# Optimizing Lock Granularity for Non-Nested Resource Access under the Priority-Ceiling Protocol

Catherine E. Nemitz
*Davidson College*
Davidson, North Carolina, USA
canemitz@davidson.edu

Tanya Amert
*Carleton College*
Northfield, Minnesota, USA
tamert@carleton.edu

***Abstract*—When multiple tasks require mutually exclusive access to shared resources, a trade-off arises between the duration of time that one task holds a resource (*i.e.* the duration of a critical section) and the blocking suffered by other tasks. Prior work has explored this trade-off in the context of a single shared resource. In this paper, the decision of grouping individual accesses into critical sections is formulated as an optimization problem for systems with a single shared resource. This formulation is also extended to a limited setting with multiple resources in which nested resource access is disallowed. The scale of the resulting optimization problem is explored via experiments with synthetically generated task sets. Additionally, a discussion is presented of the challenges in extending this work to general nesting of shared-resource accesses.**

***Index Terms*—Real-time systems, shared resources, mutual exclusion, uniprocessor scheduling, schedulability**

## I. Introduction

Tasks in a hard real-time system may require the use of *shared resources* including hardware accelerators and regions of shared memory. Each task may make multiple accesses to a given resource, as is common in applications relying on access to a graphics processing unit (GPU). Access to these resources must be protected to ensure anticipated timing behavior and correct execution of all tasks. A task executes within a *critical section* when it is guaranteed protected access to a resource. A critical section may include multiple accesses to a resource, possibly with interleaving intermediate computations. Defining which access(es) belong to each critical section can be done on a per task basis and ensure correct execution behavior.

Unfortunately, in a real-time system, deciding the assignment of resource accesses to critical sections in isolation can result in a system that is unschedulable, *i.e.* unable to meet its timing requirements. The time when a task is executing in a critical section can delay the execution of higher-priority tasks, causing *priority-inversion blocking (blocking)*. Therefore, conventional wisdom suggests making each critical section as short as possible, *i.e.* containing as few accesses as possible.

Maintaining short critical sections requires a task to have more critical sections than if accesses were further combined. However, when locks are used to protect resource access, there is time associated with each acquisition and release of the lock. Furthermore, when a task accesses a resource after use of that resource by a different task, it may experience a longer execution time than when repeatedly using the resource without an interleaving task; this behavior has been observed in the use of GPUs [2]. We refer to any such delays that are incurred when resource accesses are separated into different critical sections as *overhead*. Combining multiple accesses into a single critical section reduces the overhead incurred by a task and, in turn, reduces its total execution time.

Therefore, there is a tension between reducing the delays induced on higher-priority tasks (by having few accesses per critical section) and reducing the execution time, which also allows more execution of lower-priority tasks (by having many accesses per critical section). Prior work has shown how to optimally assign accesses to critical sections for fixed-priority systems with only a single shared resource protected by the Priority Inheritance Protocol by using a greedy algorithm [2].

In this work, we formulate this problem of assigning resource accesses to critical sections as an optimization problem. This formulation provides a framework for more complex resource-sharing systems; we additionally formulate the optimization problem for systems with multiple shared resources with non-overlapping critical sections, and discuss the extension to overlapping (*i.e.*, nested) critical sections.

## II. Background

We now introduce our task and resource model, detail our scheduling and synchronization assumptions, and discuss prior work on critical-section granularity as well as response-time analysis formulations as optimization problems.

**Task model.** We consider a task system $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ of $n$ tasks. Each task $\tau_i$ releases a (potentially infinite) sequence of jobs, with jobs spaced by a minimum inter-arrival time of $T_i$ time units; we refer to a generic job of task $\tau_i$ as $J_i$. A job $J_i$ of task $\tau_i$ executes for at most its worst-case execution time (WCET) of $C_i$ time units, and must complete execution within a relative deadline of $D_i \leq T_i$ time units after its release. We define the response time of a job $J_i$ to be the time between its release and completion, and let $R_i$ be the worst-case response time of any job $J_i$ of task $\tau_i$.

**Resource model.** Some tasks may require access to shared resources. When job $J_i$ requires access to a shared resource $\mathcal{L}_k$, it issues an *acquisition request*. Job $J_i$ *holds* the resource from the time the acquisition request is *satisfied* until job $J_i$ issues a *release request*. We say that a job experiences *priority-*

*inversion blocking* (blocking) at any time instant that it has an unsatisfied acquisition request due to a resource held by a lower-priority job. The instructions executed while job $J_i$ holds resource $\mathcal{L}_k$ comprise a critical section.

