6.16
$x_i^p(t)$	Lower bound described in Definition 6.19
$x_i^{p_r}(t)$	Particular choice of $x_i^p(t)$ defined in (6.73)
$x_i^s(s_I)$	Asymptotic bound on $x_i(t)$ in the absence of overload when $s(t) = s_I$ (see (6.37))
$x_{i,k}^f$	x-sufficient choice of $x_i\left(y_{i,k}\right)$ in the Few Tasks Case (see (6.15))

$x^m_{i,k}$	x-sufficient choice of x_i $(y_{i,k})$ in the Many Tasks Case (see (6.24))
$y_{i,k}$	PP of $\tau_{i,k}$
Y_i	Relative PP of τ_i in virtual time (see (6.6))
$\beta_p(t_0,t_1)$	Total time in $[t_0,t_1)$ when P_p is available to level C (see Definition 6.7)
δ	$\min_{\tau_i \in \tau}(x^s_i(1) - x^s_i(s_r))$ (defined in (6.41))
$\Delta(t)$	Function such that $x_i(t) = x^s_i(s_r) + \Delta(t)$ is x-sufficient for t in $[t_r,t_n)$ (see (6.51))
$\Delta^e(t)$	Exponential component of $\Delta(t)$ (see (6.47))
$\Delta^\ell(t)$	Linear component of $\Delta(t)$ (see (6.45))
$\theta_{i,k}$	A set of jobs pending at $t^b_{i,k}$ (see Lemma 6.4)
$\overline{\theta_{i,k}}$	Set of tasks without jobs in $\theta_{i,k}$ (see Lemma 6.4)
κ	Jobs with $y_{i,k} \in [0,t_r)$ and $t^c_{i,k} \in [t_r,\infty)$ (see Definition 6.17)
λ	Constant value of $\Delta(t)$ in $(-\infty,t_r)$ (see (6.40))
ρ	Upper bound on amount of time that a job is pending after t_r or its PP in $[t_r,t_n)$ (see (6.50))
σ_p	Constant used to characterize supply restriction (see Property 6.12)
τ	Set of all level-C tasks
τ_i	Task i
$\tau_{i,k}$	Job k of τ_i
ϕ	Slope of $\Delta^\ell(t)$ (see (6.46))
ψ	Jobs with $t^b_{i,k} \in [0,t_r)$ and $y_{i,k} \in [t_r,\infty)$ (see Definition 6.17)
$\Omega_{i,k}(j)$	Indicator variable to account for pessimism in $W_{i,k}$ (see (6.104))

APPENDIX D: COMPUTING AND ANALYZING $x_I^s(s_I)$

Recall that $x_i^s(s_I)$ is defined in (6.37). As promised in Section 6.3, we now describe how to use linear programming to compute $x_i^s(s_I)$ and prove several results mentioned in that section.

Our technique for computing $x_i^s(s_I)$ involves formulating a linear program very similar to that described in Chapter 3 in the absence of restricted supply.

The formulation of our LP is based on the following theorem, which is similar to Corollary 3.1. We use w instead of s to avoid confusion with $s(t)$ or s_I.

Theorem D.1. *If* $\forall i, u_{tot} - L_i \cdot U_i^v \cdot s_I > 0$ *and*

$$\forall i, x_i^c \triangleq \max\left\{0, \frac{w + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v \cdot s_I}\right\}, \tag{D.1}$$

(choice), where

$$w \triangleq \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^c - S_j) + \sum_{\tau_j \in \tau} S_j, \tag{D.2}$$

then $x_i^s(s_I) = x_i^c$ *satisifies the definition of* $x_i^s(s_I)$ *in* (6.37).

Observe that w *is independent of the task index* i.

Proof. Let $\tau_i \in \tau$ be arbitrary. We consider two cases, one for each term of the max in the definition of x_i^c in the statement of the theorem.

Case 1: $x_i^c = 0$**.** In this case, by the definition of x_i^c in the statement of the lemma and the condition that $u_{tot} - L_i \cdot U_i^v \cdot s_I > 0$,

$$w + (m - u_{tot} - 1)C_i + O^{rn} \leq 0. \tag{D.3}$$

Therefore,

$$\frac{\sum_{m-1 \text{ largest}}(C_j + U_j^v \cdot s_I \cdot x_j^c - S_j) + \sum_{\tau_j \in \tau} S_j + (m - u_{tot} - 1)C_i + O^{rn} + L_i \cdot U_j^v \cdot s_I \cdot x_i^c}{u_{tot}}$$

$$= \{\text{By the definition of } w \text{ in (D.2)}\}$$

$$\frac{w + (m - u_{tot} - 1)C_i + O^{rn} + L_i \cdot U_i^v \cdot s_I \cdot x_i^c}{u_{tot}}$$

$$= \{\text{By the case we are considering}\}$$

$$\frac{w + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot}}$$

$$\leq \{\text{By (D.3)}\}$$

$$0.$$

Thus, $x_i^s(s_I) = x_i^c$ satisfies the definition of $x_i^s(s_I)$ in (6.37) for τ_i.

