CHALMERS
UNIVERSITY OF TECHNOLOGY

Global Scheduling of Real-Time Tasks on Multiprocessors based on Priority Promotion

Master's thesis in Embedded Electronic System Design

TIANQI WEN

Department of Microtechnology and Nanoscience
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025

Master's thesis 2025

Global Scheduling of Real-Time Tasks on Multiprocessors based on Priority Promotion

TIANQI WEN

Department of Microtechnology and Nanoscience
Chalmers University of Technology
Gothenburg, Sweden 2025

Global Scheduling of Real-Time Tasks on Multiprocessors based on Priority Promotion
TIANQI WEN

Supervisor: Risat Pathan, Department of Computer Science and Engineering
Examiner: Per Larsson-Edefors, Department of Microtechnology and Nanoscience

Master's Thesis 2025
Department of Microtechnology and Nanoscience
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Global Scheduling of Real-Time Tasks on Multiprocessors based on Priority Promotion

TIANQI WEN
Department of Microtechnology and Nanoscience
Chalmers University of Technology

Abstract

With the widespread deployment of intelligent embedded systems across various application scenarios, the processing capabilities and scheduling strategies of single-processor platforms are increasingly unable to meet the growing computational demands of real-time tasks. To host more and more functions with additional processing requirements and stricter real-time constraints, multiprocessor platforms are attracting growing attention and adoption. However, scheduling real-time tasks on multiprocessor platforms still faces many challenges.

One major challenge is how to assign priorities in an optimal way for scheduling on multiprocessors. There are many methods for assigning fixed priorities to tasks in multiprocessor systems, but the optimal method is still unclear, and some tasks cannot be scheduled feasibly under fixed-priority assignments. To overcome the limitations of fixed-priority scheduling, it becomes necessary to introduce some form of dynamic priority. Although the schedulers under dynamic-priority may perform better in terms of schedulability, they are significantly more complex to implement than the schedulers under fixed-priority. This leads to another challenge: how to make a compromise between static and dynamic priority behaviour?

Dual-priority (DP) scheduling provides a new approach to balance between pure fixed-priority and full dynamic-priority scheduling strategies. In dual-priority scheduling, each task is assigned two priorities: an initial priority and a promoted priority. The task begins execution with the initial priority, and after a fixed duration, which is called priority promotion time, its priority is elevated to the promoted priority. However, to the best of our knowledge, there has been no prior research on applying DP scheduling to multiprocessor platforms. This thesis focuses on the study of global scheduling with priority promotion in multiprocessor systems and proposes a DP scheduling algorithm for multiprocessors, along with two tests to verify schedulability.

Furthermore, with a priority assignment algorithm known as Audsley's optimal priority assignment, we develop a hybrid algorithm where a subset of the tasks will have pure fixed priority, and each of the other tasks will have two priorities: one before the priority promotion point(PPP) and one after the PPP. We have evaluated the proposed test using extensive simulation experiments, and the results demonstrate that the proposed test performs well compared to the existing test.

Keywords: Global Scheduling, Dual Priority, Multi Processors, Schedulability Test, OPA Algorithm.

Acknowledgements

We would like to express our gratitude to professor Risat Pathan as our supervisor for his valuable and constructive feedback during the whole master's thesis and for his patient guidance.
We would also like to thank the professor Per Larsson-Edefors as our examiner for their useful advice and patient guidance on writing the thesis.
Finally, we would like to thank Chalmers University for providing all the necessary resources to carry out the thesis.

Tianqi Wen, Gothenburg, July 2025

Contents

List of Figures

List of Tables

1

Introduction

With the widespread deployment of intelligent embedded systems across various application scenarios, the processing capabilities and scheduling strategies of uniprocessor platforms are increasingly unable to meet the growing demands of real-time tasks. To address higher computational loads and stricter real-time constraints, multiprocessor platforms are attracting growing attention and adoption. However, effectively allocating and scheduling real-time tasks in multiprocessor systems remains a complex and challenging research focus [1].

Existing studies have shown that scheduling real-time tasks on multiprocessor platforms remains a fundamentally challenging problem due to the complexity of coordinating task execution across multiple processors while satisfying strict timing constraints. Unlike uniprocessor systems, where optimal algorithms such as earliest deadline first (EDF) can guarantee schedulability under certain conditions, there exists no universally optimal scheduling algorithm for multiprocessor systems. The lack of such an optimal solution is due to several factors: the compromise between the different strategies, the need to handle both preemptions and migrations efficiently, and the combinatorial explosion of scheduling possibilities. Finding globally optimal real-time scheduling solutions for multiprocessor systems under general conditions is computationally difficult, with many related problems proven to be NP-hard [2]. Consequently, from both theoretical and practical perspectives, the pursuit of efficient and reliable real-time scheduling strategies remains a primary goal for researchers. Simultaneously, the development of rigorous and trustworthy schedulability analysis methods to determine whether a given task set can meet its deadlines under specific multiprocessor scheduling strategies is a key research direction.

In terms of global scheduling, especially global fixed-priority scheduling, such as global rate monotonic (RM), the well-known Dhall's effect can lead to unschedulability even when the task utilization (i.e., computational load) of a task set is not very high [3]. A key problem is that tasks' periods/deadlines and execution times may have a complex relationship, which makes it difficult for all tasks to meet their deadlines. However, in critical systems such as the braking system of cars and the scheduling system in aircraft, meeting all task deadlines is crucial. Currently, assigning priority to the tasks and determining their feasibility are both challenging for global scheduling. The works in [4] demonstrated that the optimal priority assignment (OPA) algorithm in [5] can also be applied to determine the fixed-priority ordering of the tasks scheduled using a global approach on multiprocessors when

the schedulability test meets three specific conditions. Based on these three conditions, a schedulability test can be OPA-compatible or OPA-incompatible. For OPA-incompatible schedulability tests, the test cannot be applied with OPA to find a schedulable priority assignment, but if a priority assignment is given, we can use these tests to check if all the deadlines will be met or not. For OPA-compatible schedulability tests, not only can they determine whether a given priority assignment makes the task set schedulable, but they can also find the optimal or feasible priority assignment that makes the task set schedulable.

In the current research, OPA-compatible global schedulability tests are the state-of-the-art tests and the most advanced tools [1]. But when the test applied to the OPA is a "poor" or imprecise test, such as a test that may always signal some task set as unschedulable, whereas the task set in fact is schedulable. We cannot use these imprecise tests to find a good schedulable priority assignment. To that end, there are feasible task sets that may not be deemed schedulable using any of the state-of-the-art tests. More noteworthy is that [4] also shows that any exact schedulability test for general periodic task sets is incompatible with OPA. Thus, the OPA algorithm with any OPA-compatible test we know so far is suboptimal. This means there may be some task sets that are schedulable using global fixed priority scheduling, but existing OPA-compatible tests cannot find the priority assignment. In this case, introducing dynamic priority can help determine whether the task set is schedulable or not. And the work on [6], [7] and the example shown in section 2.3.2 shows that the minimal dynamic priority behavior approach based on the priority promotion strategy could be a viable alternative to schedule real-time tasks, as is explained in next paragraph.

In fixed-priority global scheduling, all arriving tasks wait in a global waiting queue, and the scheduler always assigns the highest-priority task to each processor. A task with a relatively higher priority is never preempted by a relatively lower priority task in fixed-priority global scheduling. Nevertheless, with the priority promotion mechanism, each task will have one or more promotion points (PPP), which are the fixed offset relative to the arrival time of the task. Then, when the promotion point comes, the task in execution or in the ready queue will get a promoted priority. What's more, introducing priority promotion ensures that a task's priority over other tasks can change only at the promotion points, which are related to the minimal dynamic priority behavior approach by dual priority scheme in [7]. In this study, each task has at most one PPP, which essentially means a task can have two fixed priority levels, one before the PPP and one after the PPP. Since the number of these PPPs is bounded, the mechanism is easy to implement and thus more practical for the industry. Therefore, whether there exist some task sets that could achieve better performance under a global scheduling algorithm with priority promotion still requires further investigation.

1.1 Contribution

This master's thesis focuses on the study of global scheduling with priority promotion in multiprocessor systems. To apply the priority promotion mechanism within global

scheduling, several key issues must be addressed:

1. Given a task set and a set of PPPs, how can the schedulability be analyzed?
2. How should PPP be assigned?
3. How much is the performance improvement after applying the priority promotion mechanism?

For the first problem, we need a schedulability test to evaluate whether a given task set is schedulable. With the priority promotion mechanism, each task is assigned two fixed priorities. Compared to pure fixed-priority scheduling, the pattern of preemption among tasks becomes significantly more complex due to the priority change during the execution. Thus, deriving the schedulability test requires considering various scenarios and substantial mathematical reasoning.

Regarding the second problem, because task priority changes at PPP, the task preemption patterns before and after these points can be totally different. The assignment of PPPs significantly affects preemption between tasks. Additionally, what priority is before and after the PPP will influence the changes in task priority at PPP. Therefore, we need a strategy to assign the PPP, and it is necessary to carefully consider each task's priority both before and after the PPP.

For the third problem, we need to establish a method for comparing our proposed approach with existing theories and methods. To ensure the reliability of results, extensive simulations using numerous task sets across varying numbers of processors and tasks are required. Consequently, we will need to implement a simulator to evaluate the effectiveness of our proposed approach.

More specifically, this thesis provides the following contributions:

- We propose a dual-priority (DP) scheduling algorithm for a set of periodic tasks on the multiprocessor platform.
- We propose a schedulability test for the DP scheduling algorithm: deadline analysis with dual priority scheduling (DA-DP), is proposed to evaluate whether a given task set, with assigned priority promotion points, can be successfully scheduled.
- Based on the DA-DP and the existing test for global fixed-priority (GFP) scheduling, we propose a hybrid priority assignment algorithm using OPA where a subset of the tasks are assigned a lower fixed priority without any priority promotion, and the rest of the tasks are assigned dual priorities. This priority assignment introduces dual priority to a subset of the tasks only when needed; otherwise, the task set can still be scheduled using GFP scheduling.
- There are numerous ways to assign promotion points, and finding the best one is very difficult since some PPP assignments are effective for one task set while not as effective for another task set. This thesis also proposes different heuristics to assign PPP for arbitrary task sets and shows the effectiveness of these heuristics.
- To evaluate the performance of our proposed test and algorithm, we develop both the algorithms and the schedulability tests in a simulator and compare them with those of the state-of-the-art approach.

1.2 Thesis Outline

The structure of this thesis is as follows:

- Chapter 2 provides an overview of real-time scheduling theory and essential background information for understanding global scheduling with priority promotion. It also reviews related works on priority promotion mechanisms.
- Chapter 3 derives a schedulability test for global scheduling with priority promotion and summarizes the proposed schedulability test in formula form.
- Chapter 4 proposes a hybrid algorithm to assign the priority of the task and also discusses the different priority promotion point assignment strategies.
- Chapter 5 details the implementation of the simulation framework, including task set generation and two simulations for evaluating the schedulability test and different priority promotion point assignment strategies.
- Chapter 6 shows the simulation results, evaluates the effectiveness of the proposed schedulability test, and provides an analysis of the results.
- Chapter 7 discusses areas where this thesis is worth expanding and potential improvements.
- Chapter 8 concludes the thesis work.

2

Technical Background

This chapter presents the required technical background to understand the problem that this master's thesis will focus on.

2.1 Real-time Systems

Real-time systems are computing systems designed to process data and deliver outputs within strictly defined timing constraints or deadlines. Unlike general-purpose systems, the correctness of real-time systems depends not only on logic accuracy but also on timely execution. Missing timing constraints can lead to system failures.

In a real-time system, a real-time application like control and monitoring may be designed as a collection of periodic tasks, and each task τ_i is an independent unit of execution that needs to be completed periodically.

The model of the basic real-time task is shown in Figure 2.1. For each task τ_i, there are parameters such as the execution time C_i, deadline D_i, and period T_i. Each execution of such a periodic task is referred to as a job. And the upward arrow in Figure 2.1 is used to represent the release of the jobs, the downward arrow is used to represent the absolute deadline of the jobs. The k^{th} job of the task arrives at a specific release time r_i^k. The interval between consecutive jobs' release times is the period T_i. The time that is needed to finish the job is the execution time C_i, and the task needs to finish its execution time before the deadline D_i.

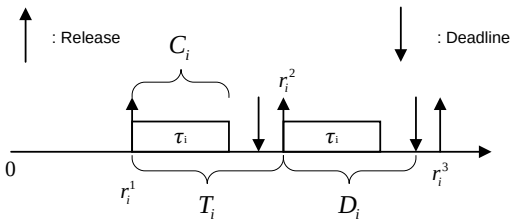

Figure 2.1: Real-time task model

Moreover, tasks in real-time systems sometimes have additional parameters: execution time, deadline, and priority, where the latter can represent the order of

execution of tasks when more than one task is ready for execution and can influence the running of the task beyond the basic period.

We will discuss more details about the priority in section 2.3. Specifically, certain specialized scheduling algorithms may include other parameters like priority promotion point P_i, which is a variable we will use for our research. P_i is not necessary for the task model, and we will discuss P_i in detail in section 2.3.2. The utilization of a task τ_i is defined as $\frac{C_i}{T_i}$, which represents the computational load of the task. Then the sum of utilization for each task: $\frac{C_1}{T_1} + \frac{C_2}{T_2} + ... + \frac{C_n}{T_n}$, is the load of the task set or the application. For a single processor, it is necessary that the utilization load is less than 1 to meet the deadlines.

Furthermore, when a task's deadline is equal to its period, it is called an implicit-deadline task; otherwise, it is known as a constrained-deadline task. In this thesis, we consider a set of implicit-deadline periodic tasks.

2.2 Global Scheduling

In multiprocessor scheduling, two main approaches are commonly discussed: global scheduling and partitioned scheduling.

Under partitioned scheduling, which is shown in Figure 2.2, each processor has its own local ready queue. Once tasks are allocated to a specific processor based on a partitioning strategy such as first fit, they can only execute on that processor. Consequently, the scheduling on each processor can be treated as a separate uniprocessor scheduling problem [1]. Since there are no task migrations between processors in partitioned scheduling, it is simple to implement. However, it lacks flexibility and makes it difficult to achieve balanced workloads across processors in the sense that a task may need to wait in its assigned local ready queue even if some other processor is idle.

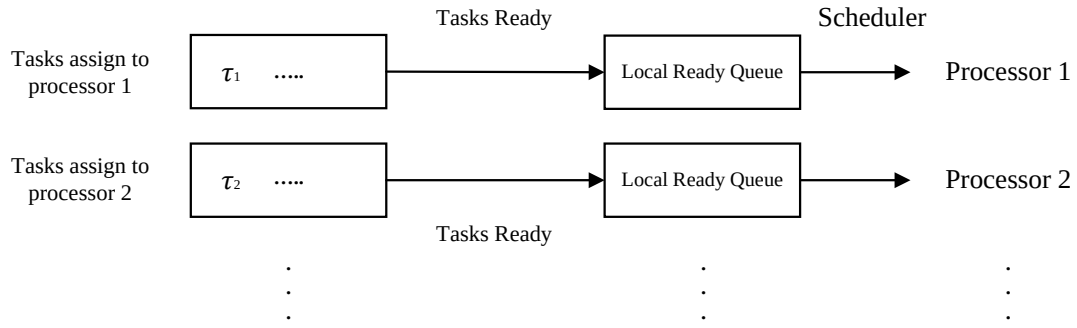

Figure 2.2: Partitioned scheduling

Under global scheduling, which is shown in Figure 2.3, there is only one global ready queue. When the tasks are ready, they will be stored in the same global ready

queue, and when a new task needs to be scheduled, the scheduler selects the task by an algorithm from this global queue and assigns it either to an idle processor or preempts a relatively lower priority task that is currently executing on a processor. Although this approach helps avoid overloading a single processor, it also introduces several challenges. As mentioned in [1], Dhall's effect in global scheduling can lead to unschedulability even when the task utilization or load is high.