**Uniprocessor scheduling.** We focus on uniprocessor platforms scheduled using a fixed-priority scheduling policy; we assume that tasks are indexed in order of decreasing priority, *e.g.* that task $\tau_1$ has higher priority than task $\tau_2$.

**Synchronization protocols.** Requests to acquire and release shared resources can be managed via synchronization protocols, such as the Priority Ceiling Protocol (PCP) [11]. Under the PCP, the highest priority of any task that accesses a given resource $\mathcal{L}_k$ is called the *priority ceiling* of that resource and denoted $\text{PC}(k)$. When a job $J_i$ issues a request, it must wait until its priority exceeds that of the highest priority ceiling of any held resource $\mathcal{L}_k$. Thus, $J_i$ may be blocked when a resource's priority ceiling is at most the priority of task $\tau_i$ (*i.e.* $\text{PC}(k) \leq i$).

A job $J_i$ may experience blocking from a lower-priority job $J_j$ holding a resource with a higher priority ceiling than the priority of $J_i$. Even if $J_i$ does not access any resource, it may be blocked by $J_j$ executing a critical section when $J_j$ has inherited the priority from a blocked task with higher priority than $J_i$. A job may by blocked for the duration of at most one such critical section, regardless of the number of requests it makes or the number of resources it requires (Theorem 12 from [11]). If there is only one resource in the system, this is also true of the Priority Inheritance Protocol (PIP). As with the PCP, the PIP enforces that a job $J_i$ holding a resource inherits the priority of the highest priority job blocked by $J_i$. Considerations of a priority ceiling are not included in the PIP.

**Access model and critical-section granularity.** We define an *access* to a shared resource to be the smallest sequence of instructions that together require mutually exclusive use of that resource. We assume that each instruction within a job is part of either an *access segment* utilizing a single shared resource[1] or a *non-access segment* that does not require any shared resources. We let $A_i$ and $\Gamma_i$ denote the tuples of access and non-access segments of task $\tau_i$, respectively. Similarly, we label the $\nu^{th}$ access segment of a job of task $\tau_i$ as $\alpha_i^\nu$, and use $\gamma_i^\nu$ to indicate the following non-access segment[2]. We use $|\alpha_i^\nu|$ (resp., $|\gamma_i^\nu|$) to indicate the duration of a given access (resp., non-access) segment[3]. See Fig. 1a for an illustration of the access and non-access segments of an example task.

A critical section is therefore a sequence of access and non-access segments. A critical section that contains accesses for only a single resource is said to be *non-nested*, while a critical section that contains accesses for multiple resources is said to be *nested*.

**Example 1.** *The assignment of access segments to critical sections depicted in Fig. 1b–d illustrates how some non-access segments may be included in a critical section. All critical sections shown in Fig. 1 are non-nested.*

*In contrast, consider the task depicted in Fig. 2a. Here, each access is annotated with which resource is required. Fig. 2b illustrates one possible assignment of accesses to critical section, in which the second critical section is nested; the task requires access to multiple resources during its execution. Instead, Fig. 2c shows an assignment that results in only non-nested critical sections.*

[1]We explore in Sec. III-C the implications of relaxing this assumption.
[2]Thus, non-access segments are indexed from 0, access segments from 1.
[3]A non-access segment may have zero duration.

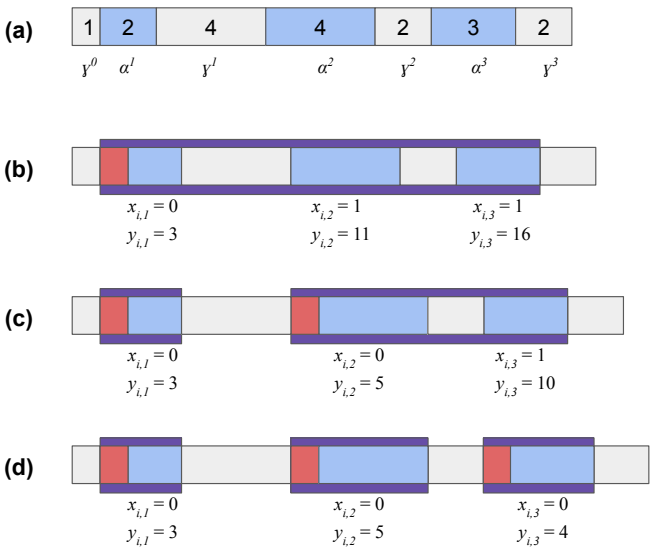


Fig. 1. An example task with (a) access and non-access segments annotated with durations and (b–d) possible assignments of accesses to critical sections with the associated overhead. Access and non-access segments are depicted as blue and gray boxes, respectively. Critical sections are indicated with larger purple boxes, with the per-critical-section overhead shown as red boxes.