Case 2: $x_i^c > 0$**.** In this case, by the definition of x_i^c in the statement of the lemma we have

$$x_i^c = \frac{w + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v \cdot s_I}.$$

We manipulate this expression to the form in the definition of $x_i^s(s_I)$ in (6.37). Multiplying both sides by $\frac{u_{tot} - L_i \cdot U_i^v \cdot s_I}{u_{tot}}$ yields

$$\frac{u_{tot} - L_i \cdot U_i^v \cdot s_I}{u_{tot}} \cdot x_i^c = \frac{w + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot}}.$$

Adding $\frac{L_i \cdot U_i^v \cdot s_I}{u_{tot}} \cdot x_i^c$ to both sides yields

$$x_i^c = \frac{w + (m - u_{tot} - 1)C_i + O^{rn} + L_i \cdot U_i^v \cdot s_I \cdot x_i^c}{u_{tot}}.$$

Finally, substituting the expression for w in (D.2),

$$x_i^c = \frac{\sum_{m-1 \text{ largest}}(C_j + U_j^v \cdot s_I \cdot x_j^c - S_j) + \sum_{\tau_j \in \tau} S_j + (m - u_{tot} - 1)C_i + O^{rn} + L_i \cdot U_i^v \cdot s_I \cdot x_i^c}{u_{tot}}.$$

By the case we are considering, both sides of this expression must be greater than zero. Thus, $x_i^s(s_I) = x_i^c$ satisfies the definition of $x_i^s(s_I)$ in (6.37) for τ_i. □

Our LP has, for each τ_i, a variable x_i^c as in Theorem D.1, corresponding to $x_i^s(s_I)$, and an auxilliary variable z_i. Our LP also has task-independent variables w (as in Theorem D.1), G (corresponding to the first sum in the definition of w in (D.2)), S_{sum} (corresponding to the second sum in the definition of w in (D.2)) and auxilliary variable b. All other quantities that appear are constants.

We will present constraint sets for determining x_i^c in the same order as the constraint sets in Section 3.3. The first constraint set ensures that

$$x_i^c \geq \max\left\{0, \frac{w + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v \cdot s_I}\right\}. \tag{D.4}$$

It corresponds to Constraint Set 3.1 that defined x_i, but provides only an inequality. Although Theorem D.1 requires equality, this discrepancy will be handled in Lemma D.3 below.

Constraint Set D.1.

$$\forall i : x_i^c \geq \frac{w + (m - u_{tot} - 1) \cdot C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v \cdot s_I},$$
$$\forall i : x_i^c \geq 0.$$

Because we consider S_i to be a constant, we do not require a constraint that corresponds with Constraint Set 3.2.

The next constraint set is used to determine the value of the first sum in the definition of w in (D.2). This sum corresponds with $G(\vec{x}, \vec{Y})$ defined in (3.12), so the constraint is almost identical to Constraint Set 3.3. As discussed there, this constraint actually ensures that G provides an upper bound on that sum, rather than an exact value. In other words, it actually guarantees that

$$G \geq \sum_{m-1 \text{ largest}} (C_i + U_i^v \cdot s_I \cdot x_i^c - S_i). \tag{D.5}$$

Although Theorem D.1 requires equality, this discrepancy will be handled in Lemmas D.1–D.2 below.

Constraint Set D.2.

$$G = b \cdot (m-1) + \sum_{\tau_i \in \tau} z_i,$$
$$\forall i : z_i \geq 0,$$
$$z_i \geq C_i + U_i^v \cdot s_I \cdot x_i^c - S_i - b.$$

Rather than having a constraint that corresponds to Constraint Set 3.4 (where we viewed the Y_j terms as variables), we define a constant

$$S_{\text{sum}} = \sum_{\tau_j \in \tau} S_j. \tag{D.6}$$

The next constraint provides a bound on the value of w. This constraint differs from the definition of w in (D.2) because it is an inequality. However, we will show in Lemma D.2 below that for an optimal solution to the LP, it reduces to an equality. This constraint corresponds to Constraint Set 3.5.

Constraint Set D.3.

$$w \geq G + S_{\text{sum}}.$$

We must show that, for some appropriate optimization function, an optimal solution to Constraint Sets D.1–D.3 can be used to compute the values of w and x_i^c described in Theorem D.1. We will show below that if we minimize w and an optimal solution is found, we can use the resulting w in the definition of x_i^c in (D.1), and (D.2) must be satisfied as well with that choice of x_i^c.

To do so, we first characterize the value of w for any feasible solution, providing a lower bound in Lemma D.1 and a characterization relating to an optimal value in Lemma D.2. Observe that the expression in Lemma D.1 is identical to the definition of w in (D.2), except that it replaces the equality with an inequality.