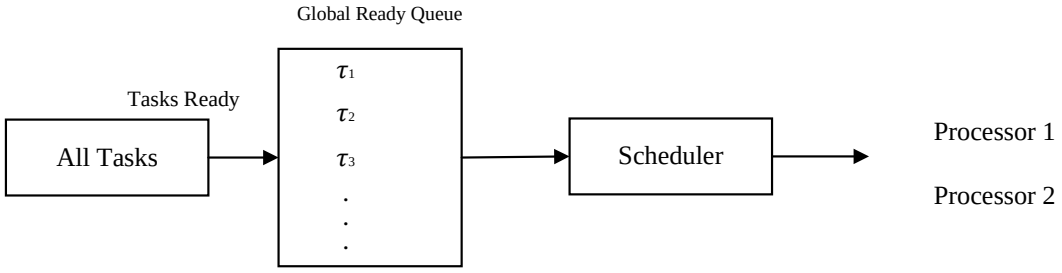

Figure 2.3: Global scheduling

In this thesis, we will focus on global scheduling, and in the following subsection, we will discuss the global scheduling algorithm that the scheduler uses to select a task from the global ready queue and assign it to a processor.

2.2.1 Global Scheduling Algorithm

The global scheduling algorithm is an algorithm that the scheduler uses to decide which task should be assigned to the processor when the task is ready to run. Examples of widely studied global scheduling algorithms include global earliest deadline first (GEDF) and global fixed priority (GFP).

GFP assigns fixed priorities to tasks regardless of runtime conditions. These fixed priorities are determined before execution and are maintained consistently throughout runtime. This static assignment ensures simplified implementation, but sometimes the task with higher priority keeps running in the processor will leading to the starvation of tasks with lower priority. What's more, the priority assignment is a big challenge in GFP scheduling for the multiprocessor system. The optimal priority assignment strategy for the uniprocessor system is known, but there is no optimal one known for the multiprocessor system. There are many strategies to assign priority to tasks in global scheduling, such as the TkC priority assignment strategy, which can be used to circumvent Dhall's effect [8].

The GEDF assigns priorities based on the earliest absolute deadlines. Since tasks have different deadlines and are released periodically, some tasks may have the nearest absolute deadline at the beginning, but as the run-time changes, the task has the nearest absolute deadline will change too. So the priority order will change at run-time when applying the GEDF algorithm, which means a highly dynamic priority behavior. This dynamic priority behavior can solve the starvation that happened when applying the GFP, but the pattern of preemptions may be different

and may even increase for some task set in comparison to the pure GFP scheduling. When switching the task from one processor to another, systems need to spend time and resources on saving/restoring the task states and other things, resulting the high overhead. Maintaining the dynamic priority scheduling is more complex than implementing fixed priority scheduling, which may incur more overhead.

The challenge arises when attempting to compromise between these two algorithms. Fully dynamic priorities, like those in GEDF, are effective but at the cost of increased scheduling overhead and complexity. Conversely, fully static priorities, as in GFP, are simple to implement and reduce overhead, but can significantly reduce efficiency. The priority promotion mechanism we will discuss in section 2.3.2 is an effective way to balance between the fully dynamic priorities and the pure fixed priorities. To the best of our knowledge, there is no study that applies the priority promotion in the global scheduling, so it is worth trying and find the potential of the priority promotion in the multi-processor system.

2.3 Task Priority

Task priority refers to a level assigned to each task, which determines the order in which tasks are selected for execution. In preemptive scheduling, higher-priority tasks can interrupt and preempt the execution of lower-priority tasks, thereby gaining earlier access to processor resources. The assignment of priorities decides the sequence in which tasks access computational resources, thereby deciding which task can execute first, and such an assignment is important in ensuring whether all the deadlines are met or not.

2.3.1 Priority Assignment Strategy

The priority assignment strategy can be categorized into fixed and dynamic strategies. In fixed-priority scheduling, each task is assigned a predetermined static priority, typically based on timing parameters such as deadlines or periods. For example, rate monotonic (RM) scheduling assigns priorities based on task periods. It will assign a higher priority to the task with a shorter period. In contrast, the dynamic priority assignment strategy allows task priorities to change at runtime based on the task's parameters. A classic example is earliest deadline first (EDF), which assigns priorities dynamically according to the tasks' absolute deadlines, adapting to changes during system execution.

The priority assignment strategy has a significant impact on system schedulability. If a task with high utilization, i.e., heavy load, is assigned a lower priority, the task is likely to miss its deadline. How to find a good strategy to assign the priority is still a challenge. For a task set containing n tasks, there are $n!$ possible priority assignments, most of which may not result in a schedulable system if the utilization or load of the task set is relatively high. Under a specific scheduling algorithm, only a small fraction of these assignments typically lead to a feasible schedule. To address this, [1] summarizes the optimal priority assignment (OPA) algorithm proposed by Audsley [9], which systematically determines an optimal static priority ordering for

a given schedulability test. A schedulability test is a test that can be used to test whether the task set is schedulable or not under a given scheduling algorithm on a particular processing platform.

The OPA algorithm is shown in Algorithm 1. It starts from the lowest available priority and attempts to assign it to one of the tasks that haven't been assigned a priority. A schedulability test A is then applied, and assuming all the other tasks that have not yet been assigned any priorities will have higher priorities. If the test is passed, the algorithm proceeds to assign the next priority level to another task. If the test fails, the algorithm tries another task for that priority. This process continues iteratively until all tasks have been successfully assigned priorities, in which case the task set is deemed schedulable. If none of the tasks can be assigned a priority at a particular level, then the task set is deemed to be not schedulable. The run-time complexity of the tests is N^2 times the time complexity of the test. For example, if the test can be solved in polynomial time $O(1)$, then the overall algorithm can run in polynomial time $O(N^2)$, which can be completed within a limited time.

Algorithm 1 :OPA

```
1: N = Number of Task
2: for each priority n from level N to level 1, lowest first do
3:     for each unassigned task τ_i do
4:         if τ_i is schedulable at priority level n according to schedulability test A
5:             with all unassigned tasks assumed to have higher priorities then
6:             Assign τ to priority n
7:             break (back to line 2, continue a new loop with priority level n-1)
8:         end if
9:         if none of the tasks can be assigned at priority n  then
10:            return unschedulable
11:        end if
12:    end for
13: end for
14: return schedulable
```

However, most existing schedulability tests are conservative in nature, which means the test will have some pessimism, and some schedulable task sets will be recognized as unschedulable. We will discuss the schedulability test in detail in the next section. Then, if the test is overly pessimistic, the OPA algorithm may fail to identify any valid priority assignment, leading to potentially schedulable task sets being pessimistically recognized as unschedulable.

2.3.2 Priority Promotion

Under the priority promotion mechanism, each task has one or more promotion points. When a promotion point is reached, the priority of the task either in the running or in the ready queue is promoted. In the example shown in Figure 2.4, we consider four periodic tasks executing on two processors under GFP scheduling. We

assume that each task is assigned an initial priority level, for which a lower number indicates a higher priority.

In the example of Figure 2.4, we use the model of the basic real-time task with parameters such as the execution time C_i, deadline D_i, and period T_i that were introduced in the previous section 2.1. The value of C_i and priority are shown in the top left corner of Figure 2.4. We then use the upward arrow to represent the release of the jobs and the downward arrow to represent the absolute deadline of the jobs. Then the execution of the task is drawn as the rectangular block, and the red block shows the execution that occurs after the deadline. So this example shown in Figure 2.4 is unschedulable under the GFP scheduling.

We then apply the PPP and try to apply the priority promotion mechanism for the same example. We assume that each task is promoted by one priority level at each promotion point until it reaches the highest level. As shown in Figure 2.5, we use the dotted upward arrow to show the PPP.

It can be observed that at time $t = 8$, if τ_3 had not been promoted to priority level 1 via priority promotion, it would have been preempted by τ_2 and subsequently missed its deadline. A similar situation occurs at $t = 24$. At time $t = 28$, τ_4 is not preempted by τ_1 because τ_4 was promoted to the same priority level as τ_1 at $t = 27$. Consequently, when τ_1 is released, it cannot preempt a task with the same priority level, allowing τ_4 to meet its deadline just in time.

If we only apply GFP scheduling, multiple tasks would miss their deadlines, but the same task set is schedulable within the priority promotion mechanism. This demonstrates that introducing a dynamic behavior to the priority assignment through priority promotion can indeed offer improvements. The remaining question is how to determine the promotion points that enable tasks to meet their deadlines. A priority promotion policy or a heuristic algorithm is required to determine these promotion points. In the example shown in Figure 2.5, we use a strategy from [10] to assign the PPP that uses relative deadlines among tasks to decide when each task's priority should be promoted.

The study in [11] also proposes a heuristic called execution-based priority promotion (EPP) for task PPP assignment under uniprocessor FP scheduling, and further improves EPP by introducing an adjustment factor to control the speed of this heuristic. In [11], they also propose a schedulability test and experimentally evaluate the schedulability of systems applying EPP, demonstrating promising results on uniprocessor platforms. We will try EPP with our scheduling algorithm in the global multiprocessor scheduling setup and design other heuristics based on the EPP, which will be shown in section 5.1.1.

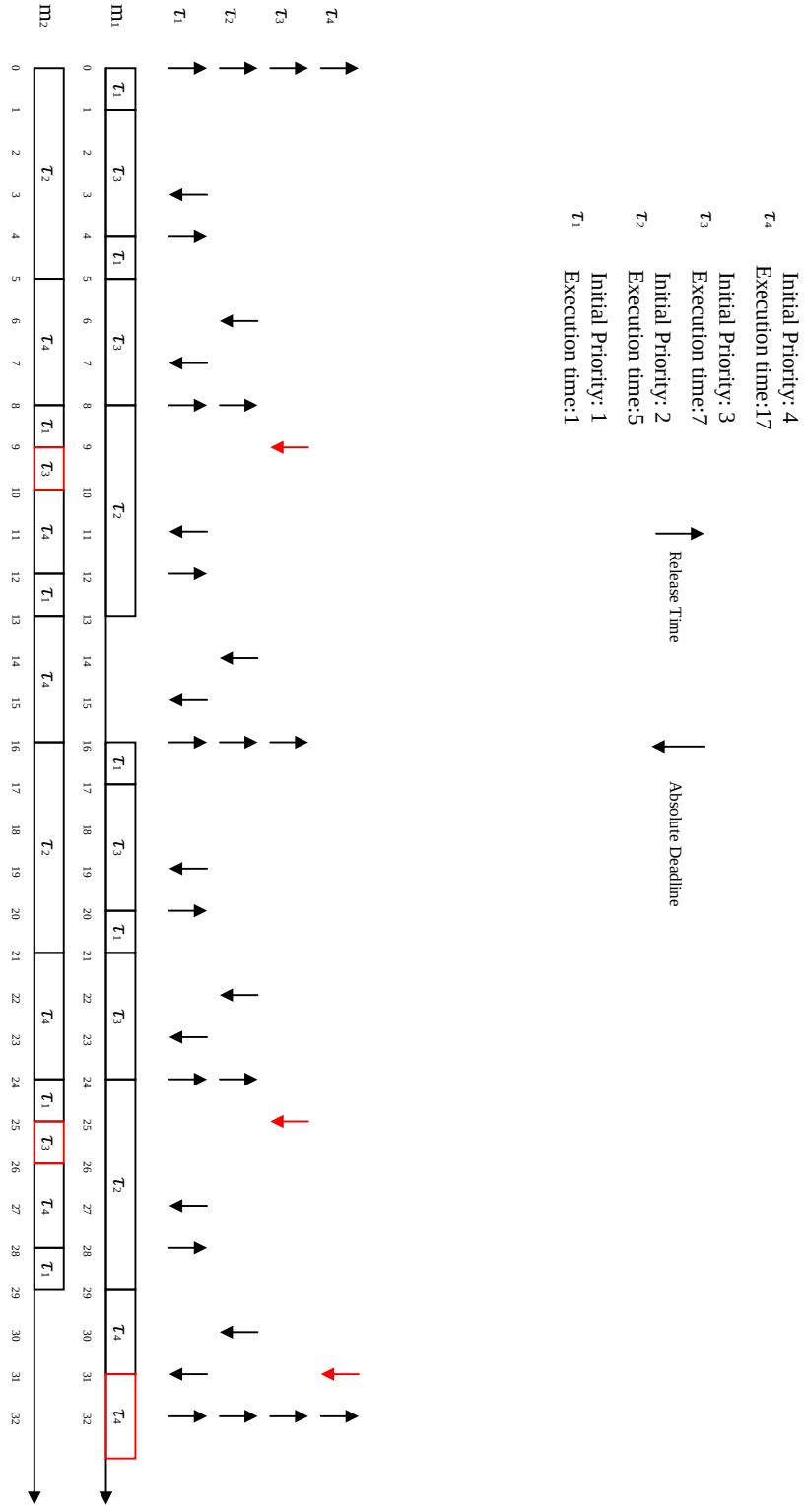

Figure 2.4: Example of FP with global scheduling

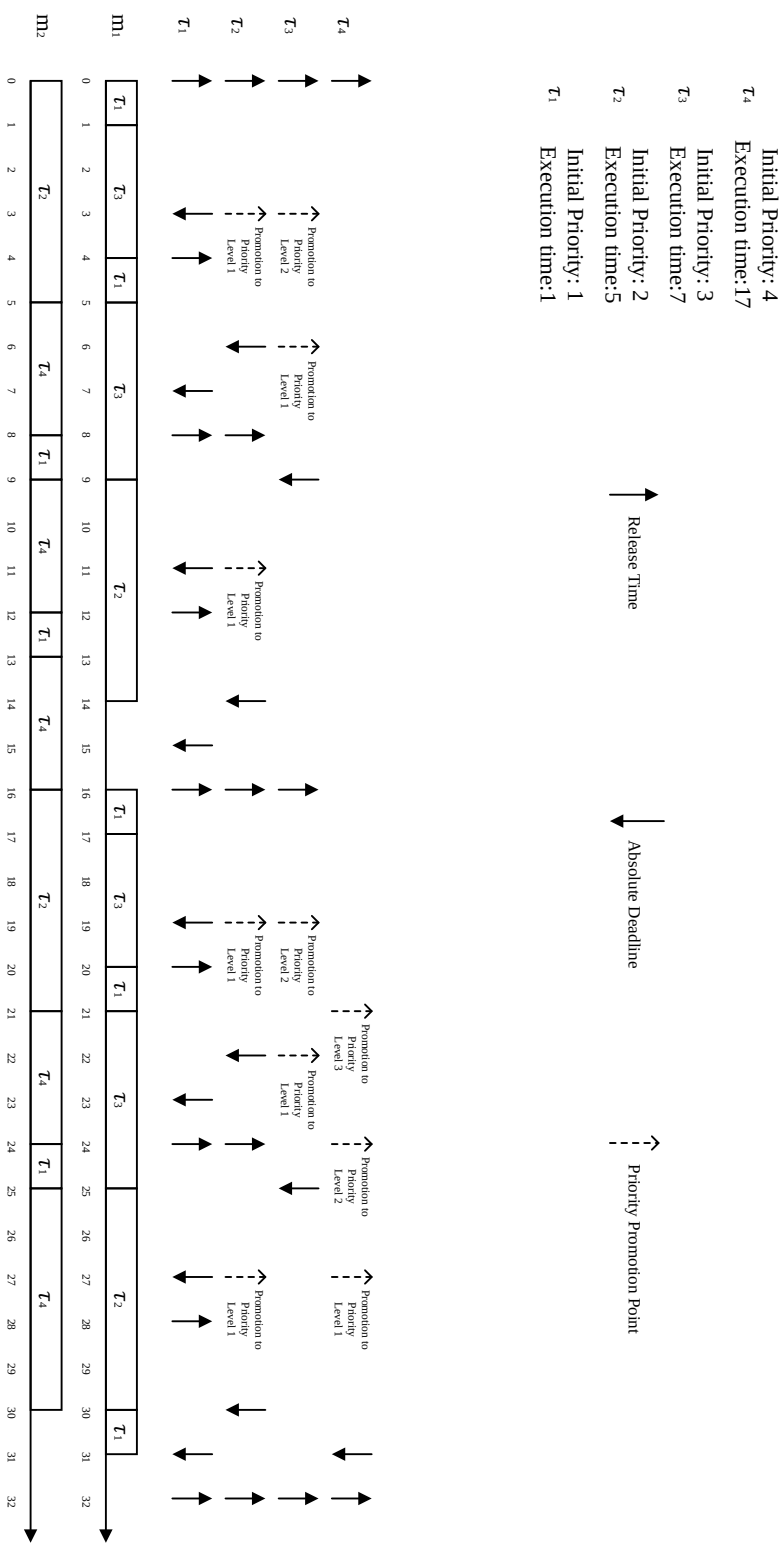

Figure 2.5: Example of priority promotion with global scheduling

2.4 Schedulability Test

In real-time systems, schedulability analysis is a method used to determine whether a given set of tasks can meet all their deadlines under specific constraints and scheduling policies. A widely used approach to construct schedulability tests is based on the fundamental strategy proposed by Baker [12]. The key steps are as follows:

1. Consider a time interval $[t_0, t]$ and assume that a task τ_i misses its deadline at the end of this busy interval.
2. Derive the necessary condition under which the task misses its deadline.
3. Derive an upper bound on the maximum interference caused by other tasks within this interval.
4. Formulate a necessary unschedulability test by combining the necessary condition from step 2 with the upper bound derived in step 3.
5. Negate the necessary unschedulability test to obtain a sufficient schedulability test.