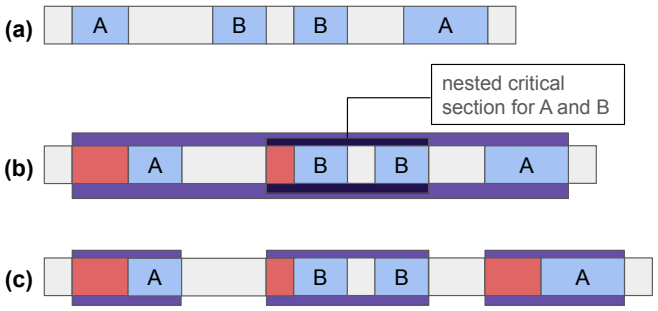


Fig. 2. Illustration of (a) a task with accesses for multiple shared resources, (b) an allocation of those accesses to nested critical sections, and (c) an allocation of accesses to non-nested critical sections.

We assume the acquisition of a resource incurs some additional *overhead*, which may be caused by factors such as the synchronization protocol or the hardware itself, such as the observed increased running time for the first access running on a GPU [2]. The overhead associated with a given resource $\mathcal{L}_k$ is denoted $\mathcal{O}_k$, which we simplify for single-resource systems to $\mathcal{O}$. The overhead at the start of each critical section is

depicted in Figs. 1 and 2.

The *granularity* of a critical section is the degree to which multiple access segments are coalesced into the critical section; coarse-grained critical sections are therefore those containing many access segments, whereas the finest-grained critical section is one composed of a single access segment. Recent work [2] has demonstrated that the granularity of lock acquisition and release requests can impact both response times and schedulability. This impact comes from both the differing resulting critical-section durations and the different number of times overhead is incurred, which in turn impacts the WCET.

**Response-time analysis.** The schedulability of a fixed-priority system can be determined by conducting response-time analysis to check if the worst-case response time of any task exceeds its deadline. This approach has also been encoded as an Integer Linear Program (ILP) by Baruah and Ekberg [4]. We build on this approach and thus present the original ILP here.

The response time for a task $\tau_i$ is modeled by the variable $R_i$. In order for the task system to be schedulable, the response time of each task must be at most its deadline. This is encoded in the first constraint.

$$\forall i \;\; R_i \leq D_i \tag{1}$$

The integer variable $Z_{i,j}$ captures the number of executions a higher priority task $\tau_j$ may need to complete during the period of a task $\tau_i$.

$$\forall i, \forall j < i \;\; Z_{i,j} \geq \frac{R_i}{T_j} \tag{2}$$

For task systems without the blocking caused by shared resources, the response time can then be constrained as mirrors the typical response time analysis expression [8].

$$\forall i \;\; C_i + \sum_{j=1}^{i-1} Z_{i,j} * C_j \leq R_i$$

We update this expression in Sec. III-A to include blocking.

If any feasible solution to these constraints is found, then the task system is schedulable: the value assigned to each $R_i$ must account for the impact of higher-priority workload and ensure $R_i \leq D_i$. Minimizing the sum of all response times forces each $R_i$ to its minimal value, which then corresponds to the response time of each task as computed with traditional response time analysis [8].

$$\text{minimize} \sum_i R_i$$

## III. Formulating the Optimization Problem

In this section, we present the formulation of the lock granularity problem. We begin with constraints on schedulability and then incorporate blocking and the decision of how to group resource accesses for a single resource. Then, we expand on this approach to support multiple resources in a limited setting.

### A. Problem Formulation for a Single Resource

We first consider a system with a single shared resource. We use Eq. (1) and Eq. (2) without modification; deadlines must be met and the same considerations about higher priority workload exist when adding a shared resource to a system.

**Incorporating blocking.** In the presence of shared resources, blocking must be accounted for in the response time analysis. We therefore introduce the variable $B_i$ to represent the maximum blocking that task $\tau_i$ may incur. This method of accounting for blocking mirrors how it is typically handled in response time analysis [8].

$$\forall i \;\; C_i + B_i + \sum_{j=1}^{i-1} Z_{i,j} * C_j \leq R_i \tag{3}$$

Next, we seek to bound the blocking $B_i$ experienced by task $\tau_i$. In a system with only a single resource, we let PC represent the priority ceiling of that resource. Recall that a job $J_i$ can only be blocked if task $\tau_i$ has priority index at least that of the priority ceiling of a resource and $J_i$ is being delayed by a job of a lower-priority task[4].