Lemma D.1. *For any feasible assignment of variables satisfying Constraint Sets D.1–D.3,*

$$w \geq \sum_{m-1 \text{ largest}} (C_i + U_i^v \cdot s_I \cdot x_i^c - S_i) + \sum_{\tau_j \in \tau} S_i.$$

Proof. We have

$$\begin{aligned}
w \geq\ & \{\text{By Constraint Set D.3}\} \\
& G + S_{\text{sum}} \\
\geq\ & \{\text{By (D.5)}\} \\
& \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^c - S_j) + S_{\text{sum}} \\
=\ & \{\text{By the definition of } S_{\text{sum}} \text{ in (D.6)}\} \\
& \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^c - S_j) + \sum_{\tau_j \in \tau} S_j.
\end{aligned}$$

□

The next lemma will be used to characterize the optimal value of w with an appropriate optimization function.

Lemma D.2. *If V is a feasible assignment of variables that satisfies Constraint Sets D.1–D.3 and*

$$w > \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^c - S_j) + \sum_{\tau_j \in \tau} S_j,$$

then there is also a feasible assignment of variables V' (with variable assignments denoted with primes) such that $w' < w$. In other words, w has not taken its optimal value upon minimization.

Proof. We use the following assignment for V':

$$w' = \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^c - S_j) + S_{\text{sum}}, \tag{D.7}$$

$$\forall i, x_i^{c\prime} = x_i^c, \tag{D.8}$$

$$G' = \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^c - S_j), \tag{D.9}$$

$$b' = (m-1)^{th} \text{ largest value of } C_j + U_j^v \cdot s_I \cdot x_j^c - S_j, \tag{D.10}$$

$$\forall i, z_i' = \max\{0, C_j + U_j^v \cdot s_I \cdot x_j^c - S_j - b'\}. \tag{D.11}$$

By the statement of the lemma, the definition of S_{sum} in (D.6), and by (D.7),

$$w' < w. \tag{D.12}$$

We will first show that Constraint Set D.1 holds by considering arbitrary τ_i.

$$\begin{aligned} x_i^{c\prime} &= \{\text{By (D.8)}\} \\ &\quad x_i^c \\ &\geq \{\text{By (D.4)}\} \\ &\quad \max\left\{0, \frac{w + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v \cdot s_I}\right\} \\ &\geq \{\text{By (D.12)}\} \\ &\quad \max\left\{0, \frac{w' + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v \cdot s_I}\right\}. \end{aligned} \tag{D.13}$$

Constraint Set D.2 holds by (D.8)–(D.11).

To show Constraint Set D.3 holds,

$$\begin{aligned} w' &= \{\text{By (D.7)}\} \\ &\quad \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^c - S_j) + S_{\text{sum}} \end{aligned}$$

$$
\begin{aligned}
&= \{\text{By (D.9)}\}\\
&G' + S_{\text{sum}}. \qquad \square
\end{aligned}
$$

By Lemmas D.1–D.2, we can minimize w as our optimization objective, and the definition of w in Theorem D.1 must be satisfied by the resulting solution. However, the definition of x_i^c in (D.1) is not guaranteed to hold, because Constraint Set D.1 also guaranteed only an inequality. Fortunately, we can use the resulting value of w in (D.1) to compute correct values of x_i^c, as shown in the following lemma.

Lemma D.3. *If V is a feasible assignment of variables satisfying Constraint Sets D.1–D.3, then there is also an assignment V' (with variables denoted with primes, as before) such that $w' = w$ and $\forall i, x_i^{c\prime} = \max\left\{0, \frac{w+(m-u_{tot}-1)C_i+O^{rn}}{u_{tot}-L_i \cdot U_i^v \cdot s_I}\right\}$.*

Proof. We use the following assignment for V':

$$w' = w, \tag{D.14}$$

$$\forall i, x_i^{c\prime} = \max\left\{0, \frac{w+(m-u_{tot}-1)C_i+O^{rn}}{u_{tot}-L_i \cdot U_i^v \cdot s_I}\right\}, \tag{D.15}$$

$$G' = \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^{c\prime} - S_j), \tag{D.16}$$

$$b' = (m-1)^{th} \text{ largest value of } C_j + U_j^v \cdot s_I \cdot x_j^{c\prime} - S_j, \tag{D.17}$$

$$\forall i, z_i' = \max\{0, C_j + U_j^v \cdot s_I \cdot x_j^{c\prime} - S_j - b'\}. \tag{D.18}$$

Constraint Set D.1 holds by (D.15).

Constraint Set D.2 holds by (D.16)–(D.18).

To show that Constraint Set D.3 holds, we first show that for arbitrary j,

$$
\begin{aligned}
x_j^c &\geq \{\text{By (D.4)}\}\\
&\quad \max\left\{0, \frac{w+(m-u_{tot}-1)C_j+O^{rn}}{u_{tot}-L_j \cdot U_j^v \cdot s_I}\right\}\\
&= \{\text{By (D.15)}\}\\
&\quad x_j^{c\prime}.
\end{aligned} \tag{D.19}
$$

Then, we have

$$\begin{aligned}
w' &= \{\text{By (D.14)}\} \\
&\quad w \\
&\geq \{\text{By Lemma D.1}\} \\
&\quad \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^c - S_j) + \sum_{\tau_j \in \tau} S_j \\
&\geq \{\text{By (D.19)}\} \\
&\quad \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^{c\prime} - S_j) + \sum_{\tau_j \in \tau} S_j \\
&= \{\text{By (D.16)}\} \\
&\quad G' + \sum_{\tau_j \in \tau} S_j \\
&= \{\text{By the definition of } S_{\text{sum}} \text{ in (D.6)}\} \\
&\quad G' + S_{\text{sum}}. \qquad \square
\end{aligned}$$

We will next show that, if Property 6.13 holds, a minimum feasible w does in fact exist. While proving this result, we will several times exploit the fact that S_j is nonnegative, as shown now.