As shown in step 5, we obtain a sufficient schedulability test by negating the necessary unschedulability test. Currently, all tests are sufficient for GFP scheduling because the workload or interference computation is pessimistic for global scheduling. This pessimism arises from the fact that neither priority assignment nor the critical instant is known for scheduling under a multiprocessor platform. A critical instant for τ_i is an arrival time that yields τ_i's worst-case response time, i.e., the longest delay between its release and completion that can occur while higher-priority tasks behave according to their specifications.

A sufficient test has pessimism and cannot guarantee the offline schedulability of all the possible schedulable task sets. Some schedulable task sets can be recognized as unschedulable due to pessimism; a task set that passes such a test is schedulable, but a task set that fails the test cannot determine its schedulability. Therefore, among a large number of theoretically schedulable task sets, the more schedulable task sets the schedulability test can find, the better the schedulability test is.

2.5 Related Work

In this section, we introduce some of the most important findings in real-time systems research, specifically focusing on multiprocessor scheduling and dual-priority scheduling.

2.5.1 Partitioned Scheduling

Partitioned scheduling has always been a popular paradigm due to its approach of statically assigning tasks to individual processors, enabling the extensive theory developed for single-processor scheduling to be leveraged on multiprocessor systems. However, early research indicated that assigning sporadic or periodic tasks to processors under partitioned scheduling is an NP-hard problem, even when each processor subsequently employs an optimal single-processor scheduling algorithm (e.g.,

EDF or fixed priority (FP) scheduling). Consequently, practical work has primarily concentrated on heuristic partitioned algorithms, such as the first-fit heuristic. For instance, the rate monotonic first-fit (RMFF) algorithm was introduced in [13], which applies the first-fit rule to rate-monotonic scheduling and uses a sufficient schedulability test proposed in [14] to test the feasibility of the task for each processor. [15] provided a utilization-based schedulability test for RMFF, establishing that RMFF guarantees schedulability if the total utilization of the task set does not exceed $M(2^{\frac{1}{2}} - 1)$, where M represents the number of processors. Nevertheless, this limit permits only about 41% total utilization, leading to under-utilization of processor resources. Subsequent research, such as [16], has applied different heuristic partitioned strategies to improve this utilization bound to 69.3% using an approach called task splitting, where a few tasks are allowed to migrate from one processor to another, and the rest of the tasks are never allowed to migrate.

This master's thesis will focus on investigating the potential of priority promotion mechanisms on multiprocessor platforms. Since the main research in partitioned scheduling revolves around heuristic packing rules, we will focus on the related work of global scheduling in the next subsection.

2.5.2 Global Scheduling

Under global scheduling, tasks can migrate between processors during execution, resulting in better average processor utilization but complicating schedulability analysis and implementation. Since each task may run on any processor, the rich theory from single-processor scheduling cannot be directly applied to global scheduling. Therefore, research in global scheduling primarily focuses on priority assignment strategies and schedulability tests specifically suitable for global scheduling.

Regarding priority assignment strategies, considerable research has been conducted on GFP scheduling. One common method is a straightforward heuristic priority assignment approach compatible with any schedulability test. Another approach employs Audsley's OPA algorithm [9], which can only handle schedulability tests meeting OPA's compatibility conditions. On a single-processor platform, FP scheduling schedulability tests rely on the concept of critical instants—moments when tasks experience their worst response times, typically occurring when all competing tasks simultaneously release their computation requests [14]. However, in global FP scheduling, these critical instants are generally unknown because simultaneous releases do not necessarily lead to the worst-case response time [17]. To effectively determine the schedulability when applying the global FP scheduling, the state-of-the-art analysis method is RTA-LC, as mentioned in [1]. Further improvements were introduced in [17], proposing the RTA-CE analysis method. Among various schedulability tests evaluated in [1], the combination of the OPA algorithm and the DA-LC schedulability test provided the best performance. Nonetheless, even the most effective DA-LC (OPA) approach fails to identify all schedulable task sets due to the limitations inherent in purely static priority strategies.

Dynamic priority strategies, however, potentially address this limitation. A response-time analysis (RTA) for global EDF (GEDF) scheduling was presented in [18], ex-

plicitly detailing its consideration of critical instants. The results from [18] demonstrated that EDF scheduling employing RTA generally outperforms all fixed-priority assignment methods. However, implementing GEDF in practice is complicated and requires consideration of the overhead caused by dynamic priority-induced task migrations.

Current research rarely addresses hybrid solutions between purely static and purely dynamic approaches on multiprocessor platforms. However, dual priority scheduling research on single-processor platforms has successfully achieved a balance between static and dynamic priorities with significant improvements, as discussed in the next subsection.

2.5.3 Dual Priority Scheduling

Dual priority (DP) scheduling, initially proposed by Burns et al. in [19], assigns each low background priority task a fixed idle-time stealing delay after which it is promoted to a higher priority. This priority promotion mechanism achieves nearly optimal responsiveness for aperiodic tasks while maintaining hard real-time guarantees for periodic tasks. Existing research on DP scheduling primarily focuses on single-processor platforms, investigating issues such as whether it can achieve the 100% utilization bound [20], how to determine appropriate priority promotion points, and developing suitable schedulability tests for DP scheduling [11]. Due to priority promotion, even on single-processor platforms, critical instants become uncertain, complicating schedulability analysis. Subsequent studies on DP scheduling, such as [11], introduced a PPP assignment strategy suitable for single processors and developed schedulability tests based on given priority promotion points. Pathan [21] further investigated priority promotion mechanisms for single processors by applying more than two PPPs. Nevertheless, current research lacks studies applying DP scheduling to multiprocessor platforms, primarily due to the increased complexity of preemptions and the challenges in determining critical instants necessary for deriving the schedulability test on a multiprocessor platform.

3

Schedulability Analysis

In this section, we propose a schedulability test for dual-priority scheduling, which is called deadline analysis with dual-priority scheduling (DA-DP). In the subsequent subsections, we present the system model, deadline analysis, workload calculation, improved calculation of workload in a more accurate way, and an overview of applying the DA-DP.

3.1 System Model

This section introduces the real-time system model used to derive our schedulability tests. We first show the task model, followed by the priority assignment model, and finally, we describe the scheduling model in our analysis.

3.1.1 Task Model

To derive our schedulability tests, we consider a task set τ consisting of N periodic tasks. Each task is a single unit of execution with basic parameters such as its execution time, deadline, and period, along with an additional parameter: the priority promotion time. The priority promotion time defines a specific PPP at which the task's priority is promoted, resulting in two different fixed priorities before and after this PPP.

Figure 3.1 illustrates the task model for the i^{th} task τ_i within the task set τ. Here, C_i represents the execution time required by each job of task τ_i, T_i indicates the period, i.e., the interval between consecutive job arrivals, and D_i denotes the relative deadline, which is the maximum allowed interval between a job's release and its completion. Additionally, P_i marks the priority promotion time, indicating the interval between the job's release and its PPP.

Furthermore, in Figure 3.1, the execution of each job is represented by rectangles, with upward arrows meaning job release times, downward arrows indicating absolute deadlines, and dotted arrows marking the PPP. We refer to the two different fixed priorities before and after the promotion point as the initial priority and the promoted priority, respectively.

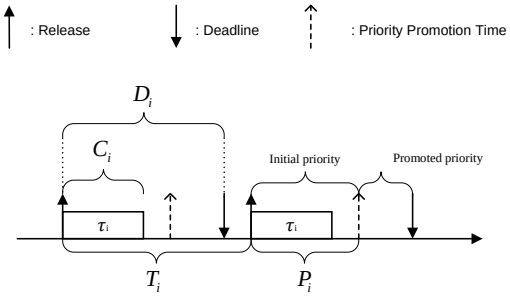

Figure 3.1: System task model

3.1.2 Priority Assignment Model

We utilize positive integers to represent task priorities, where smaller numerical values represent higher priorities. Each task in the task set τ is assigned two priorities: an initial priority and a promoted priority. We adopt the initial priority and promoted priority assignment approach proposed in [22], where the promoted priority of any task is set higher than the initial priority of all other tasks, and tasks with higher initial priorities are assigned correspondingly higher promoted priorities.

This approach ensures that tasks assigned a lower initial priority can complete their execution after their initial priority is promoted, without significantly disrupting tasks with a higher initial priority. Table 3.1 shows an example of priority assignments for a task set containing four tasks.

Task	Initial Priority	Promoted Priority
τ_1	5	1
τ_2	6	2
τ_3	7	3
τ_4	8	4

Table 3.1: Priority example

A summary of the notation and symbols is given in this chapter is shown in Table 3.2.

3.1.3 Scheduling Model

We consider a multiprocessor platform employing global scheduling with M processors. Under global scheduling, tasks may execute on any available processor and are permitted to migrate between processors. Additionally, we assume a preemptive scheduling policy: a currently executing lower-priority task can be preempted at any time by a higher-priority task, interrupting the execution of the lower-priority task and allocating the processor to the higher-priority task.

Symbol	Description
τ	Task set
τ_k	k-th task $\in \tau$
J_k^l	l-th job of task τ_k
C_k	Worst-case computation time of τ_k
D_k	Relative deadline of τ_k
P_k	Relative priority promoted time of τ_k
r_k^l	Release time of job J_k^l
p_k^l	Absolute priority promoted time of job J_k^l
d_k^l	Absolute deadline of job J_k^j
R_k	$\max_{J_k^j \in \tau}(f_k^j - r_k^j)$, i.e. response time of τ_k
$W_k^i(L)$	Workload that τ_k should be work done in length L
$W_k^i(a,b)$	Same as $W_k^i(L)$ but the length L is replaced by the length of the interval (a,b) and $L = b - a$
$W_k^i(a,b)^{UB}$	The upper bound of $W_k^i(a,b)$
$I_k^i(D_k)$	an upper bound on the interference from a higher-priority task τ_i within an interval of length D_k

Table 3.2: List of symbols and descriptions for DA-DP

3.2 Deadline Analysis

Deadline analysis is a straightforward sufficiency test, and a famous approach was introduced by Bertogna et al. in [23].

The main method of deadline analysis is expressed in Equation (3.1). Here, we do analysis for task τ_k, for which D_k represents the deadline of the task τ_k, C_k denotes the execution time of τ_k, and $I_k^i(D_k)$ indicates the interference experienced by τ_k within an interval of length D_k. Interference $I_k^i(D_k)$ occurs when all processors are occupied, leaving no available processor to execute task τ_k. Therefore, if we can accurately calculate the total interference caused by other tasks to τ_k, Equation (3.1) allows us to verify whether there is enough time available for task τ_k to complete its execution before its deadline. If Equation (3.1) holds, which means time is available, we conclude that task τ_k is schedulable.

$$D_k \geq C_k + I_k^i(D_k) \tag{3.1}$$

However, without performing a simulation, it is impossible to precisely calculate the actual interference. An effective existing approach estimates an upper bound on interference by computing the workload generated by each task with a higher priority than τ_k, as shown in Equation (3.2). Workload can be understood as the time higher-priority tasks are running on a processor, and the task τ_k cannot be executed on this processor. In Equation (3.2), $W_k^i(D_k)$ represents the workload

imposed by the higher-priority task τ_i over an interval of length D_k. Although not all of this workload necessarily results in interference, dividing this workload by the number of processors M provides an upper bound for the interference.

$$I_k^i(D_k)^{UB} = \left\lfloor \frac{1}{M} \sum_{i \in hp(k)} \min(W_k^i(D_k), D_k - C_k + 1) \right\rfloor \tag{3.2}$$

Under GDP scheduling, calculating $W_k^i(D_k)$ becomes significantly more complicated due to PPP. Tasks' priorities change at these PPPs, resulting in different sets of tasks that can preempt τ_k before and after its PPP. For instance, before the priority promotion of τ_k, any task with a higher initial priority or already promoted can preempt τ_k. However, after τ_k's PPP, only tasks whose priority has been promoted and possesses a higher promoted priority than τ_k can preempt it. Thus, we separately estimate the upper bounds of workload for two intervals to compute the total workload, as shown in Equation (3.3). Here $W_k^i(r_k^l, p_k^l)^{UB}$ is refers to the workload during the interval (r_k^l, p_k^l), which is the workload before τ_k's priority promotion point, from the job release time r_k^l to τ_k's priority promotion point p_k^l. Conversely, $W_k^i(p_k^l, d_k^l)^{UB}$ is the workload after τ_k's priority promotion point.

$$W_k^i(D_k)^{UB} = W_k^i(r_k^l, p_k^l)^{UB} + W_k^i(p_k^l, d_k^l)^{UB} \tag{3.3}$$

After we calculate $W_k^i(D_k)^{UB}$, we can use this value as the workload to calculate the upper bound of the total interference as shown in Equation (3.2). Then we can do the deadline analysis as expressed in Equation (3.1) to test the schedulability of the task.

Next, we will analyze and derive the corresponding expression for $W_k^i(D_k)^{UB}$ under dual-priority scheduling.

3.3 Computing the Workload

This section will focus on calculation of $W_k^i(r_k^l, p_k^l)^{UB}$ and $W_k^i(p_k^l, d_k^l)^{UB}$. We will consider two cases to calculate the workload: When i is smaller than k, and when i is larger than k.

For each case, we consider a time window length of L. The length of the interval L is equal to the length of the interval under calculation of the workload. For example, when we calculate $W_k^i(r_k^l, p_k^l)^{UB}$, L will equal to length of interval (r_k^l, p_k^l), which is P_k.

When i is smaller than k:

The initial priority of task τ_i is higher than that of task τ_k, and task τ_i's promoted priority is greater than both the initial and promoted priorities of task τ_k.