To ensure $B_i$ correctly accounts for the maximum possible blocking, we constrain it to be at least as large as any individual critical section that could cause blocking. Note that minimizing the sum of all $R_i$ values in conjunction with Eq. (3) effectively searches for the minimum allowed value for each $B_i$. The minimum possible value for $B_i$ is the duration of the longest critical section of a lower priority task, matching the worst-case behavior under the PCP. We define the variable $L_i^{max}$ to represent the maximum duration of any critical section of a job $J_i$.

$$\forall i \geq \text{PC}, \forall j > i \;\; B_i \geq L_j^{\max} \tag{4}$$

**Grouping accesses into critical sections.** The above constraints all follow directly from the application of the scheduling policy and resource access protocol. Note, however, that the critical-section durations depend on how resource accesses are grouped. To encode this decision, we introduce the binary decision variable $x_{i,\nu}$, for which a value of 1 indicates that the $\nu^{\text{th}}$ access segment $\alpha_i^\nu$ of $\tau_i$ is combined with the prior access segment $\alpha_i^{\nu-1}$ and the intermediate non-access segment $\gamma_i^{\nu-1}$ into the same critical section.

By definition, the first access of a task cannot be combined with any prior access.

$$\forall i, |A_i| \neq 0 \;\; x_{i,1} \leq 0 \tag{5}$$

The solution of the model and resulting assignment of values to the remaining decision variables define the critical sections in a recursive manner.

**Example 2.** *Consider the task $\tau_i$ depicted in Fig. 1a, with $\Gamma_i = (1, 4, 2, 2)$ and $A_i = (2, 4, 3)$. By Eq. (5), we must have $x_{i,1} = 0$. The solution of the optimization problem will assign values to $x_{i,2}$ and $x_{i,3}$, with four possible assignments in total;*

[4]Note that a larger index $j > i$ indicates $J_j$ has a lower priority than $J_i$.

*three of these are depicted for a scenario with $\mathcal{O} = 1$ in Fig. 1b-d. For example, Fig. 1c depicts a scenario with $x_{i,2} = 0$ and $x_{i,3} = 1$, indicating that access $\alpha_i^2$ is not combined with $\alpha_i^1$ (therefore $\alpha_i^1$ composes its own critical section), and that $\alpha_i^3$ is combined in a critical section with $\alpha_i^2$, respectively.*

The value assigned to a given $x_{i,\nu}$ impacts the duration of the critical section containing access $\alpha_i^\nu$. As such, we define the variable $y_{i,\nu}$ to represent the duration of a given critical section up through the $\nu^{\text{th}}$ access. As mentioned above, the first access of a task cannot be combined with an earlier task. Thus, the duration of the critical section through that access is simply the access duration summed with the overhead.

**Example 3.** *For the task depicted in Fig. 1b-d, $x_{i,1} = 0$, so $y_{i,1} = |\alpha_i^1| + \mathcal{O} = 2 + 1 = 3$.*

We consider the first access of a task separately from later accesses.

$$\forall i, |A_i| \neq 0 \;\; y_{i,1} \geq |\alpha_i^1| + \mathcal{O} \tag{6}$$

Next we consider the implications of the two possible values for an arbitrary $x_{i,\nu}$ on the value $y_{i,\nu}$. If $x_{i,\nu} = 0$, then the $\nu^{\text{th}}$ access is the first of a critical section. As such, the critical-section duration through that access is only the duration of the access plus the overhead.

**Example 4.** *In Fig. 1d, $x_{i,2} = 0$. Therefore, $y_{i,2} = |\alpha_i^2| + \mathcal{O} = 4 + 1 = 5$.*

Alternatively, if $x_{i,\nu} = 1$, then an earlier access began this critical section and the overhead has already been included. Thus, the duration through this access is given by the previous duration value plus the durations of the interleaving non-access and the $\nu^{\text{th}}$ access: $y_{i,\nu-1} + |\gamma_i^{\nu-1}| + |\alpha_i^\nu|$.