Lemma D.4. *For all j, $S_j \geq 0$.*

Proof. We have

$$\begin{aligned}
S_j &= \{\text{By the definition of } S_j \text{ in (6.38)}\} \\
&\quad C_j \cdot \left(1 - \frac{Y_i}{T_i}\right) \\
&\geq \{\text{Because } Y_i \leq T_i\} \\
&\quad C_j \cdot \left(1 - \frac{T_j}{T_j}\right) \\
&= \{\text{Cancelling}\} \\
&\quad 0. \qquad \square
\end{aligned}$$

We now show that a lower bound on w exists for feasible assignments. We will later show that feasible assignments do exist. Together, these results are sufficient to show that an optimal value of w exists.

Lemma D.5. *For any feasible assignment of variables V satisfying Constraint Sets D.1–D.3,*

$$w \geq \sum_{m-1 \text{ largest}} C_j.$$

Proof. We have

$$\begin{aligned}
w \geq & \{\text{By Lemma D.1}\} \\
& \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^c - S_j) + \sum_{\tau_j \in \tau} S_j \\
\geq & \{\text{By Lemma D.4; observe that each } S_j \text{ appearing in the first summation also appears in the second}\} \\
& \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^c) \\
\geq & \{\text{By Constraint Set D.1, which implies each } x_j^c \geq 0\} \\
& \sum_{m-1 \text{ largest}} C_j.
\end{aligned}$$

□

We now show that, given Property 6.13, the LP is feasible.

Lemma D.6. *If Property 6.13 holds, then a feasible assignment of variables V for Constraint Sets D.1–D.3 exists.*

Proof. Let $C_{\max}$ be the largest C_i in the system. For notational convenience, let

$$U_{\text{sum}} \triangleq \sum_{m-1 \text{ largest}} U_j^v, \tag{D.20}$$

$$L_{\max}^t \triangleq \max_{\tau_j \in \tau}(L_j U_j^v) \tag{D.21}$$

($t$erm). We use the following assignment for V:

$$w = \max\left\{C_{\max}, \frac{U_{\text{sum}} \cdot (mC_{\max} + O^{rn}) + (u_{tot} - L_{\max}^t) \cdot ((m-1)C_{\max} + S_{\text{sum}})}{u_{tot} - L_{\max}^t - U_{\text{sum}}}\right\}, \tag{D.22}$$

$$\forall i, x_i^c = \frac{w + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v \cdot s_I}, \tag{D.23}$$

$$G = \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^c - S_j), \tag{D.24}$$

$$b = (m-1)^{th} \text{ largest value of } C_j + U_j^v \cdot s_I \cdot x_j^c - S_j, \tag{D.25}$$

$$\forall i, z_i = \max\{0, C_j + U_j^v \cdot s_I \cdot x_j^c - S_j - b\}. \tag{D.26}$$

To demonstrate that Constraint Set D.1 holds, we first bound the expression that appears in the denominator of the definition of x_i^c in (D.23).

$$\begin{aligned} u_{tot} - L_i \cdot U_i^v \cdot s_I \geq\ & \{\text{Because } s_I \leq 1\} \\ & u_{tot} - L_i \cdot U_i^v \\ \geq\ & \{\text{By the definition of “max”}\} \\ & u_{tot} - \max_{\tau_i \in \tau}(L_i \cdot U_i^v) \\ >\ & \{\text{By Property 6.13}\} \\ & \sum_{m-1 \text{ largest}} U_i^v \\ >\ & \{\text{Because each } U_i^v > 0\} \\ & 0 \end{aligned} \tag{D.27}$$

The first constraint in Constraint Set D.1 holds by (D.23). We now show that the second constraint also holds.

$$\begin{aligned} x_i^c =\ & \{\text{By (D.23)}\} \\ & \frac{w + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v \cdot s_I} \\ \geq\ & \{\text{By (D.22) and (D.27)}\} \\ & \frac{C_{\max} + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v \cdot s_I} \\ \geq\ & \{\text{By the definition of } C_{\max} \text{ and by (D.27)}\} \\ & \frac{(m - u_{tot})C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v \cdot s_I} \\ \geq\ & \{\text{By (D.27), because } C_i \text{ and } O^{rn} \text{ are nonnegative}\} \\ & 0 \end{aligned} \tag{D.28}$$

Constraint Set D.2 holds by (D.24)–(D.26).