During interval (r_k^l, p_k^l) $L = P_k$**:**

During the interval (r_k^l, p_k^l), task τ_k maintains its initial priority. In this interval, task τ_i has a higher priority than task τ_k, regardless of whether task τ_i is at its initial or promoted priority. Under these conditions, the calculation of the worst-case workload W can be treated as calculating the worst-case workload W under the worst-case scenario that is depicted in Figure 3.2.

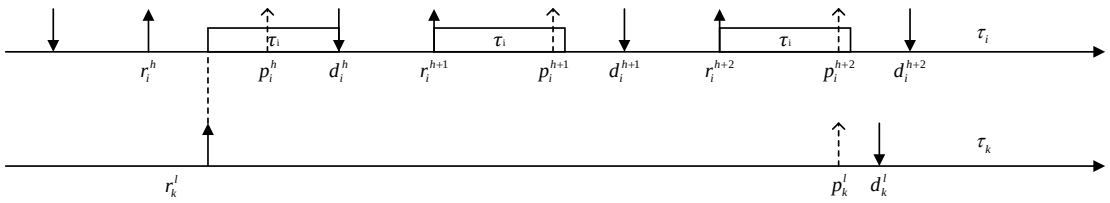

Figure 3.2: Worst-case scenario

This scenario is similar to the scenario for classical GFP scheduling described in Theorem 4 of [18]. There is a job of τ_i that starts at r_k^l and finishes after C_i time units at its deadline, and the subsequent jobs are released and executed as soon as possible. Under this scenario, task τ_i can cause the maximum number of interferences on task τ_k, as shown in [24].

Thus, the number of complete executions of jobs of τ_i within the interval P_k can be calculated as in Equation (3.4). $P_k + D_i - C_i$ is the length from r_i^h to p_k^l, so by dividing this value by T_i and taking the lower bound, we can get the number of complete executions of jobs. This approach will be used many times to calculate the number of fully executed jobs in this section.

$$N_i^L = \left\lfloor \frac{P_k - (C_i + T_i - D_i)}{T_i} \right\rfloor + 1 = \left\lfloor \frac{P_k + D_i - C_i}{T_i} \right\rfloor \tag{3.4}$$

The carry-out workload can be calculated as Equation (3.5). The carry-out workload is the last job at the end of the interval (r_k^l, p_k^l), which may not finish execution before p_k^l. In order to lower bound the value, we use $[a]_0 = \max(a, 0)$ as follows.

$$\varepsilon_{out} = \min(C_i, \left[L + D_i - C_i - N_i^L T_i\right]_0) \tag{3.5}$$

Now $W_k^i(r_k^l, p_k^l)^{UB}$ then can be calculated as Equation (3.6). By summing the execution time of fully executed jobs and the last job, we can get the total workload because of τ_i during the interval (r_k^l, p_k^l).

$$W_k^i(r_k^l, p_k^l)^{UB} = N_i^L \times C_i + \varepsilon_{out} \tag{3.6}$$

During interval (p_k^l, d_k^l) $(L = D_k - P_k)$**:**

In this interval, we will discuss the different positions of p_k^l and d_k^l. We will consider all possible different positions, and the maximum workload should be one of the cases. Then we can take the maximum workload as $W_k^i(p_k^l, d_k^l)^{UB}$.

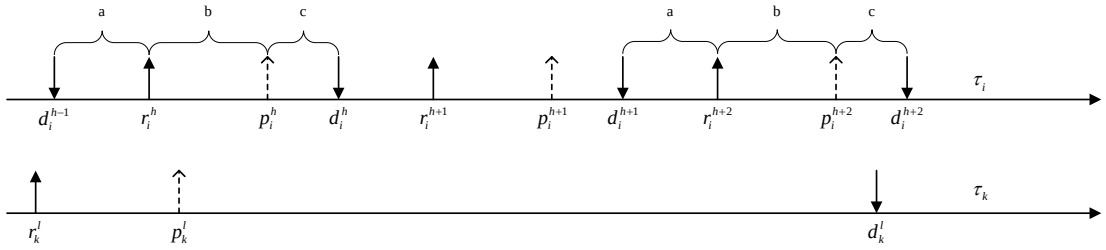

Figure 3.3: Possible position of priority promotion time and deadline

As depicted in Figure 3.3, the possible positions of p_k^l and d_k^l are defined within regions a, b, and c. Therefore, to cover all possible workload upper bounds, the following cases must be considered:
(Note that each interval is left closed and right open, for example: the region a or b belongs to $[d_i^{h-1}, p_i^h)$, while the region c belongs to $[p_i^h, d_i^h)$.)

1. p_k^l in region a or b, d_k^l in region a or b.
2. p_k^l in region a or b, d_k^l in region c.
3. p_k^l in region c, d_k^l in region c.

These three cases cover all the possible positions for the following reason.

First, the regions a, b, and c cover a full execution period of the task, so these three regions actually cover all the time.

Second, due to the priority assignment model that we introduced in section 3.1.2, and i is smaller than k, the promoted priority of task τ_k will be higher than the initial priority of task τ_i. Task τ_i can only cause interference to τ_k in region c. Then we can consider the regions a and b in the same scenario. Based on this, we can consider the regions a and b together, so there will be only four possible combinations.

Moreover, since when $i < k$, task τ_i cannot impact the execution of task τ_k in regions a or b, we only need to specifically consider how to compute the workload in region c, if p_k^l is in region a or b while d_k^l is in region c. Conversely, if d_k^l is in region a or b while p_k^l is in region c, we again only need to specifically consider computing the workload within region c. As we calculate the upper bound rather than the exact value of the workload, we only need one calculation pattern to calculate the workload for these two cases. Thus, the three cases we listed can cover all possible scenarios.

Scenario 1: p_k^l in region a and b, d_k^l in region a and b

In Scenario 1, the possible maximum and minimum lengths of L for interval (p_k^l, d_k^l) are shown in Figure 3.4. The difference between them, denoted as L_{diff}, is given by Equation (3.7). Thus, the range of L can be expressed as: $L \in (L_{\min}, L_{\min} + L_{diff})$.

$$L_{diff} = 2(P_i + T_i - D_i) \tag{3.7}$$

Assuming $L = L_{min}$, we can use Equation (3.8) to calculate the number of complete executions of jobs of τ_i within the interval L.

$$N_i^{L\_S1} = \left\lfloor \frac{L + (T_i - D_i) + P_i}{T_i} \right\rfloor = \left\lfloor \frac{L - (D_i - P_i)}{T_i} \right\rfloor + 1 \tag{3.8}$$

Since $L \in (L_{\min}, L_{\min} + L_{diff})$, as L gradually increases from L_{min} to L_{max}, the calculation result from Equation (3.8) may increase. This causes the computed workload to increase, introducing pessimism into the schedulability test and making it unnecessarily strict. However, L_{diff} is upper bounded by Equation (3.7), so the increase is also bounded.

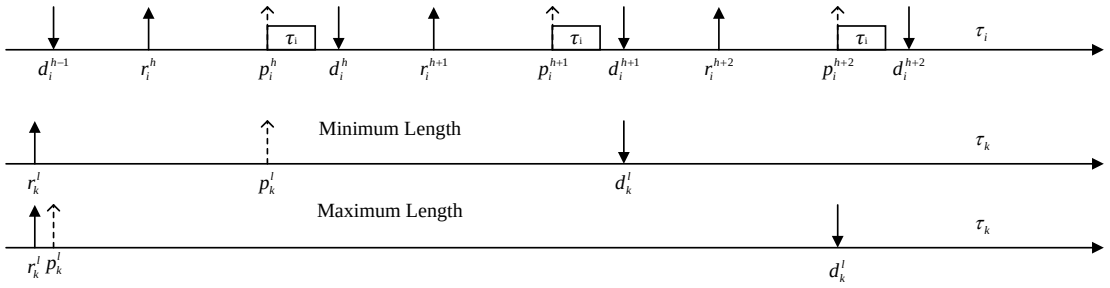

Figure 3.4: Scenario 1

In (p_k^l, d_k^l), the work of τ_i can interfere with τ_k in whole or in part execution time. If the length of the interval (p_i^h, d_i^h) is larger than C_i, the entire execution time of τ_i can interfere with τ_k. If the length of (p_i^h, d_i^h) is smaller than C_i, τ_i will be partially executed and cause interference equal to the length of (p_i^h, d_i^h). The rest of the execution must take place at the initial priority part of the task τ_i, which is the lower priority part and thus won't interfere with τ_k's analysis window.

Then, the maximum interference each job of task τ_i can impose on task τ_k is $\max(D_i - P_i, C_i)$. Consequently, the upper bound of interference under Scenario 1 can be calculated using Equation (3.9), and the equation is valid for all possible values of L in the range $(L_{\min}, L_{\min} + L_{diff})$.

$$W_k^i(L)_{S1}^{UB} = W_k^i(p_k^l, d_k^l)_{S1}^{UB} = N_i^{L\_S1} \times \max(D_i - P_i, C_i) \tag{3.9}$$

Scenario 2: p_k^l in region a and b, d_k^l in region c

Scenario 2 is depicted in Figure 3.5. Similarly, we can determine that $L_{diff} = T_i$, indicating that $L \in (L_{\min}, L_{\min} + L_{diff})$,$(L = D_k - P_k)$.

To compute the number of complete executions of task τ_i for the minimum case, we obtain the result shown in Equation (3.10).

$$N_i^{L\_S2} = \left\lfloor \frac{L}{T_i} \right\rfloor \tag{3.10}$$

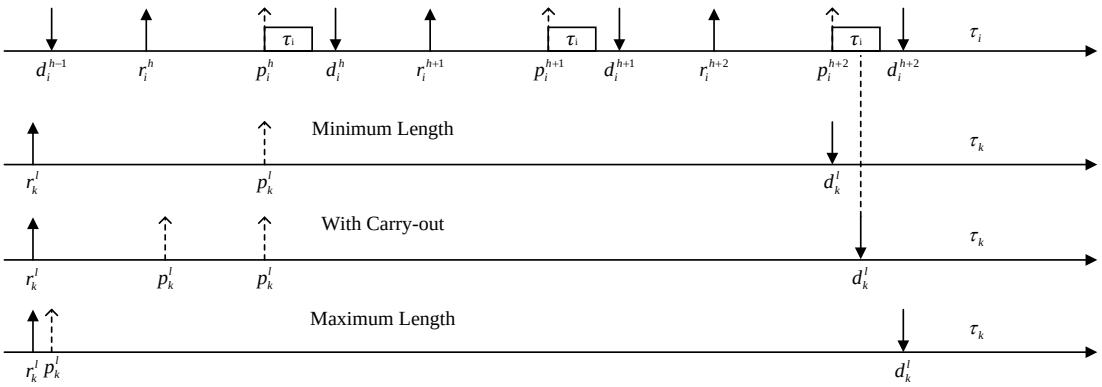

Figure 3.5: Scenario 2

Since $L_{diff} = T_i$, the result calculated from Equation (3.10) remains unchanged under Scenario 2 as L varies within $(L_{\min}, L_{\min} + L_{diff})$ due to the floor function.

Assuming p_k^l is positioned at p_i^h (the position corresponding to L_{min}), the carry-out interference can be computed using Equation (3.11).

$$\varepsilon_{out}^{S2} = \min(\left[L - N_i^{L\_S2} \times T_i\right]_0, C_i, D_i - P_i) \tag{3.11}$$

Thus, $W_k^i(p_k^l, d_k^l)$ in Scenario 2 can be calculated as shown in Equation (3.12).

$$W_k^i(L)_{S2}^{UB} = W_k^i(p_k^l, d_k^l)_{S2}^{UB} = N_i^{L\_S2} \times C_i + \varepsilon_{out}^{S2} \tag{3.12}$$

Scenario 3: p_k^l in region c, d_k^l in region c

Scenario 3 is illustrated in Figure 3.6. Similarly, we can determine that $L_{diff} = 2(D_i - P_i)$. For the minimum-length interval, the number of complete executions of jobs of τ_i can be calculated using Equation (3.13).

$$N_i^{L\_S3} = \left\lfloor \frac{L - (T_i - D_i) - P_i}{T_i} \right\rfloor = \left\lfloor \frac{L + D_i - P_i}{T_i} \right\rfloor - 1 \tag{3.13}$$

In the case depicted in Figure 3.6, $N_i^{L\_S3} \times T_i$ can represent the length from r_i^{h+1} to r_i^{h+2}. [1]

The length from d_i^h to r_i^{h+1} is equal to $T_i - D_i$ and the length from r_i^{h+2} to p_i^{h+2} is equal to P_i. What's more, regardless of whether there is carry-in interference, carry-out interference, or both, the total "carry interference" can be calculated as the length of L(that is from p_k^l to d_k^l) minus the length from d_i^h to p_i^{h+2}. So the interference can be computed using Equation (3.14).

[1](Notice that, there is not only one instance in the length of L, $N_i^{L\_S3} \times T_i$ can represent the length from r_i^{h+1} to the release time of the last fully executed job. In order to make the figure easy to understand, we assume there is only one job here. And we do this assumption for all the fully executed jobs in this section.)

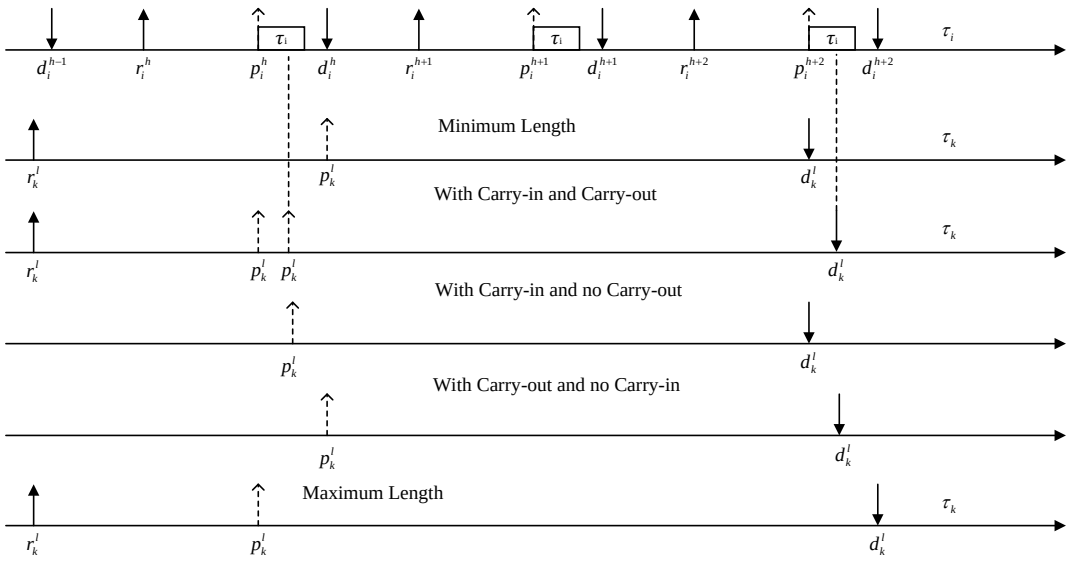

Figure 3.6: Scenario 3

$$\varepsilon_{in}^{S3} + \varepsilon_{out}^{S3} = \min(\Big[L - N_i^{L\_S3} \times T_i - (T_i - D_i) - P_i\Big]_0, 2C_i, 2(D_i - P_i)) \tag{3.14}$$

When $L_{diff} \leq T_i$, which means $2(D_i - P_i) \leq T_i$, the result obtained from Equation (3.13) remains constant as L increases from L_{min} to L_{max}.

However, when $L_{diff} > T_i$, the result from Equation (3.13) may increase as L grows from L_{min} to L_{max}, they may introduce pessimism, but not necessarily.