**Example 5.** *For the scenario depicted in Fig. 1b, $x_{i,2} = 1$, and thus $y_{i,2} = y_{i,1} + |\gamma_i^1| + |\alpha_i^2| = 3 + 4 + 4 = 11$.*

These two possibilities for the constraint on $y_{i,\nu}$ described above are applied with the indicator variable for whether $\alpha_i^\nu$ is grouped with the previous access:

$$\begin{aligned} &\forall i, |A_i| \neq 0, \forall \nu > 1 \\ &y_{i,\nu} \geq x_{i,\nu} \cdot \left(y_{i,\nu-1} + |\gamma_i^{\nu-1}|\right) + |\alpha_i^\nu| + (1 - x_{i,\nu}) \cdot \mathcal{O} \end{aligned} \tag{7}$$

**Example 6.** *Revisiting the task depicted in Fig. 1, we have $y_{i,2} \geq x_{i,2} \cdot (y_{i,1} + |\gamma_i^1|) + |\alpha_i^2| + (1 - x_{i,2}) \cdot \mathcal{O} = x_{i,2} \cdot (3+4) + 4 + (1 - x_{i,2}) \cdot 1$. In Fig. 1b, $x_{i,2} = 1$, resulting in $y_{i,2} \geq 11$, whereas in Fig. 1c-d, $x_{i,2} = 0$, resulting in $y_{i,2} \geq 5$.*

**Bounding critical-section and job durations.** The values of $y_{i,\nu}$ allow the maximum critical-section duration to be found, which is necessary to compute the blocking in Eq. (4). Indeed, the maximum critical-section duration is at least any partial critical-section duration.

$$\forall i, \forall \nu \;\; L_i^{\max} \geq y_{i,\nu} \tag{8}$$

**Example 7.** *For the task depicted in Fig. 1, Eq. (8) results in three inequalities: $L_i^{max} \geq y_{i,1}$, $L_i^{max} \geq y_{i,2}$, and $L_i^{max} \geq y_{i,3}$. For Fig. 1b, this enforces $L_i^{max} \geq 16$, whereas in the situation depicted in Fig. 1c, the constraints capture the smaller critical sections, enforcing $L_i^{max} \geq 10$ as the largest possible critical-section duration.*

Finally, recall that the motivation to consider combining resource accesses is to reduce the number of times overhead must be incurred, thereby reducing the total WCET, which we represent with the variable $C_i$. The resulting WCET must include all access and non-access durations, as well as overhead for each critical section. For an arbitrary access $\alpha_i^\nu$, overhead must be included only if that access is the first of a critical section, *i.e.* $x_{i,\nu} = 0$.

$$\forall i \;\; C_i \geq \sum_\nu |\gamma_i^\nu| + \sum_\nu \left(|\alpha_i^\nu| + (1 - x_{i,\nu}) \cdot \mathcal{O}\right) \tag{9}$$

**Example 8.** *For task $\tau_i$ depicted in Fig. 1, with $\sum_\nu |\gamma_i^\nu| = (1 + 4 + 2 + 2) = 9$, this results in $C_i \geq 9 + \sum_\nu \left(|\alpha_i^\nu| + (1 - x_{i,\nu}) \cdot \mathcal{O}\right)$. For Fig. 1c, this yields $C_i \geq 9 + (2 + 1 \cdot 1) + (4 + 1 \cdot 1) + (3 + 0 \cdot 1) = 9 + 3 + 5 + 3 = 20$.*

### *B. Expanding to Multiple Resources without Nesting*

Now we expand upon the optimization problem formulation presented in Sec. III-A to handle multiple resources. To ease the development of an initial solution, we focus on a more restricted setting by not allowing any nesting. That is, when accesses are assigned to critical sections, accesses to different resources may not be combined, as that would cause nested access and necessitate additional accounting, explored in more detail in Sec. III-C.

We present our formulation of the multi-resource problem when nested access is not allowed, beginning with basic schedulability constraints. Equations 1–3 can be applied to this problem without modification; each task must still meet its deadlines, account for the summation of higher priority work, and apply the constraint of response time analysis.

**Incorporating blocking.** To constrain the blocking $B_i$ possibly experienced by task $\tau_i$, we introduce the variable $L_{j,k}^{\max}$ to denote the longest critical section of task $\tau_j$ for resource $\mathcal{L}_k$ and enforce that $B_i$ must be at least this duration for any critical section that could cause blocking under the PCP. More specifically, we consider all lower-priority tasks and their usage of any resource $\mathcal{L}_k$ with a priority ceiling $\text{PC}(k)$ at least that of the priority of task $\tau_i$.

$$\forall k, \forall i \geq \text{PC}(k), \forall j > i \;\; B_i \geq L_{j,k}^{\max} \tag{10}$$

**Grouping access into critical sections.** We again utilize a decision variable $x_{i,\nu}$ to indicate whether the $\nu^{\text{th}}$ access is combined with the prior access. Here, too, we enforce that the first access cannot be combined with a prior access by applying Eq. (5).