By (D.22),

$$w \geq \frac{U_{\text{sum}} \cdot (mC_{\max} + O^{rn}) + (u_{tot} - L^t_{\max}) \cdot ((m-1)C_{\max} + S_{\text{sum}})}{u_{tot} - L^t_{\max} - U_{\text{sum}}}. \tag{D.29}$$

To show that Constraint Set D.3 holds, we start by multiplying both sides of (D.29) by $\frac{u_{tot} - L^t_{\max} - U_{\text{sum}}}{u_{tot} - L^t_{\max}}$, resulting in

$$\frac{u_{tot} - L^t_{\max} - U_{\text{sum}}}{u_{tot} - L^t_{\max}} \cdot w \geq \frac{U_{\text{sum}} \cdot (mC_{\max} + O^{rn})}{u_{tot} - L^t_{\max}} + (m-1)C_{\max} + S_{\text{sum}}.$$

Adding $\frac{U_{\text{sum}}}{u_{tot} - L^t_{\max}} \cdot w$ to both sides yields

$$w \geq \frac{U_{\text{sum}} \cdot (w + mC_{\max} + O^{rn})}{u_{tot} - L^t_{\max}} + (m-1)C_{\max} + S_{\text{sum}}. \tag{D.30}$$

Thus, we have

$$\begin{aligned}
w &\geq \{\text{Rewriting (D.30), and using (D.20)}\} \\
&\quad \sum_{m-1 \text{ largest}} \left(C_{\max} + U^v_j \cdot \frac{w + mC_{\max} + O^{rn}}{u_{tot} - L^t_{\max}} \right) + S_{\text{sum}} \\
&\geq \{\text{By the definition of } L^t_{\max} \text{ in (D.21), and because } s_I \leq 1\} \\
&\quad \sum_{m-1 \text{ largest}} \left(C_{\max} + U^v_j \cdot \frac{w + mC_{\max} + O^{rn}}{u_{tot} - L_j \cdot U^v_j \cdot s_I} \right) + S_{\text{sum}} \\
&\geq \{\text{By (D.27) and the definition of } C_{\max}\} \\
&\quad \sum_{m-1 \text{ largest}} \left(C_j + U^v_j \cdot \frac{w + mC_j + O^{rn}}{u_{tot} - L_j \cdot U^v_j \cdot s_I} \right) + S_{\text{sum}} \\
&\geq \{\text{Because } C_j > 0,\ u_{tot} > 0, \text{ and by (D.27)}\} \\
&\quad \sum_{m-1 \text{ largest}} \left(C_j + U^v_j \cdot \frac{w + (m - u_{tot} - 1)C_j + O^{rn}}{u_{tot} - L_j \cdot U^v_j \cdot s_I} \right) + S_{\text{sum}} \\
&= \{\text{By (D.23)}\} \\
&\quad \sum_{m-1 \text{ largest}} \left(C_j + U^v_j \cdot x^c_i \right) + S_{\text{sum}} \\
&\geq \{\text{Because } 0 < s_I \leq 1 \text{ and by (D.28)}\} \\
&\quad \sum_{m-1 \text{ largest}} \left(C_j + U^v_j \cdot s_I \cdot x^c_i \right) + S_{\text{sum}}
\end{aligned}$$

$$\geq \{\text{By Lemma D.4}\}$$

$$\sum_{m-1 \text{ largest}} \left(C_j + U_j^v \cdot s_I \cdot x_i^c - S_j\right) + S_{\text{sum}}$$

$$= \{\text{By (D.24)}\}$$

$$G + S_{\text{sum}}. \qquad \square$$

We combine the previous results into the main result of this appendix.

Theorem D.2. *If $\forall i, u_{tot} - L_i \cdot U_i^v \cdot s_I > 0$, then all $x_i^s(s_I)$ values can be computed by minimizing w subject to Constraint Sets D.1–D.3, followed by using the assignment of x_i^c in* (D.1).

Proof. By Lemma D.6, Constraint Sets D.1–D.3 are feasible. By Lemma D.5 and the linearity of Constraint Sets D.1–D.3, a unique minimum value of w exists. Without loss of generality, let V be an assignment that achieves such a value. Then, by Lemma D.3, there is an assignment V' such that $w' = w$ and

$$\forall i, x_i^{c\prime} = \max\left\{0, \frac{w + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v \cdot s_I}\right\} \tag{D.31}$$

(D.31) is simply (D.1) with $x_i^{c\prime}$ in place of x_i^c. Furthermore, applying Lemmas D.1 and D.2 and the fact that $w' = w$ is the minimum possible value, we have

$$w = \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I \cdot x_j^{c\prime} - S_j) + \sum_{\tau_j \in \tau} S_j.$$

Therefore, applying Theorem D.1 with $x_i^c = x_i^{c\prime}$, we have $x_i^s(s_I) = x_i^{c\prime}$. □

The one promised result that remains to be shown is that a feasible solution exists for any $s_I' \leq 1$ as long as one exists for $s_I = 1$. We show that result now.