Thus, the workload interference $W_k^i(p_k^l, d_k^l)$ in Scenario 3 can be calculated as shown in Equation (3.15) for all the L in range of $(L_{\min}, L_{\min} + L_{diff})$.

$$W_k^i(L)_{S3}^{UB} = W_k^i(p_k^l, d_k^l)_{S3}^{UB} = N_i^{L\_S3} \times C_i + \varepsilon_{in}^{S3} + \varepsilon_{out}^{S3} \tag{3.15}$$

When i is larger than k:

The initial priority of task τ_i is lower than that of task τ_k, and task τ_i's promoted priority is also lower than the promoted priority of task τ_k.

During interval (r_k^l, p_k^l):

Within this time interval, task τ_i can only preempt task τ_k during its own high-priority execution phase (p_i^h, d_i^h). This case has three scenarios which are similar to the previously discussed case where $i < k$ and the interval of interest is (p_k^l, d_k^l), which are shown in Figure 3.4, Figure 3.5, and Figure 3.6. The value of the workload still depends on the relative positions of r_k^l and p_k^l with respect to r_i^h, p_i^h, and d_i^h. The only difference lies in the starting point of the interval. Thus, to adapt the original formula, it is sufficient to replace the relevant parameter L with P_k.

During interval (p_k^l, d_k^l):

Since task τ_i has a lower priority than task τ_k even after promotion, it cannot preempt τ_k at any time during the interval (p_k^l, d_k^l). Therefore, in this case, the interference function $W_k^i(p_k^l, d_k^l) = 0$.

3.3.1 Summary and Evaluate

$W_k^i(D_k)^{UB}$ is calculated by Equation (3.16) in which $W_k^i(r_k^l, p_k^l)^{UB}$ is calculated as Equation (3.17) and $W_k^i(p_k^l, d_k^l)^{UB}$ is calculated as Equation (3.18). $W_k^i(L)_{S1}^{UB}$, $W_k^i(L)_{S2}^{UB}$ and $W_k^i(L)_{S3}^{UB}$ in Equation (3.17) and Equation (3.18) is calculated as Equation (3.9), Equation (3.12) and Equation (3.15) respectively. The length of the interval L is equal to the length of the interval under calculation of the workload.

$$W_k^i(D_k)^{UB} = W_k^i(r_k^l, p_k^l)^{UB} + W_k^i(p_k^l, d_k^l)^{UB} \tag{3.16}$$

$$W_k^i(r_k^l, p_k^l)^{UB} = \begin{cases} \left\lfloor \dfrac{P_k + D_i - C_i}{T_i} \right\rfloor C_i & \\ + \min\left(C_i,\ L + D_i - C_i - \left\lfloor \dfrac{P_k + D_i - C_i}{T_i} \right\rfloor T_i\right) & \text{if } i < k \\ \text{with } L = P_k, & \\ & \\ \max\left(W_k^i(L)_{S1}^{UB},\ W_k^i(L)_{S2}^{UB}, W_k^i(L)_{S3}^{UB}\right) & \text{otherwise} \\ \text{with } L = P_k, & \end{cases} \tag{3.17}$$

$$W_k^i(p_k^l, d_k^l)^{UB} = \begin{cases} \max\left(W_k^i(L)_{S1}^{UB},\ W_k^i(L)_{S2}^{UB}, W_k^i(L)_{S3}^{UB}\right) & \text{if } i < k \\ \text{with } L = D_k - P_k, & \\ & \\ 0, & \text{otherwise} \end{cases} \tag{3.18}$$

We perform many experiments for this workload calculation approach. We calculate the workload for different task sets by using Equation (3.16), Equation (3.17), and Equation (3.18). We then found that most of the time, the calculated workload will be significantly larger than the actual workload.

We identified these two main reasons:

First, we partitioned the computation of $W_k^i(D_k)^{UB}$ into two intervals without fully considering the potential impact of unfinished jobs from the first interval on the second interval's workload. Specifically, we independently calculated the upper-bound workloads for each interval. As a result, if there were unfinished jobs in interval one, their entire execution times were counted as workload in both intervals, introducing unnecessary pessimism.

Second, we categorized workload calculation into three separate scenarios. However, since the scenario applicable to each task is uncertain, and to ensure the correctness of the schedulability test, we consistently selected the maximum value from these scenarios. This approach introduced additional pessimism, particularly if the maximum computed workload frequently overestimates the actual workload.

Our approach can be improved based on these insights, but this is for future work. In order to calculate the workload in a good way, we propose another approach for dual-priority scheduling in the next section.

3.4 Improved Workload Computing

This section will focus on a new approach to calculate $W_k^i(r_k^l, p_k^l)^{UB}$ and $W_k^i(p_k^l, d_k^l)^{UB}$.

When i is smaller than k:

The initial priority of task τ_i is higher than that of task τ_k, and task τ_i's promoted priority is greater than both the initial and promoted priorities of task τ_k.

During interval (r_k^l, p_k^l):

We consider the worst-case scenario shown in Figure 3.7, where a job of task τ_i starts execution at time r_k^l and finishes at its deadline after executing C_i time units, and the subsequent tasks execute as early as possible. This scenario aligns with the worst-case condition described in [18]. This scenario represents the worst case, and no other pattern would lead to a higher workload.

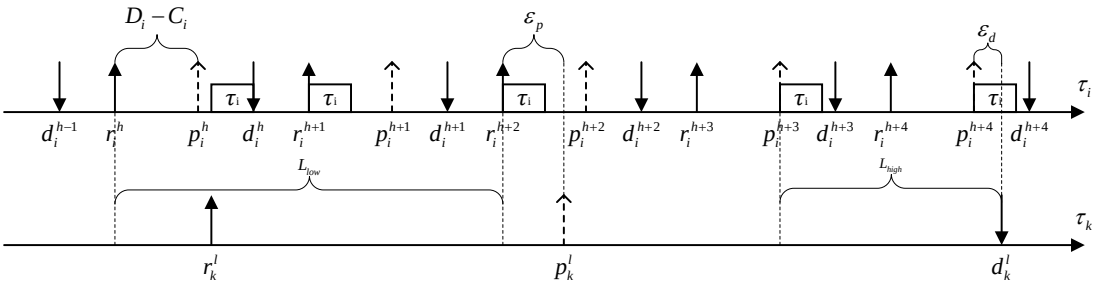

Figure 3.7: Worst-case scenario when $C_i < \varepsilon_p$

As shown in Figure 3.7, the interval from r_i^h to p_k^l has a length of $P_k + D_i - C_i$. Therefore, using Equation (3.19), we can determine the number of jobs of task τ_i that complete full execution time within the interval (r_k^l, p_k^l), denoted as N_i^{low}.

$$N_i^{low} = \left\lfloor \frac{P_k + D_i - C_i}{T_i} \right\rfloor \tag{3.19}$$

We define L_{low} as the length from the release of the first job and the release of the last job in the interval (r_k^l, p_k^l). The range of L_{low} is also shown in Figure 3.7. This

length is the same as the total period for the jobs that complete full execution time within the interval (r_k^l, p_k^l). Consequently, the length of L_{low} is given by $N_i^{low} \times T_i$.

The ε_p shown in Figure 3.7 is the carry-out workload that is due to the last job of task τ_i in the interval (r_k^l, p_k^l). This job may not be fully executed inside the interval (r_k^l, p_k^l). We need to calculate the value of the execution time for this job, and calculate the remaining part of its execution time in the interval (p_k^l, d_k^l). There is some pessimism if we calculate this job as a fully executed job, because when we calculate the workload for the interval (p_k^l, d_k^l), we may calculate it again as a fully executed job, which will introduce unnecessary pessimism.

Since $L_{low} + \varepsilon_p = P_k + D_i - C_i$, we can derive the value of ε_p using Equation (3.20).

$$\varepsilon_p = \left[P_k + D_i - C_i - N_i^{low} \times T_i\right]_0 \tag{3.20}$$

When $C_i > \varepsilon_p$, the maximum carry-out interference is ε_p; however, when $C_i < \varepsilon_p$, the maximum carry-out interference becomes C_i. Thus, the interference within the interval (r_k^l, p_k^l) can be calculated using Equation (3.21).

$$W_k^i(r_k^l, p_k^l)^{UB} = N_i^{low} \times C_i + \min(\varepsilon_p, C_i) \tag{3.21}$$

During interval (p_k^l, d_k^l):

In the interval (p_k^l, d_k^l), the maximum interference depends on the relationship between ε_p and C_i. If $C_i < \varepsilon_p$, the worst-case scenario remains as shown in Figure 3.7.

However, when $C_i > \varepsilon_p$, as shown in Figure 3.8, the last job of task τ_i before p_k^l cannot be completed and will instead execute after the priority of task τ_i promoted. This unfinished part of the work needs to be finished at the first promoted priority part of the task τ_i in the interval (p_k^l, d_k^l). Specifically, in Figure 3.8, the job release at r_i^{h+2} cannot finished before p_k^l, then the remaining part need to be executed at τ_i's promoted priority, which is the interval (p_i^{h+2}, d_i^{h+2}), introducing additional workload to $W_k^i(p_k^l, d_k^l)^{UB}$,

First, we will discuss the case when $C_i < \varepsilon_p$:

When $C_i < \varepsilon_p$, as shown in Figure 3.8, all jobs of task τ_i before p_k^l can complete execution, thus not causing execution after p_i^{h+2}.

We define L_{high} as the length from the release time of the first fully executed job in the interval (r_k^l, p_k^l) to d_k^l. In Figure 3.8, it is the interval (p_i^{h+3}, d_k^l). In order to calculate L_{high}, we need to calculate the length of the number of fully executed jobs of τ_i before p_k^l, including the last job of task τ_i before p_k^l. We obtain N_i^{high} as shown in Equation (3.22). The upper bound function can include the last period of task τ_i before p_k^l in the count.

$$N_i^{high} = \left\lceil \frac{P_k + D_i - C_i}{T_i} \right\rceil \tag{3.22}$$

Then $N_i^{high} \times T_i$ can represent the length of the interval (p_i^h, p_i^{h+3}), and we can use the length of the interval (p_i^h, d_k^l) minus the length of the interval (p_i^h, p_i^{h+3}) to get

L_{high}. Consequently, the length from p_i^h to r_i^{h+3} can be computed as $N_i^{high} \times T_i$, and L_{high} can be calculated using Equation (3.23).

$$L_{high} = \Big[D_k + D_i - C_i - N_i^{high} \times T_i - P_i\Big]_0 \tag{3.23}$$

Then we can use L_{high} to calculate the carry-out workload ε_d in the interval (p_k^l, d_k^l), which is the workload due to the last job of task τ_i in (p_k^l, d_k^l). This job may not be fully executed, as shown by ε_d in Figure 3.7, so we need to calculate this part separately.

Since task τ_i can only preempt task τ_k after its priority promoted point after p_k^l, the maximum interference caused by task τ_i in the interval (p_k^l, d_k^l) is $\min(D_i - P_i, C_i)$, as shown in Equation (3.24).

$$C_i^{'} = \min(D_i - P_i, C_i) \tag{3.24}$$

It can be noticed that the length from p_i^{h+3} to p_i^{h+4} is also T_i; Therefore, we can calculate the length of the interval (p_i^{h+3}, p_i^{h+4}) by $\left\lfloor \frac{L_{high}}{T_i} \right\rfloor T_i$. Then we just need to use L_{high} minus the length of the interval (p_i^{h+3}, p_i^{h+4}) to calculate ε_d, shown as Equation (3.25). We also bounded it by $C_i^{'}$ because it cannot execute beyond its execution time. Thus, the interference within the interval (p_k^l, d_k^l) can be computed using Equation (3.26).

$$\varepsilon_d = \min(C_i^{'}, L_{high} - \left\lfloor \frac{L_{high}}{T_i} \right\rfloor T_i) \tag{3.25}$$

$$W_k^i(p_k^l, d_k^l)^{UB} = \left\lfloor \frac{L_{high}}{T_i} \right\rfloor C_i^{'} + \varepsilon_d \quad (\text{while} \quad C_i < \varepsilon_p) \tag{3.26}$$

Now we will disscuss the case when $C_i > \varepsilon_p$:

When $C_i > \varepsilon_p$, as shown in Figure 3.8, the last job of task τ_i before p_k^l cannot complete execution, which leads to additional workload after the priority promoted point at p_i^{h+2}. We use a dashed rectangle in Figure 3.8 to represent this additional workload, and use ε_{addi} to represent it in the equations.

This additional interference ε_{addi} can be calculated as $C_i - \varepsilon_p$, but since after p_k^l, task τ_i can only executed after its PPP, so we need to bound this workload by the length of the promoted priority part $D_i - P_i$. Then the calculation of ε_{addi} is as shown in Equation (3.27).

$$\varepsilon_{addi} = \min(D_i - P_i, C_i - \varepsilon_p) \tag{3.27}$$

Consequently, based on Equation (3.26), the value of $W_k^i(p_k^l, d_k^l)^{UB}$ when $C_i > \varepsilon_p$ is given by Equation (3.28), which ε_{addi} is considered.

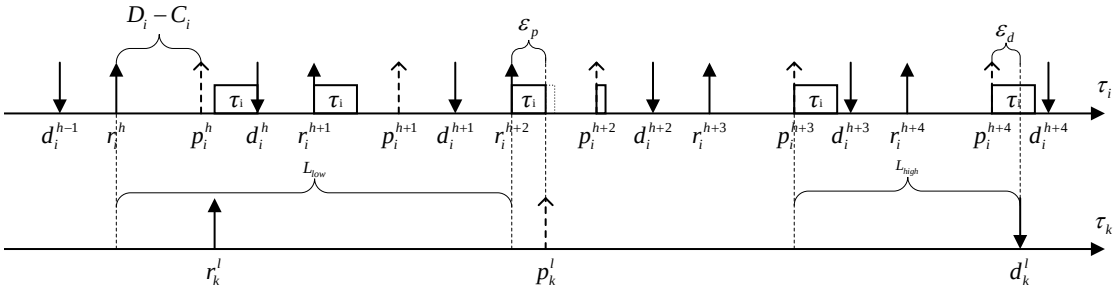

Figure 3.8: Worst-case scenario when $C_i > \varepsilon_p$

$$W_k^i(p_k^l, d_k^l)^{UB} = \left\lfloor \frac{L_{high}}{T_i} \right\rfloor C_i' + \varepsilon_d + \varepsilon_{addi} \quad (\text{while} \quad C_i > \varepsilon_p) \tag{3.28}$$

Moreover, we can noticed that when $C_i > \varepsilon_p$, the value of ε_{addi} is positive, whereas when $C_i < \varepsilon_p$, ε_{addi} is negative. Hence, we can integrate the calculations for $W_k^i(p_k^l, d_k^l)^{UB}$ under both conditions $C_i > \varepsilon_p$ and $C_i < \varepsilon_p$ into a unified expression, as presented in Equation (3.29).

$$W_k^i(p_k^l, d_k^l)^{UB} = \left\lfloor \frac{L_{high}}{T_i} \right\rfloor C_i' + \varepsilon_d + [\varepsilon_{addi}]_0 \tag{3.29}$$

When i is larger than k:

The initial priority of task τ_i is lower than that of task τ_k, and task τ_i's promoted priority is also lower than the promoted priority of task τ_k.