To enforce that no critical sections may be nested, we disallow a resource access from being combined into a critical section with another access to a different resource. For this, we define a function *res*() that maps an access to its associated resource.

$$\forall i, \forall \nu > 1, \textit{res}(\alpha_i^\nu) \neq \textit{res}(\alpha_i^{\nu-1}) \;\; x_{i,\nu} \leq 0 \tag{11}$$

We again account for the duration of a critical section through the $\nu^{th}$ access with the variable $y_{i,\nu}$. Because the overhead may differ per resource, we modify Eq. (6) to instead account for the overhead of the resource in use ($\mathcal{O}_{res(\alpha_i^0)}$).

$$\forall i, |A_i| \neq 0 \quad y_{i,1} \geq |\alpha_i^1| + \mathcal{O}_{res(\alpha_i^0)} \tag{12}$$

Similarly, the constraint on duration for later accesses also is updated to include the resource-specific overhead.

$$\forall i, |A_i| \neq 0, \forall \nu > 1$$
$$y_{i,\nu} \geq x_{i,\nu} \cdot \left(y_{i,\nu-1} + |\gamma_i^{\nu-1}|\right) + |\alpha_i^\nu| + (1 - x_{i,\nu}) \cdot \mathcal{O}_{res(\alpha_i^\nu)} \tag{13}$$

**Bounding critical-section and job durations.** In contrast to the single resource problem, we now account for the maximum critical-section duration on a per-resource basis. Thus, we update Eq. (8) to enforce a constraint only on the relevant duration, $L^{\max}_{i,res(\alpha_i^\nu)}$, based on access $\alpha_i^\nu$.

$$\forall i, \forall \nu \quad L^{\max}_{i,res(\alpha_i^\nu)} \geq y_{i,\nu} \tag{14}$$

Computing the WCET is similar to our single-resource approach (Eq. (9)), here accounting for the resource-specific overhead that must be incurred if a given access is not combined with a prior access ($x_{i,\nu} = 0$).

$$\forall i \quad C_i \geq \sum_\nu |\gamma_i^\nu| + \sum_\nu \left(|\alpha_i^\nu| + (1 - x_{i,\nu}) \cdot \mathcal{O}_{res(\alpha_i^\nu)}\right) \tag{15}$$

### C. The Challenge of Nested Resource Access

As part of supporting multiple shared resources, the PCP handles properly nested critical sections. However, the assignment of accesses to critical sections is complicated if critical sections may be nested.

**Example 9.** *Consider the task depicted in Fig. 3a. If all of these accesses were combined into critical sections starting and ending with accesses for the same resource, it would result in improperly nested critical sections, as depicted in Fig. 3b. This could be remedied by requiring the task to continue to hold Resource A until the completion of the last access of Resource B, effectively resulting in an outer critical section for A and an inner nested critical section for B, as depicted in Fig. 3c. However, doing so unnecessarily increases the duration of the critical section for A, but saves incurring additional overhead for B in contrast to the critical sections shown in Fig. 3d.*

The above example raises two concerns: (1) how to account for critical-section durations, and (2) how to ensure properly nested critical sections. For considering critical-section durations, the critical section of each resource should be computed separately, so that the blocking impact on higher priority tasks can be modeled more precisely. This, however, leads to challenges with the current approach; the critical-section duration cannot be calculated in the recursive manner previously described.

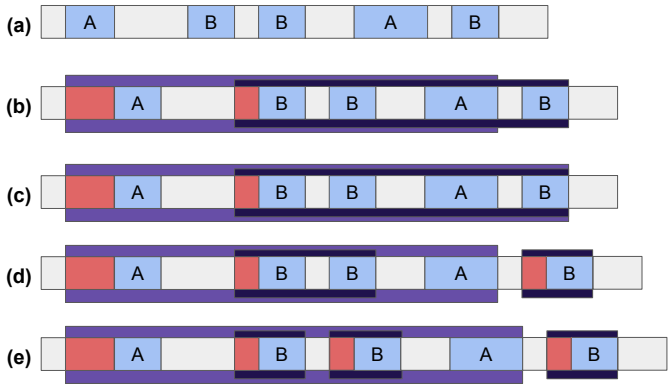


Fig. 3. An example task with (a) access and non-access segments, (b) a disallowed assignment of critical sections that results in improper nesting, and (c-e) multiple allowed assignments of critical sections.