Theorem D.3. *If a feasible assignment of variables V satisfies Constraint Sets D.1–D.3 with $s_I = 1$ and $u_{tot} - L_i \cdot U_i^v > 0$, then a feasible assignment of variables V' exists that satisfies Constraint Sets D.1–D.3 for arbitary $s_I' \leq 1$.*

Proof. We use the following assignment for V':

$$w' = w, \tag{D.32}$$

$$\forall i, x_i^{c\prime} = x_i^c, \tag{D.33}$$

$$G' = \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I' \cdot x_j^c - S_j), \tag{D.34}$$

$$b' = (m-1)^{th} \text{ largest value of } C_j + U_j^v \cdot s_I' \cdot x_j^c - S_j, \tag{D.35}$$

$$\forall i, z_i' = \max\{0, C_j + U_j^v \cdot s_I' \cdot x_j^c - S_j - b'\}. \tag{D.36}$$

Because $u_{tot} - L_i \cdot U_i^v > 0$ and $0 < s_I \leq 1$,

$$u_{tot} - L_i \cdot s_I \cdot U_i^v > 0. \tag{D.37}$$

To show that Constraint Set D.1 holds, we consider two cases for each τ_i.

Case 1: $w + (m - u_{tot} - 1)C_i + O^{rn} \leq 0$. In this case,

$$\begin{aligned}
x_i^{c\prime} &= \{\text{By (D.33)}\} \\
&\quad x_i^c \\
&\geq \{\text{By Constraint Set D.1}\} \\
&\quad 0 \\
&= \{\text{By (D.37) and the case we are considering}\} \\
&\quad \max\left\{0, \frac{w + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot s_I \cdot U_i^v}\right\}.
\end{aligned}$$

Case 2: $w + (m - u_{tot} - 1)C_i + O_{i,k} > 0$. In this case,

$$\begin{aligned}
x_i^{c\prime} &= \{\text{By (D.33)}\} \\
&\quad x_i^c \\
&\geq \{\text{By (D.4) with } s_I = 1\} \\
&\quad \max\left\{0, \frac{w + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v}\right\} \\
&= \{\text{By (D.32)}\} \\
&\quad \max\left\{0, \frac{w' + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot U_i^v}\right\}
\end{aligned}$$

$$\begin{aligned}
&\geq \{\text{By (D.37) and the case we are considering, because } 0 < s_I < 1\} \\
&\quad \max\left\{0, \frac{w' + (m - u_{tot} - 1)C_i + O^{rn}}{u_{tot} - L_i \cdot s_I \cdot U_i^v}\right\}.
\end{aligned}$$

Constraint Set D.2 holds by (D.33)–(D.36).

To show Constraint Set D.3 holds,

$$\begin{aligned}
w' &= \{\text{By (D.32)}\} \\
&\quad w \\
&\geq \{\text{By Lemma D.1 with } s_I = 1\} \\
&\quad \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot x_j^c - S_j) + S_{\text{sum}} \\
&\geq \{\text{Because } 0 < s_I' \leq 1\} \\
&\quad \sum_{m-1 \text{ largest}} (C_j + U_j^v \cdot s_I' \cdot x_j^c - S_j) + S_{\text{sum}} \\
&= \{\text{By (D.34)}\} \\
&\quad G' + S_{\text{sum}}. \qquad \square
\end{aligned}$$

Finally, we briefly discuss the use of linear programming to determine a choice of Y_i. The value of Y_i in our analysis is assumed to remain constant when the virtual clock speed changes. Furthermore, during the typical behavior of a system, $s_I = 1$ should be used. Therefore, if using linear programming to determine the best choice of Y_i, it should be done using $s_I = 1$.

Up to this point, Y_i has been assumed to be a constant. Similarly, S_i, which depends on Y_i by the definition of S_i in (6.38), and S_{sum}, which depends on S_i by the definition of S_{sum} in (D.6), have also been considered to be constants. These can be changed to variables as long as the following constraint sets are added. The first constrains the choice of Y_i itself to match the assumptions used in our analysis.

Constraint Set D.4.

$$\forall i, Y_i \geq 0,$$

$$\forall i, T_i \geq Y_i.$$

The next constraint set simply specifies the value of S_i according to the definition of S_i in (6.38).

Constraint Set D.5.

$$\forall i, S_i = C_i \cdot \left(1 - \frac{Y_i}{T_i}\right).$$

The final constraint set specifies the value of S_{sum} according to the definition of S_{sum} in (D.6).

Constraint Set D.6.

$$S_{\text{sum}} = \sum_{\tau_j \in \tau} S_j.$$

Any optimization function can be used that ensures, under an optimal solution, the minimal value of w with respect to the chosen values of Y_i. If the optimization function has such a property, then all of the reasoning in this appendix will continue to hold. Because each x_i^c cannot increase as a result of a decrease in w, such a property is easy to achieve for a reasonable optimization function that attempts to minimize x_i^c values. If w is not actually guaranteed to be minimized, the resulting Y_i values can then be used with the method described in Theorem D.2. This can only result in better response-time bounds.

BIBLIOGRAPHY

Anderson, J. and Srinivasan, A. (2000). Early-release fair scheduling. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 35–43.