Since in the interval (p_k^l, d_k^l), the priority of task τ_i will never be higher than that of task τ_k. So there is no interference in the interval (p_k^l, d_k^l), which means $W_k^i(p_k^l, d_k^l)^{UB} = 0$. N_i^{low} we can calculate as same approach when i<k, but task τ_i can only cause the interference on task τ_k after its priority promotion point, so we need to bound C_i here as well, as shown in Equation (3.30). Then the new N_i^{bound} can be calculated as Equation (3.31).

$$C_i^{bound} = \min(D_i - P_i, C_i) \tag{3.30}$$

$$N_i^{bound} = \left\lfloor \frac{P_k + D_i - C_i^{bound}}{T_i} \right\rfloor \tag{3.31}$$

When i is larger than k, we still need to consider the carry-out workload, which we use ε_p to represent in this case. As shown in Figure 3.9, if $\varepsilon_p < P_i$, the carry-out interference is equal to 0. If $\varepsilon_p > P_i$ the carry-out interference is equal to $\min(\varepsilon_p - P_i, C_i^{bound})$. Thus, $W_k^i(r_k^l, p_k^l)^{UB}$ can be calculated as shown in Equation (3.33).

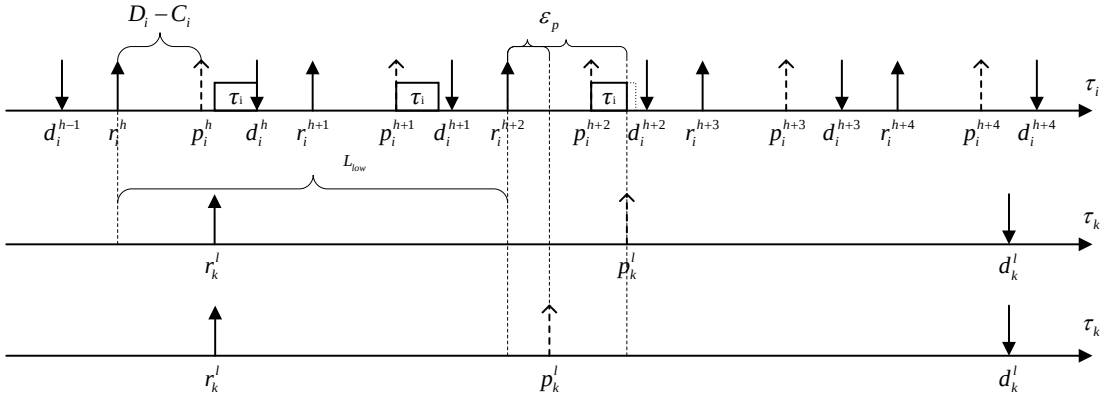

Figure 3.9: Worst-case scenario when i>k

Since we use a bounded C_i here, the calculation of ε_p should be bounded too. We can calculate ε_p^{bound} by Equation (3.32).

$$\varepsilon_p^{bound} = P_k + D_i - C_i^{bound} - N_i^{bound} \times T_i \tag{3.32}$$

$$W_k^i(r_k^l, p_k^l)^{UB} = N_i^{bound} \times C_i^{bound} + \min(\left[\varepsilon_p^{bound} - P_i\right]_0, C_i^{bound}) \tag{3.33}$$

3.5 Overall DA-DP test

The DA-DP test is shown in Equation (3.34). Note that under GDP scheduling, even tasks initially assigned lower priorities can interfere with higher initial priority tasks after their priority promotion points. Specifically, once a task's priority has been promoted, its priority level may be promoted during execution, and higher than the priorities of tasks that had a higher priority at the beginning, enabling it to preempt and cause interference to these tasks. As a result, when computing the workload $W_k^i(D_k)$, it becomes essential to consider the workload contributions from all tasks (except the task currently being analyzed, τ_k), rather than restricting the calculation solely to tasks initially assigned higher initial priorities. This comprehensive approach ensures that all potential interference scenarios, resulting from priority promotions, are accurately captured.

If Equation (3.34) holds, then the task set is schedulable. The calculation of $W_k^i(D_k)^{UB}$ is shown in Equation (3.35), Equation (3.36) and Equation (3.37).

N_i^{low}, ε_p, in Equation (3.36) is calculated as Equation (3.19) and Equation (3.20). C_i^{bound}, N_i^{bound} and ε_p^{bound} is calculated as Equation (3.30), Equation (3.31) and Equation (3.32).

ε_d, ε_{addi}, L_{high} and $C_i^{'}$ in Equation (3.37) is calculated as Equation (3.25), Equation (3.27), Equation (3.23) and Equation (3.24) respectively.

$$D_k \geq C_k + \left\lfloor \frac{1}{m} \sum_{i \in \{\tau \setminus \tau_k\}} \min(W_k^i(D_k)^{UB}, D_k - C_k + 1) \right\rfloor \tag{3.34}$$

$$W_k^i(D_k)^{UB} = W_k^i(r_k^l, p_k^l)^{UB} + W_k^i(p_k^l, d_k^l)^{UB} \tag{3.35}$$

$$W_k^i(r_k^l, p_k^l)^{UB} = \begin{cases} N_i^{low} \times C_i + \min(\varepsilon_p, C_i), & \text{if } i < k \\ N_i^{bound} \times C_i^{bound} + \min(\left[\varepsilon_p^{bound} - P_i\right]_0, C_i^{bound}) & \text{otherwise} \end{cases} \tag{3.36}$$

$$W_k^i(p_k^l, d_k^l)^{UB} = \begin{cases} \left\lfloor \frac{L_{high}}{T_i} \right\rfloor C_i' + \varepsilon_d + \max(0, \varepsilon_{addi}) & \text{if } i < k \\ 0, & \text{otherwise} \end{cases} \tag{3.37}$$

4

Priority Assignment Algorithm

Based on the DA-DP test we have in section 3.5, we know how to test the schedulability of a task set with a given set of PPPs. Then, in order to apply the global dual priority scheduling, we still need to know how to assign the priority and PPPs for a task set.

In this chapter, we will first introduce an existing algorithm to assign the priority for the task under the global fixed priority scheduling, which is the DA-OPA algorithm. Then, based on the DA-OPA algorithm, we derive a hybrid algorithm called the DA-OPA-DP algorithm, which can be used for the global dual priority scheduling. At the end of the chapter, we talk about how to assign the PPPs as well.

4.1 DA-OPA Algorithm

The OPA algorithm can determine an optimal static priority ordering for a given schedulability test, as we introduced in section 2.3.1. We will show how the existing DA test from [1] works with the OPA algorithm in this section.

The algorithm with the DA test is shown in Algorithm 2. Starting from the lowest priority level, the algorithm attempts to assign each unassigned task to the current priority, treating all other tasks as having higher priorities, and performs the DA test on the selected task. If it passes the test, the task is assigned to this priority, and the algorithm proceeds to the next higher priority level. If the task does not pass the test, the algorithm tries to assign other unassigned tasks at the current priority level until all tasks have been tested. If no tasks can be assigned to this priority level, the algorithm concludes that the task set is unschedulable and terminates. If all tasks are successfully assigned priorities, the algorithm returns a schedulable result.

4.2 DA-OPA-DP Algorithm

To combine the strengths of DA-DP and the existing DA-OPA approach, we introduce a hybrid algorithm, the DA-OPA-DP algorithm. In this hybrid algorithm, we will first assign the priority in the DA-OPA algorithm. If the algorithm returns an unschedulable result, we will take the remaining tasks that haven't been assigned a priority by the DA-OPA algorithm, into a new subset of the original task set. Then we will assign the priority promotion points to the tasks in this subset. After the

Algorithm 2 :DA-OPA

```
1: N = Number of Task
2: for each priority level n from level N to level 1, lowest first do
3:     for each unassigned task τ_i do
4:         if τ_i is schedulable at priority level n according to schedulability test DA
5:             with all unassigned tasks assumed to have higher priorities then
6:             Assign τ to priority n
7:             break (back to line 2, new loop with priority level n-1)
8:         end if
9:         if none of the tasks can be assigned at priority n  then
10:            return unschedulable
11:        end if
12:    end for
13: end for
14: return schedulable
```

PPPs are assigned, we will do the OPA algorithm again for the subset, but with the proposed DA-DP test for the global dual priority scheduling.

Next, we will introduce in detail the DA-OPA-DP algorithm. The overall DA-OPA-DP algorithm is shown in Algorithm 3. The strategy we used on line 10 in Algorithm 3 will be discussed in detail in the next section.

As shown in Algorithm 3, the DA-OPA-DP algorithm introduces some main modifications to the existing DA-OPA algorithm:

When the DA-OPA algorithm using the DA test fails to assign a task at a certain priority level, as shown in the loop from line 13 in Algorithm 3, instead of immediately concluding unschedulability, the DP algorithm is then only applied to these priority-unassigned tasks. These tasks' both initial and promoted priorities will have higher priority than the fixed priority of the tasks for which priorities are already assigned, as shown in lines 16 and 17 in Algorithm 3.

Notice that the way we assume all unassigned tasks have higher priorities in line 16 of Algorithm 3 is that we assume that both the initial and promoted priorities of the remaining unassigned tasks are higher than that of the initial and promoted priority of the task τ_k under consideration.

A straightforward example is illustrated in Figure 4.1. The procedure of this example follows the blue line, starting from the bottom left to the top right. Consider a task set with 10 tasks. The DA-OPA-DP algorithm initially attempts to assign priorities using the DA test, and each task here only has one priority level without any PPPs. Starting from priority level 21 and progressing to the next higher priority level until a failure occurs. Assuming a failure occurs at priority level 17, this means 4 tasks have successfully been assigned priorities. At this stage, the remaining 6 unassigned tasks are going to the DP-OPA procedure, starting from priority level 16. Thus, the remaining tasks' initial and promoted priorities are both higher than those assigned previously using DA-OPA.

Algorithm 3 :DA-OPA-DP

1: N = Number of Task
2: **for** each priority level n from $2N + 1$ to level $N + 1$, lowest first **do**
3: **for** each unassigned task τ_i **do**
4: **if** τ_i is schedulable at priority level n according to schedulability test DA
5: with all unassigned tasks assumed to have higher priorities **then**
6: **Assign** τ_i to priority n
7: **break** (back to line 2, continue new loop with priority level n-1)
8: **end if**
9: **if** none of the tasks can be assigned at priority n **then**
10: Take all the unassigned tasks left in the task set as a new task set.
11: assign the priority promotion point by a strategy
12: N_{remain} = Number of unassigned tasks
13: **for** each priority level $n^{'}$ from N_{remain} to level 1, lowest first **do**
14: **for** each remaining unassigned task τ_k **do**
15: **if** τ_k is schedulable at priority level $n^{'}$ according to the DA-DP test, assuming all unassigned tasks have higher initial and promoted priorities in comparison to that of the task τ_k's initial and promoted priority, respectively **then**
16: **Assign** initial priority of τ_k to $n^{'}$
17: **Assign** promoted priority of τ_k to $n^{'} - N$
18: **break**
19: (back to line 13, continue a new loop with $n^{'}$-1 priority level)
20: **end if**
21: **if** none of the tasks can be assigned at priority $n^{'}$ **then**
22: **return** unschedulable
23: **end if**
24: **end for**
25: **end for**
26: **end if**
27: **end for**
28: **end for**
29: **return** schedulable

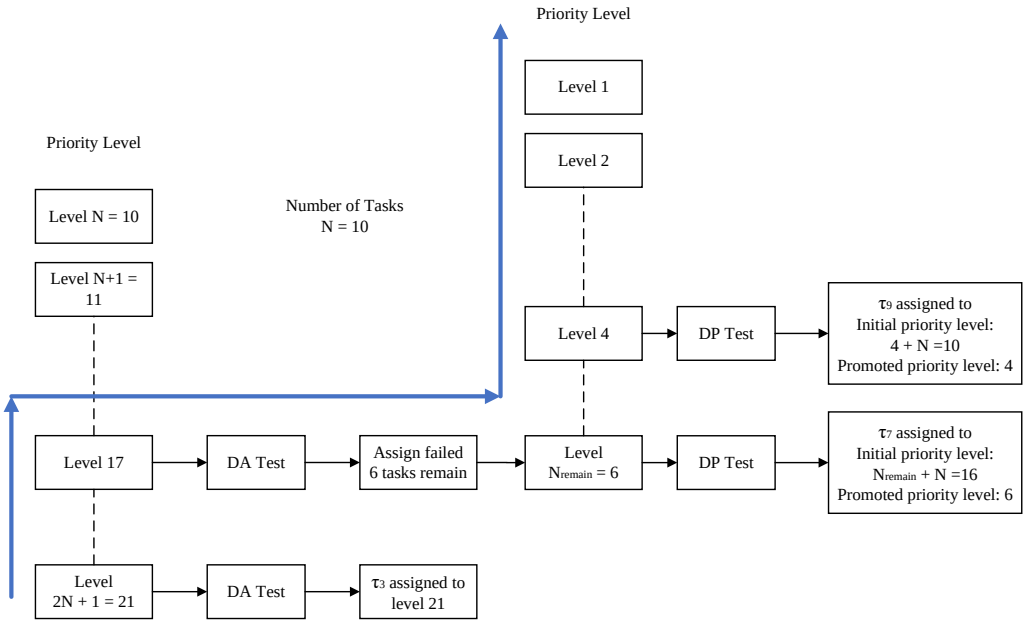

Figure 4.1: Example of DA-OPA-DP

4.3 Priority Promotion Time Assign Strategy

To implement the DA-OPA-DP algorithm, it is necessary to develop a strategy for assigning the priority promotion point (PPP) for each task within the task set. The study in [10] describes a heuristic for assigning PPP for single-processor dual-priority scheduling scenarios, where the PPP for task τ_k is computed according to Equation 4.1, where hp means the task set that includes the task has higher priority than task τ_k.

$$P_k = D_k - C_k - \sum_{i \in hp} C_i \tag{4.1}$$

However, directly applying this method would require the assigned task priorities to determine the PPP of higher-priority tasks, making it incompatible with the OPA algorithm.

To address this issue, we propose calculating the PPP based solely on fundamental task parameters such as deadlines, periods, or utilization.

After extensive simulations with various strategies, we found that setting PPP based on task utilization yields superior results. Specifically, tasks with higher utilization should have earlier PPP (smaller P_k values), whereas tasks with lower utilization should have later PPP (larger P_k values).

Different PPP assignment strategies were extensively tested, and their comparative performance will be discussed further in Chapter 6 and Chapter 7.

5

Simulation

In this chapter, we introduce the simulator that we implemented to evaluate our proposed DA-OPA-DP algorithms with schedulability tests. What's more, the simulator is also used to test different PPP assignment strategies. We will first introduce how we do the different simulations, then talk about the details of the simulator.

5.1 Simulation Methods

In this section, we introduce the simulation method used to identify a strategy for assigning the PPP, as well as the simulation conducted to evaluate the performance of the DA-OPA-DP algorithms.

5.1.1 Simulation 1

In order to use a good strategy to assign the PPP when we evaluate the DA-OPA-DP algorithm, we will first do a simulation to test different PPP assignment strategies in simulation 1. In this simulation, we will only change the PPP assignment strategy before we apply the DA-OPA-DP algorithm.

Existing research lacks studies applying dual-priority scheduling on multiprocessors; hence, there is no established PPP assignment strategy suitable for multiprocessor systems. However, some studies have applied priority promotion on single-processor systems, providing PPP assignment strategies that can serve as references. We referred to the PPP assignment strategy from [11] that we discussed previously in section 2.3.2, which is called the EPP strategy, represented as Equation (5.1), termed as Heuristic 1.

$$H_1 : P_k = D_k - C_k - \sum_{i \in hp} C_i \tag{5.1}$$

In Equation (5.1), P_k represents the relative priority promotion time of task τ_k. By adding P_k to the job's release time, we calculate the job's absolute PPP. C_k is the execution time of task τ_k, and $\sum_{i \in hp} C_i$ is the cumulative execution time of tasks that have a higher initial priority than τ_k.