**Example 10.** *Consider Fig. 3c. If using the recursive approach described previously to track critical section duration, a decision variable for the second access of the task (for Resource B) would need to account for both the start of a critical section for Resource B and a possible continuation for Resource A. However, if the fourth access (also for Resource A) ultimately is not part of the same critical section, then a decision variable at the second access should indicate to not include further durations in the first critical section.*

To enforce properly nested critical sections, it is necessary to keep track of those which are ongoing.

**Example 11.** *In Fig. 3c, it is necessary to keep track of the outer critical section for Resource A in order to determine that Resource A must be held until the release of Resource B.*

The above example illustrates another scenario in which tracking critical-sections duration may have additional challenges when nesting is allowed; a critical section for a given resource may have an extended duration beyond any access to that resource.

In addition to tracking ongoing critical sections, there may be multiple separate critical sections for the same resource nested in an outer critical section.

**Example 12.** *In contrast to Fig. 3d which shows one outer and one inner critical section, Fig. 3e depicts two separate inner critical sections for Resource B, each nested within an outer critical section for Resource A. This type of allocation may be necessary when higher priority tasks for Resource B have a more limited capacity to incur blocking than the tasks that require Resource A.*

The challenges discussed in this section must be answered if nested critical sections are to be allowed. We leave such an exploration to future work.

## IV. Evaluation

Specifying the lock granularity problem as an optimization problem enables finding an assignment of accesses to critical

sections if one exists. However, this can come at the cost of significant running time to find a solution. We therefore explore the size of the generated optimization problems to choose lock granularity for tasksets from a range of experimental setups to quantify the differences independent of optimization solver or platform. Then, we look at the runtime of a few specific examples.

**Problem size.** The size of an optimization problem can be measured in the number of variables and constraints. The number of tasks in a taskset has a clear impact on both measures. For example, the number of response-time variables and constraints from Eq. (1) and Eq. (3) grow linearly in the number of tasks, while the number of accounting variables of the form $Z_{i,j}$ and constraints from Eq. (2) and Eq. (4) grow quadratically in the number of tasks. Other variables and constraints depend on the number of tasks and the number of resource accesses made by those tasks. This includes the variables $x_{i,\nu}$ and $y_{i,\nu}$ and constraints like those resulting from Eq. (7) and Eq. (8).

We look at a few different types of tasksets to illustrate the impact different parameters have on problem size, borrowing commonly used ranges [1], [6]. We consider tasksets randomly generated using different per-task utilizations, either light (0.001 – 0.1) or medium (0.1 – 0.4), up to a system utilization of 0.6, with a goal number of eight accesses per task and a single resource, for which tasks had a 0.6 probability of accessing the resource. (Tasks with resource accesses had time balanced between access and non-access segments, and if this was not possible within the parameters, the task did not access any resources.) Other parameters, like task periods and access durations, were also set, but these do not impact the number of variables or constraints.

We utilized the version 9.1.2 of the Gurobi optimization solver [10] to build and execute our single-resource optimization problem. For medium-utilization tasks, with an average of 3.2 tasks per task set, the averages across 10 randomly generated tasksets were 36.8 variables and 26.9 and 10.9 linear and quadratic constraints, respectively (as reported by Gurobi's model attributes `NumConstrs` and `NumQConstrs`). For light-utilization tasks there were an average of 13.2 tasks across the 10 task sets; this resulted in an average of 194.8 variables, as well as 229.1 and 35.9 linear and quadratic constraints, respectively.

**Runtime.** We formulated the single-resource optimization problem in Python and ran the solver on a shared server, with dedicated access to 8 multi-threaded cores and 16GB of memory on a dual-socket machine equipped with two 2.30GHz Intel Xeon Gold 5218 CPUs. We used the same experimental setup as describe above, which included both medium- and light-utilization tasks with short periods (3–33 ms) and GPU-inspired accesses with durations of 10–200 $\mu$s. We assigned task priorities in Deadline Monotonic [3] order.

Solving 10 randomly generated task systems with these parameters resulted in an average runtime (measured as wall-clock time during the Gurobi solving step) of 15.8 ms for medium-utilization tasksets and 139.8 ms for light-utilization tasksets. Filtering by feasibility, we found that feasible optimization problems took on average 51.6 and 174.5 ms for medium- and light-utilization tasks, respectively; for infeasible tasksets, the runtimes were respectively only 0.4 and 1.1 ms.