Anderson, J. H., Bud, V., and Devi, U. C. (2008). An EDF -based restricted-migration scheduling algorithm for multiprocessor soft real-time systems. *Real-Time Systems*, 38(2):85–131.

Anderson, J. H., Erickson, J. P., Devi, U. C., and Casses, B. N. (2014). Optimal semi-partitioned scheduling in soft real-time systems. In *Proceedings of the 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. To appear.

Anderson, J. H. and Srinivasan, A. (2004). Mixed pfair/erfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*, 68(1):157–204.

Aydin, H., Melhem, R., Mosse, D., and Mejia-Alvarez, P. (2001). Optimal reward-based scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 50(2):111–130.

Baker, T. P. and Baruah, S. K. (2009). An analysis of global EDF schedulability for arbitrary-deadline sporadic task systems. *Real-Time Systems*, 43(1):3–24.

Baker, T. P., Cirinei, M., and Bertogna, M. (2008). Edzl scheduling analysis. *Real-Time Systems*, 40(3):264–289.

Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Van der Ster, S., and Stougie, L. (2012). The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems*, pages 145–154.

Baruah, S., Cohen, N., Plaxton, C., and Varvel, D. (1996). Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625.

Baruah, S., Koren, G., Mishra, B., Raghunathan, A., Rosier, L., and Shasha, D. (1991). On-line scheduling in the presence of overload. In *Proceedings of the 32nd Annual Symposium On Foundations of Computer Science*, pages 100–110.

Baruah, S., Mok, A., and Rosier, L. (1990). Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 182–190.

Baruah, S. K., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N., and Stougie, L. (2010). Scheduling real-time mixed-criticality jobs. In Hliněený, P. and Kučera, A., editors, *Mathematical Foundations of Computer Science 2010*, volume 6281 of *Lecture Notes in Computer Science*, pages 90–101. Springer Berlin Heidelberg.

Bastoni, A., Brandenburg, B., and Anderson, J. (2010). An empirical comparison of global, partitioned, and clustered multiprocessor edf schedulers. In *Proceedings of the 31st Real-Time Systems Symposium*, pages 14–24.

Bastoni, A., Brandenburg, B., and Anderson, J. (2011). Is semi-partitioned scheduling practical? In *Proceedings of the 23rd Euromicro Conference on Real-Time Systems*, pages 125–135.

Bernat, G., Burns, A., and Llamosi, A. (2001). Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321.

Block, A. (2008). *Adaptive Multiprocessor Real-Time Systems*. PhD thesis, The University of North Carolina at Chapel Hill.

Bougueroua, L., George, L., and Midonnet, S. (2007). Dealing with execution-overruns to improve the temporal robustness of real-time systems scheduled FP and EDF. In *Proceedings of the 2nd International Conference on Systems*, pages 52–52.

Brandenburg, B. B. (2011). *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, The University of North Carolina at Chapel Hill.

Brandenburg, B. B. and Anderson, J. H. (2007). Feather-trace: A light-weight event tracing toolkit. In *Proceedings of the 3rd Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pages 61–70.

Buttazzo, G. C., Lipari, G., Caccamo, M., and Abeni, L. (2002). Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302.

Buttazzo, G. C. and Stankovic, J. A. (1995). Adding robustness in dynamic preemptive scheduling. In Fussell, D. S. and Malek, M., editors, *Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems*, volume 297 of *The Springer International Series in Engineering and Computer Science*, pages 67–88. Springer US.

Chakraborty, S., Kunzli, S., and Thiele, L. (2003). A general framework for analysing system properties in platform-based embedded system designs. In *Proceedings of the 2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 190–195.

Cho, H. (2006). *Utility Accrual Real-Time Scheduling and Synchronization on Single and Multiprocessors: Models, Algorithms, and Tradeoffs*. PhD thesis, Virginia Polytechnic Institute and State University.

Clark, R. K. (1990). *Scheduling Dependent Real-Time Activities*. PhD thesis, Carnegie Mellon University.

Devi, U. C. and Anderson, J. H. (2008). Tardiness bounds under global edf scheduling on a multiprocessor. *Real-Time Systems*, 38(2):133–189.

Durbin, M. (2010). *All About High-Frequency Trading*. McGraw-Hill, 1 edition.

Elliott, G. A., Kim, N., Erickson, J. P., Liu, C., and Anderson, J. H. (2014). Minimizing response times of automotive dataflows on multicore. In *Proceedings of the 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. To appear.

Erickson, J., Coombe, G., and Anderson, J. (2012). Soft real-time scheduling in google earth. In *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 141–150.

Garyali, P. (2010). On best-effort utility accrual real-time scheduling on multiprocessors. Master's thesis, The Virginia Polytechnic Institute and State University.

Ghazalie, T. M. and Baker, T. P. (1995). Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9(1):31–67.

Hamdaoui, M. and Ramanathan, P. (1995). A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Transactions on Computers*, 44(12):1443–1451.