Since Heuristic 1 was designed for a single-processor platform, we adapted it for multiprocessors by dividing $\sum_{i \in hp} C_i$ by the number of processors M, as shown in

Equation (5.2), termed Heuristic 2. However, Heuristic 1 and Heuristic 2 need to know the priority order to calculate the total execution time of higher priority tasks. But when applying the OPA algorithm, we do not know the exact priority order. So we cannot use Heuristic 1 and Heuristic 2 to assign the PPP with the OPA algorithm. Since we do not know the PPPs, we cannot apply our schedulability test. So we cannot use our hybrid DA-OPA-DP algorithm. Instead, we do a simple simulation that does the deadline analysis first, using the same way in [11] to consider the tasks for assigning priorities (i.e., the task with a relatively larger period will have the lower initial and promoted priorities, respectively, in comparison to that of the tasks with shorter periods). We filter out(i.e., discard) the schedulable task set, since these are already schedulable using the test for GFP. For the rest of the task sets, we apply the DA-OPA test to see how much improvement Heuristic 1 can have. When we generate 2000 tasks, there are only one or two task sets that are unschedulable under GFP with the DA test, but schedulable under GDP with the DA-DP test. This shows that these two heuristics are not suitable for a multiprocessor platform. So we will not consider these two heuristics.

$$H_2 : P_k = D_k - C_k - \left\lfloor \frac{1}{M} \sum_{i \in hp} C_i \right\rfloor \tag{5.2}$$

Considering the fact that the tasks require priority promotion because they cannot be completed within their execution time before the deadline due to preemption from other tasks, we proposed Heuristic 3 (Equation (5.3)) by incorporating task utilization. Here, U_k denotes the utilization of task τ_k, calculated as its execution time divided by its period, reflecting the task load.

$$H_3 : P_k = (D_k - C_k)(1 - U_i) = D_k(1 - U_k)^2 \tag{5.3}$$

During testing, Heuristic 3's results remained unsatisfactory. Therefore, we modified Heuristic 3 by changing the squared term to an exponent x as shown in Equation (5.4), and simulated various x values. Although different values of x affected results differently, there was no significant improvement. Detailed results are also discussed in Chapter 6.

$$H_4 : P_k = D_k(1 - U_k)^x \tag{5.4}$$

Considering that our hybrid algorithm will only try GDP scheduling for the remaining tasks that cannot be scheduled by OPA with GFP, as shown in line 13 of Algorithum 3, we thought of taking the number of remaining tasks N_{remain} into account as a parameter in the heuristic. Then we formulated Heuristic 5 (Equation (5.5)). Simulation results showed that Heuristic 5 performed notably better, which will be discussed in detail in Chapter 6.

$$H_5 : P_k = \left(1 - \frac{U_k}{N_{remain} \times 10}\right) \times D_k \tag{5.5}$$

5.1.2 Simulation 2

To evaluate our proposed DA-OPA-DP algorithm, we apply the best PPP assignment strategy from Simulation 1 and simulate varying input parameters. To isolate and assess the impact of individual parameters, the following tests were conducted:

1. Varying only the number of tasks within task sets across different utilizations.
2. Varying only the number of processors across different utilizations.

5.2 Simulator

Our simulator aims to evaluate our proposed algorithms and test different PPP assignment strategies.

First, in order to evaluate our proposed DA-OPA-DP algorithms, we need to test our DA-OPA-DP algorithms with different randomly generated task sets. We will generate many random task sets under different configurations of simulation parameters to see how many task sets can get a schedulable result under the DA-OPA-DP algorithms. If the task set cannot be schedulable under other algorithms, but can be schedulable under DA-OPA-DP algorithms, that means the DA-OPA-DP test is better. The more task sets that can be schedulable under the algorithm, the better the algorithm is.

We will generate the task set based on the input shown on the left of Figure 5.1. Here, N is to decide how many tasks in the task set, M means the number of processors, and U means the total utilization/load of the task set. The range of period $[a, b]$ means that the task set generator will generate the period in the interval $[a, b]$.

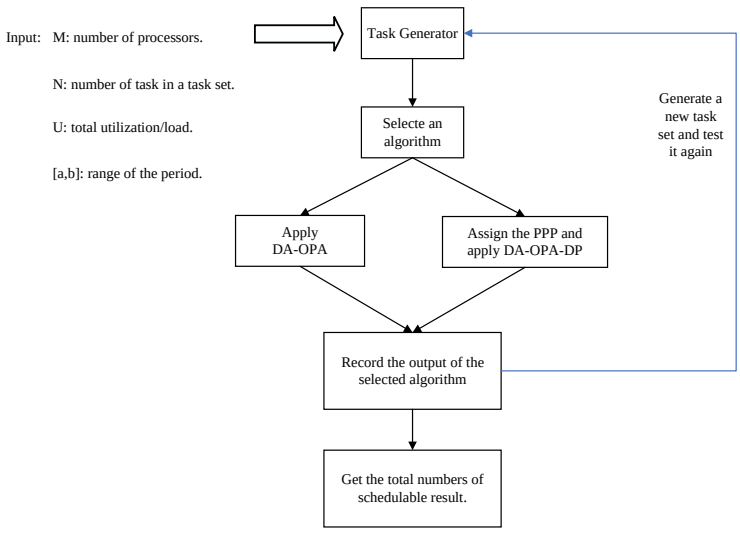

Figure 5.1: Simulator process

The simulator primarily consists of three stages shown in Figure 5.1: the task set generator, the schedulability tests, and the result.

The task set generator will generate a task set based on the input, such as the number of processors, the number of tasks, the total utilization, and the range of periods. Then we apply a selected algorithm with the generated task set and get the output of the algorithm, schedulable or unschedulable. We will run this loop that generates a task set and applies an algorithm 2000 times. The task set generator will generate different task sets based on the input for each loop. We will record the output of the algorithm for each loop, and calculate the number of the loop that returns a schedulable result. Then divide this number by 2000, and we get the percentage of the task set that can be deemed to be schedulable under the selected algorithm. The higher the percentage, the more effective the algorithm.

After finishing the evaluation of the algorithms, we can simply change the strategy used to assign the PPPs and run the simulator to compare the different percentages of schedulable output of the DA-OPA-DP algorithm to test different PPP assignment strategies.

5.2.1 Task Set Generator

The first component of the simulator is the task set generator. We will introduce the implementation details of the task set generator in this subsection. The overall workflow is illustrated in Figure 5.2.

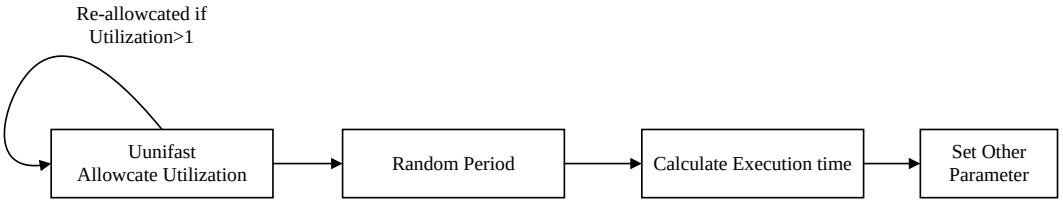

Figure 5.2: Task set generator process

Initially, we adopt the UUnifast-Discard algorithm, as described in [1], to assign individual task utilization, and shown as algorithm 4. U_i in the algorithm is the utilization/load of the task τ_i.

It is important to note that the discard part of the UUnifast-Discard algorithm is to throw away task sets that are invalid. These invalid task sets are mainly because this algorithm cannot directly generate task sets with a total utilization $U > 1$. When applied in the context of multiprocessor systems, this limitation may lead to some tasks being assigned a utilization greater than 1, as noted in [1]. To address this issue, whenever a task with utilization exceeding 1 is detected, the entire task set is discarded and regenerated. This retry process is repeated until a valid task set is obtained or until a maximum of 1000 attempts have been reached.

Once all task utilizations are successfully generated, a random period is assigned to each task within the input period range. The execution time of each is then calculated by multiplying the utilization and the period of the task. For implicit-deadline tasks, the deadline is set equal to the period. For constrained-deadline

Algorithm 4 :UUnifast-Discard

Input: Total Utilization = U
Number of tasks in a task set = N
Number of processors = M
for i = 1 to $n-1$ **do**
  assign task τ_i's utilization U_i to $U(1 - rand^{\frac{1}{n-i}})$
  assign $U = U - U_i$
end for
assign $U_n = U$

tasks, the deadline is randomly selected to be greater than the execution time but less than the period. Other task parameters, such as priority and PPP, are initialized as 0, and we will assign a value to these parameters when we need to use them. For example, we will only assign the PPP when we need to apply the DA-OPA-DP algorithm.

It is worth noting that for computational simplicity, the unit of time is normalized to 1. As a result, all parameters except utilization are integers. This normalization may lead to a discrepancy between the actual and target total utilization of the generated task set, especially when the period range is small. For example, if the utilization of a task τ_i is 0.2, and the period of task τ_i is 37. Then the execution time of task τ_i should be 0.2×37, which is equal to 7.4, but we will assign an integer value 7 to the execution time of task τ_i. This will make the real utilization of task τ_i equal to 7/37, which is around 0.189. However, when the period range is very large, this mismatch becomes negligible.

5.2.2 Result Recording

After generating a task set, we apply either the DA-OPA or DA-OPA-DP algorithm to it. Both algorithms are implemented in the C programming language, following the pseudo code presented in Algorithm 2 and Algorithm 3, respectively.

For Simulation 1, we generate 2000 task sets and apply only the DA-OPA-DP algorithm to each task set. We use a counter variable to track the results: each time the DA-OPA-DP algorithm determines that a task set is schedulable, we increment the counter by one. After completing the test of all 2000 sets, the final value of this counter represents the total number of schedulable task sets identified by the DA-OPA-DP algorithm. Since all task sets are randomly generated, a higher number of schedulable task sets suggests that the algorithm can successfully manage a wider range of scenarios, reflecting better overall schedulability performance. The number of schedulable task sets serves as a direct indicator of the algorithm's effectiveness, and we will use this result to evaluate the performance of the algorithm; the larger the number of schedulable task sets, the more effective the algorithm is. Moreover, in order to show a clear figure, we divide the number of schedulable task sets by 2000 to get the percentage of schedulable task sets as our final result.

For Simulation 2, we again generate 2000 task sets. First, we apply the DA-OPA algorithm to these task sets and record the number of schedulable sets using the same

method as in Simulation 1. Then, we generate another 2000 task sets and apply the DA-OPA-DP algorithm to obtain their corresponding result. The task sets we generate are all random, so we apply each algorithm to different randomly generated task sets to ensure the generality of the results. This approach helps maintain both the fairness and general applicability when we use the result to evaluate the performance of different algorithms. We will show and discuss the results we get for Simulations 1 and 2 in the next chapter.

6

Results

In this chapter, we present the results of simulations 1 and 2, which were introduced in sections 5.1.1 and 5.1.2.

6.1 Simulation 1 Results

For Simulation 1 in section 5.1.1, we aim to determine an effective strategy for assigning priority promotion points (PPPs). Therefore, we will focus on common combinations of input parameters. Specifically, we will evaluate performance across varying utilizations with the following parameter sets: $(N = 10, M = 4, \text{Period} \in [20, 1000])$, $(N = 20, M = 8, \text{Period} \in [20, 1000])$, and $(N = 40, M = 16, \text{Period} \in [20, 1000])$.

As we discussed in section 5.1.1, we will first do the simulation for Heuristic 3 as shown in Equation (5.3).

Figure 6.1 illustrates the percentage of schedulable task sets as a function of total utilization for the input combination $(N = 10, M = 4, \text{Period} \in [20, 1000])$ under Heuristic 3. The blue line represents the schedulable percentage using the DA-OPA algorithm, while the orange line indicates the schedulable percentage using DA-OPA-DP. At low total utilizations, both algorithms achieve a 100% schedulability rate, whereas at very high utilizations, schedulability drops to 0% for both. In the intermediate utilization range, DA-OPA-DP performs similarly to DA-OPA, suggesting that Heuristic 3 is not particularly effective for assigning PPPs.

When we change the inputs to $(N = 20, M = 8, \text{Period} \in [20, 1000])$ and $(N = 40, M = 16, \text{Period} \in [20, 1000])$, the results are shown in Figure 6.2 and Figure 6.3, respectively. The improvement achieved by applying DA-OPA-DP with Heuristic 3 for assigning PPPs remains negligible.

Subsequently, we test Heuristic 4 with varying x values ranging from 2 to 6, as defined by Equation (5.4). We initially test the input set $(N = 10, M = 4, \text{Period} \in [20, 1000])$, starting from $x = 3$ and incrementing by 1 at each step. The results are presented in Figure 6.4. We observed minimal changes across different values of x. Given that setting $x = 2$ makes Heuristic 4 identical to Heuristic 3, the improvement achieved by varying x remains negligible. When we change the inputs to $(N = 20, M = 8, Period \in [20, 1000])$ and $(N = 40, M = 16, Period \in [20, 1000])$, the result shown in Figure 6.5 and Figure 6.6 respectively. The percentage of the

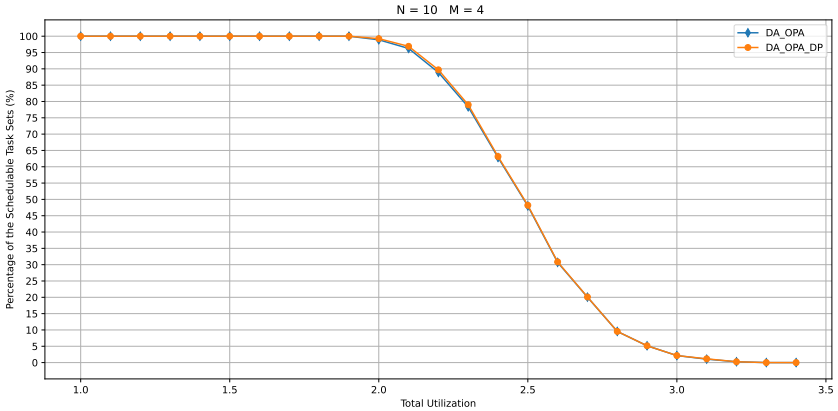

Figure 6.1: Percentage of the schedulable task sets against the total utilization when N = 10 and M = 4, applying Heuristic 3, compare the performance of DA-OPA and DA-OPA-DP algorithms

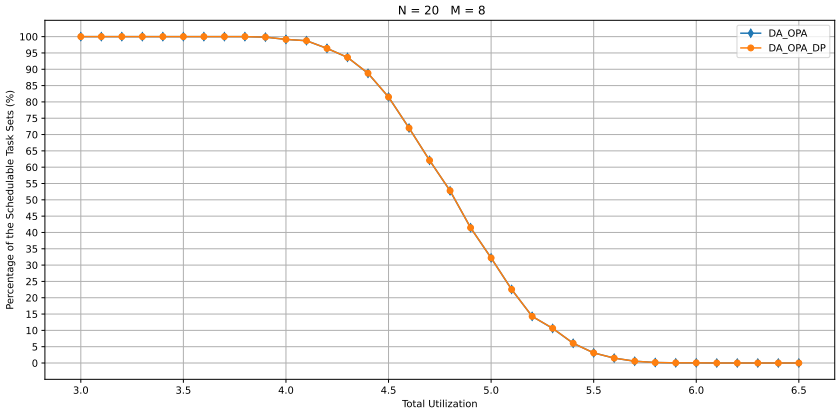

Figure 6.2: Percentage of the schedulable task sets against the total utilization when N = 20 and M = 8, applying Heuristic 3, compare the performance of DA-OPA and DA-OPA-DP algorithms

schedulable task sets still has minimal changes. This indicates that Heuristic 4 is still not particularly effective for assigning PPPs.