**Generalizing to multiple resources.** The results above are for the single-resource case. For multiple resources, the number of constraints generated by some of the inequalities remains the same. For example, Eq. (13) results in the same number of constraints, simply using different overhead values per resource. Two inequalities differ from the problem for a single resource. The number of constraints generated by Eq. (10) grows linearly in the number of resources in the system and quadratically in the number of tasks. Eq. (11) is unique to the multi-resource case, and the number of constraints it generates is based on the number of accesses for each task.

## V. Related Work

Prior work has considered the problem of grouping resource accesses into critical sections for systems with only a single shared resource [2]. This approach considered the PIP and presented an optimal greedy algorithm for determining the critical sections. Conceptually, a greedy approach worked for this simpler problem because each decision could be considered in isolation with a single goal: maximally combine accesses into a critical section with a duration not exceeding the blocking tolerance of higher-priority tasks. This work built on an approach for computing the allowable duration of non-preemptable regions in a task in the context of limited-preemption systems [12], [13]. However, for systems with multiple resources, the considerations expand to include different subsets of tasks that may incur blocking and different ways accesses can be combined (*e.g.*, creating nested critical sections).

While there is limited prior work on combining resource accesses into critical sections in a manner that ensures schedulability, other problems exhibit related characteristics. The segmented self-suspension task model [9] considers each task to comprise alternating segments of computation and self-suspension. Modeling self suspensions explicitly enables more accurate representation of I/O handling, computations offloaded to a hardware accelerator, etc. As mentioned above, work on non-preemptive regions (*e.g.*, [5], [7], [12], [13]) also shares some similarities; the non-preemptive execution of a task imposes delays on other tasks similar to the blocking that results from a resource access protocol. The mechanisms and the sets of impacted tasks may differ, but both problems must consider the tradeoffs within a task system as a whole.

## VI. Conclusion

In this paper, we express the critical-section granularity problem as an optimization problem. We present formulations for systems with a single shared resource and with multiple resources accessed in a non-nested manner, and we discuss the challenges that arise when generalizing this problem to multiple nested resource accesses. We additionally evaluate

our formulation for a single resource in terms of the size and runtime of the problem using the Gurobi solver.

In the future, we will expand our formulation to enable optimization problems with nested accesses. Additionally, we plan to continue our evaluation to include more variance in task parameters and task set sizes, as well as to evaluate task sets with both non-nested and nested shared-resource accesses. We also plan to explore the connections to related problems in more depth.

## REFERENCES


[1] SchedCAT: Schedulability test collection and toolkit. http://www.mpi-sws.org/\~bbb/projects/schedcat.

[2] Tanya Amert and Catherine E Nemitz. Taking one for the team: Trading overhead and blocking for optimal critical-section granularity with a shared gpu. In *Proceedings of the 32nd International Conference on Real-Time Networks and Systems*, 2024.

[3] Neil C Audsley, Alan Burns, Mike F Richardson, and Andy J Wellings. Hard real-time scheduling: The deadline-monotonic approach. *IFAC Proceedings Volumes*, 24(2):127–132, 1991.

[4] Sanjoy Baruah and Pontus Ekberg. An ilp representation of response time analysis. Technical report, Technical Report., 2021. URL: https://research.engineering.wustl.edu/~baruah/Submitted/2021-ILP-RTA.pdf.

[5] Marko Bertogna, Orges Xhani, Mauro Marinoni, Francesco Esposito, and Giorgio Buttazzo. Optimal selection of preemption points to minimize preemption overhead. In *Proceedings of the 23rd Euromicro Conference on Real-Time Systems*, pages 217–227, 2011.

[6] Björn B Brandenburg. *Scheduling and Locking in Multiprocessor Real-time Operating Systems*. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 2011.

[7] Reinder J Bril, Johan J Lukkien, and Wim FJ Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Systems*, 42:63–119, 2009.

[8] Giorgio C Buttazzo. *Hard real-time computing systems: Predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[9] Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, and Cong Liu. State of the art for scheduling and analyzing self-suspending sporadic real-time tasks. In *Proceedings of the 23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–10. IEEE, 2017.

[10] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL: https://www.gurobi.com.

[11] Lui Sha, Ragunathan Rajkumar, and John P Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[12] Gang Yao, Giorgio Buttazzo, and Marko Bertogna. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 351–360. IEEE, 2009.

[13] Gang Yao, Giorgio Buttazzo, and Marko Bertogna. Feasibility analysis under fixed priority scheduling with limited preemptions. *Real-Time Systems*, 47(3):198–223, 2011.