Herman, J., Kenna, C., Mollison, M., Anderson, J., and Johnson, D. (2012). Rtos support for multicore mixed-criticality systems. In *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 197–208.

Jan, M., Zaourar, L., and Pitel, M. (2013). Maximizing the execution rate of low-criticality tasks in mixed criticality systems. In *Proceedings of the 1st Workshop on Mixed Criticality Systems*, pages 43–48.

Koren, G. and Shasha, D. (1994). MOCA: a multiprocessor on-line competitive algorithm for real-time system scheduling. *Theoretical Computer Science*, 128(1–2):75–97.

Koren, G. and Shasha, D. (1995a). Skip-over: algorithms and complexity for overloaded systems that allow skips. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 110–117.

Koren, G. and Shasha, D. (1995b). D^{over}: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24(2):318–339.

Lee, S. K. (1994). On-line multiprocessor scheduling algorithms for real-time tasks. In *Proceedings of IEEE Region 10's Ninth Annual International Conference*, pages 607–611 vol.2.

Leontyev, H. and Anderson, J. H. (2010). Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Systems*, 44(1-3):26–71.

Leontyev, H., Chakraborty, S., and Anderson, J. H. (2011). Multiprocessor extensions to real-time calculus. *Real-Time Systems*, 47(6):562–617.

Li, H. and Baruah, S. (2012). Global mixed-criticality scheduling on multiprocessors. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems*, pages 166–175.

Li, P. (2004). *Utility Accrual Real-Time Scheduling: Models and Algorithms*. PhD thesis, Virginia Polytechnic Institute and State University.

Li, P., Wu, H., Ravindran, B., and Jensen, E. (2006). A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints. *Computers, IEEE Transactions on*, 55(4):454–469.

Lin, K. and Natarajan, S. (1988). Expressing and maintaining timing constraints in flex. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, pages 96–105.

Liu, C. and Anderson, J. (2010). Supporting soft real-time dag-based systems on multiprocessors with no utilization loss. In *Proceedings of the 31st IEEE Real-Time Systems Symposium*, pages 3–13.

Liu, C. and Anderson, J. (2011). Supporting graph-based real-time applications in distributed systems. In *Proceedings of the 17th IEEE Embedded and Real-Time Computing Systems and Applications*, pages 143–152.

Liu, J. W., Lin, K., Shih, W., Yu, A., Chung, J., and Zhao, W. (1991). Algorithms for scheduling imprecise computations. *Computer*, 24(5):58–68.

Locke, C. (1986). *Best-Effort Decision Making for Real-Time Scheduling*. PhD thesis, Carnegie Mellon University.

Megel, T., Sirdey, R., and David, V. (2010). Minimizing task preemptions and migrations in multiprocessor optimal real-time schedules. In *Proceedings of the 31st IEEE Real-Time Systems Symposium*, pages 37–46.

Mollison, M. S., Erickson, J. P., Anderson, J. H., Baruah, S. K., and Scoredos, J. A. (2010). Mixed-criticality real-time scheduling for multicore systems. In *Proceedings of the IEEE International Conference on Embedded Software and Systems*, pages 1864–1871, Washington, DC, USA. IEEE Computer Society.

Ogryczak, W. and Tamir, A. (2003). Minimizing the sum of the *k* largest functions in linear time. *Information Processing Letters*, 85(3):117–122.

Regnier, P., Lima, G., Massa, E., Levin, G., and Brandt, S. (2011). Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS)*, pages 104–115.

Santy, F., George, L., Thierry, P., and Goossens, J. (2012). Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems*, pages 155–165.

Santy, F., Raravi, G., Nelissen, G., Nelis, V., Kumar, P., Goossens, J., and Tovar, E. (2013). Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, pages 183–192, New York, NY, USA. ACM.

Sha, L., Rajkumar, R., and Lehoczky, J. P. (1990). Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185.

Spuri, M., Buttazzo, G., and Sensini, F. (1995). Robust aperiodic scheduling under dynamic priority systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 210–219.

Stoica, I., Abdel-Wahab, H., Jeffay, K., Baruah, S., Gehrke, J., and Plaxton, C. (1996). A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 288–299.

Su, H. and Zhu, D. (2013). An elastic mixed-criticality task model and its scheduling algorithm. In *Proceedings of the 2013 Design, Automation Test in Europe Conference Exhibition*, pages 147–152.

Su, H., Zhu, D., and Mosse, D. (2013). Scheduling algorithms for elastic mixed-criticality tasks in multicore systems. In *Proceedings of the 19th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 352–357.

Vestal, S. (2007). Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium*, pages 239–243.

West, R. and Poellabauer, C. (2000). Analysis of a window-constrained scheduler for real-time and best-effort packet streams. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pages 239–248.

Zhang, L. (1990). Virtual clock: A new traffic control algorithm for packet switching networks. In *Proceedings of the 5th ACM Symposium on Communications Architectures & Protocols*, pages 19–29, New York, NY, USA. ACM.

Zhu, D. and Aydin, H. (2009). Reliability-aware energy management for periodic real-time tasks. *IEEE Transactions on Computers*, 58(10):1382–1397.