When we consider the remaining number of tasks in the heuristic, as we discussed in section 5.1.1, we get Heuristic 5 as shown in Equation (5.5).

As depicted in Figures 6.7 and 6.8, for the input $(N = 10, M = 4, Period \in [20, 1000])$ and$(N = 20, M = 8, Period \in [20, 1000])$ at medium total utilization levels, the DA-OPA-DP algorithm consistently achieves an improvement of approximately 5% or greater compared to DA-OPA algorithm. Furthermore, the performance of both algorithms remains equivalent at very low or high utilization levels.

What's more, as shown in Figure 6.9 for the input $(N = 40, M = 16, Period \in [20, 1000])$, the improvement of the DA-OPA-DP algorithm over the DA-OPA algorithm is less than the result of $(N = 10, M = 4, Period \in [20, 1000])$ and$(N = 20, M = 8, Period \in [20, 1000])$ cases. This indicates that Heuristic 5 still has potential to be improved, which will be discussed further in Chapter 7.

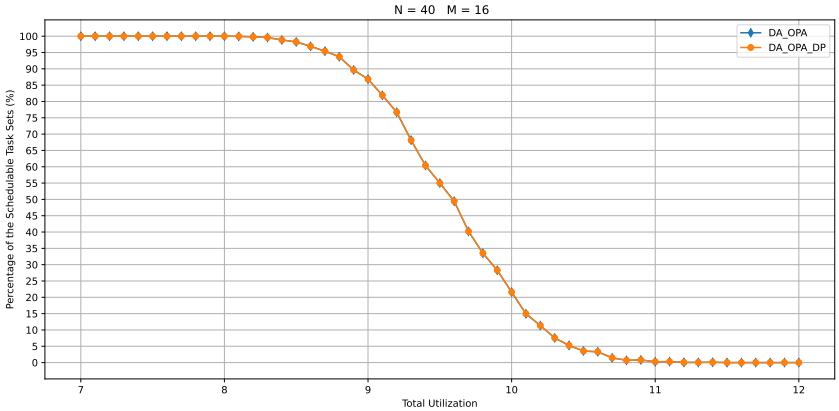

Figure 6.3: Percentage of the schedulable task sets against the total utilization when N = 40 and M = 16, applying Heuristic 3, compare the performance of DA-OPA and DA-OPA-DP algorithms

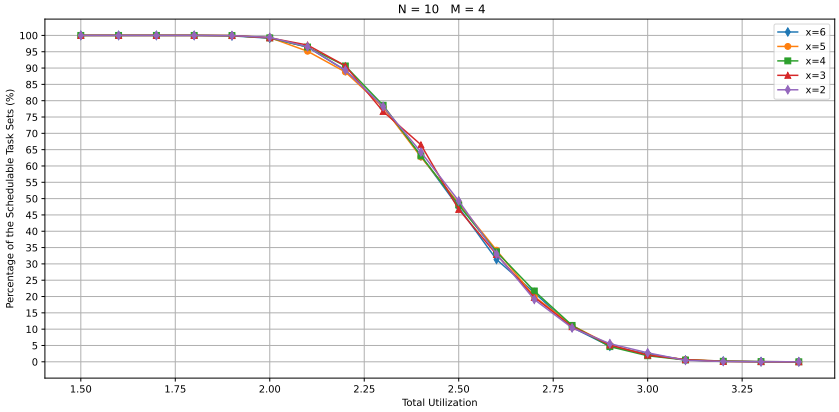

Figure 6.4: Percentage of the schedulable task sets against the total utilization when N = 10 and M = 4, applying Heuristic 4, compare the performance of DA-OPA-DP via different values of x in Heuristic 4

6.2 Simulation 2 Results

Since the result for Heuristic 5, as shown in Figures 6.7, 6.8, and 6.9, already shows the significant improvement that DA-OPA-DP with a good strategy to assign the PPP. We then do simulation 2, which is discussed in section 5.1.2, to investigate the influence of the different inputs.

First, we change the number of tasks for the same number of processors to investigate the influence of the number of tasks in the task set. As shown in Figure 6.10, we set the number of processors M equal to 4, and change the number of tasks N to 10, 15, and 25. When N increases, the performance of DA-OPA and DA-OPA-DP both decreases. Also, the difference between DA-OPA and DA-OPA-DP decreases too, which indicates that the improvement of DA-OPA-DP compared with the DA-OPA is decreasing while the number of tasks increases when the number of processors is fixed. This may be because each task's interference will have a pessimistic workload,

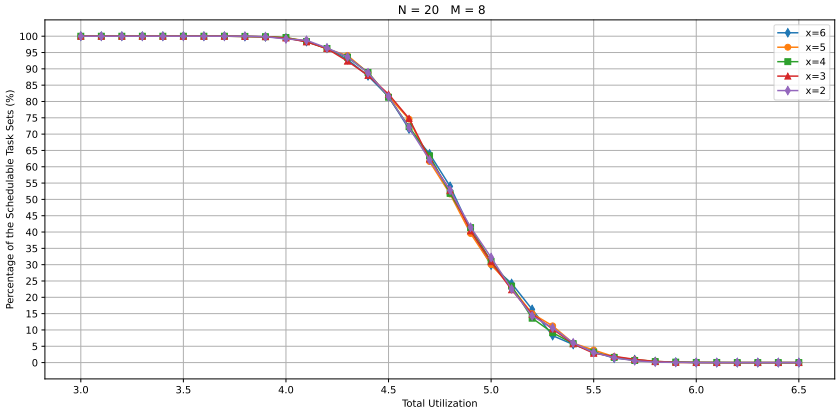

Figure 6.5: Percentage of the schedulable task sets against the total utilization when N = 20 and M = 8, applying Heuristic 4, compare the performance of DA-OPA-DP via different values of x in Heuristic 4

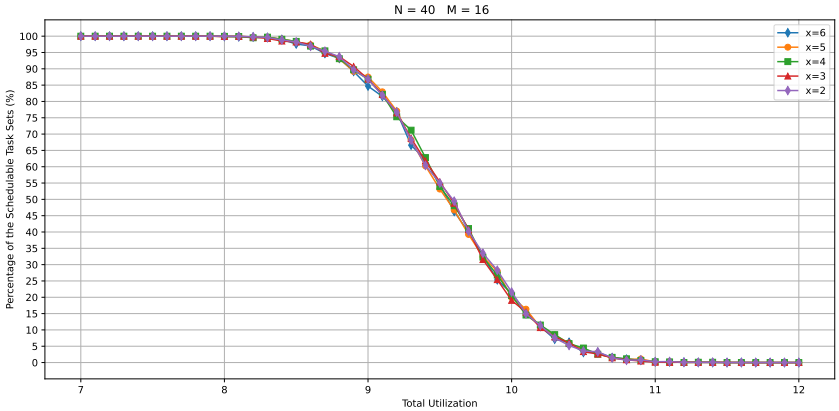

Figure 6.6: Percentage of the schedulable task sets against the total utilization when N = 40 and M = 16, applying Heuristic 4, compare the performance of DA-OPA-DP via different values of x in Heuristic 4

and such pessimism increases as N increases.

The same situation happens when we change M to 8, and apply different N equal to 15, 20, and 30, respectively, as shown in Figure 6.11. Also, as shown in Figure 6.12, when M is equal to 16, and N changes to 15, 20, and 30, the performance of DA-OPA and DA-OPA-DP, and the difference between DA-OPA and DA-OPA-DP both decrease.

Then we investigate the influence of the number of processors. We keep N equal to 20, and change M to 4, 8, and 12, as shown in Figure 6.13. It is noticeable that when M increases and is close to the value of N, the improvement of the DA-OPA-DP compared with the DA-OPA also increases. This may be because when the difference between M and N is small, the average utilization/load will be close to 1, which results in a high utilization for each task. These situations will make every task a high load, and it is difficult to schedule all the tasks under GFP scheduling. But, GDP scheduling can solve this problem, which has significantly better performance

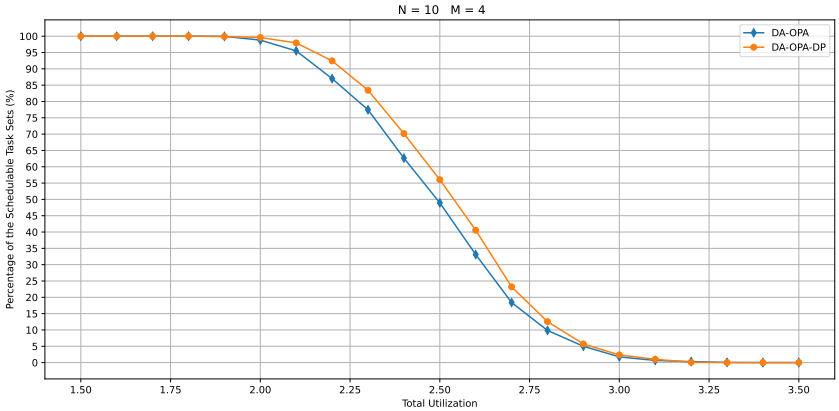

Figure 6.7: Percentage of the schedulable task sets against the total utilization when N=10, M=4, applying Heuristic 5, compare the performance of DA-OPA and DA-OPA-DP algorithms

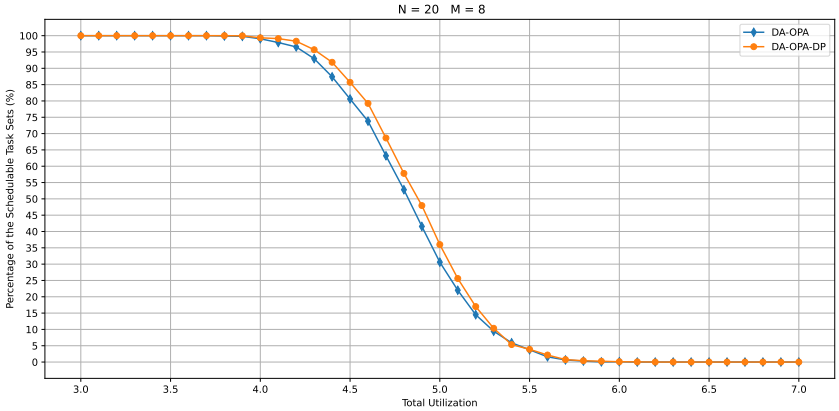

Figure 6.8: Percentage of the schedulable task sets against the total utilization when N=20, M=8, applying Heuristic 5, compare the performance of DA-OPA and DA-OPA-DP algorithms

than GFP scheduling. This improvement will be discussed further in Chapter 7.

We have the same situation when we keep N equal to 40, and change M to 8, 12, and 16, which is shown in Figure 6.14.

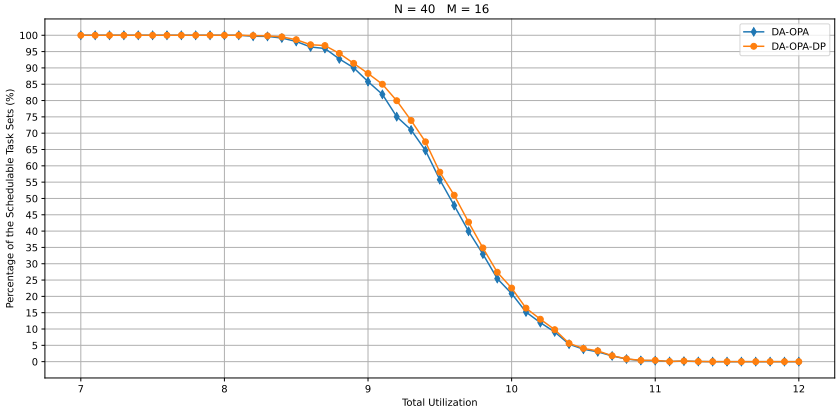

Figure 6.9: Percentage of the schedulable task sets against the total utilization when N=40, M=16, applying Heuristic 5, compare the performance of DA-OPA and DA-OPA-DP algorithms

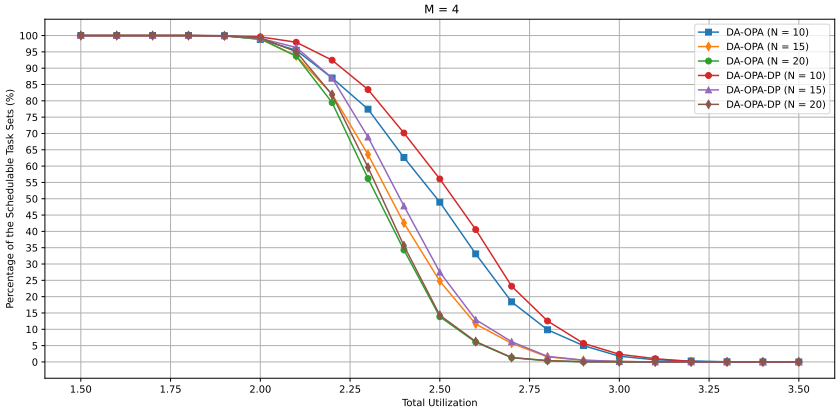

Figure 6.10: Percentage of the schedulable task sets against the total utilization when M=4, applying Heuristic 5, compare the performance of DA-OPA and DA-OPA-DP algorithms via different numbers of tasks N

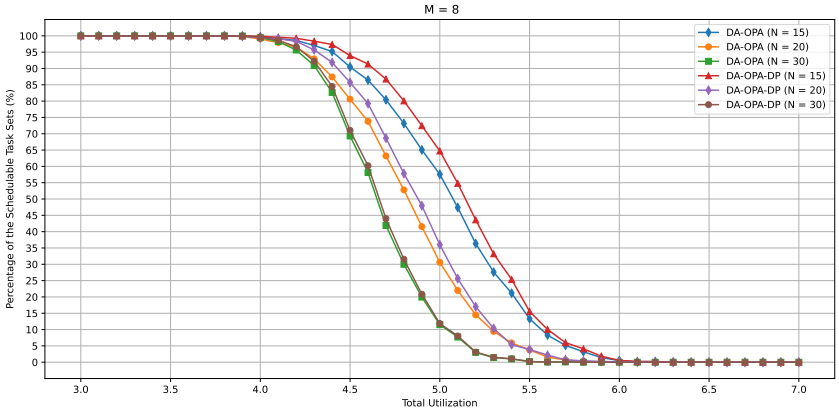

Figure 6.11: Percentage of the schedulable task sets against the total utilization when M=8, applying Heuristic 5, compare the performance of DA-OPA and DA-OPA-DP algorithms via different numbers of tasks N

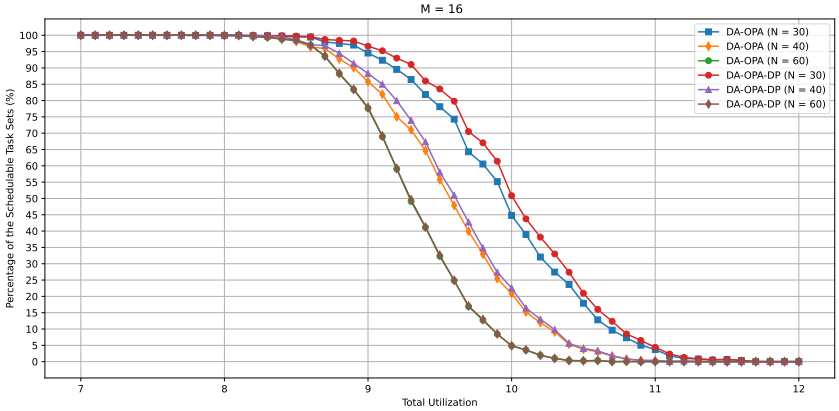

Figure 6.12: Percentage of the schedulable task sets against the total utilization when M=16, applying Heuristic 5, compare the performance of DA-OPA and DA-OPA-DP algorithms via different numbers of tasks N

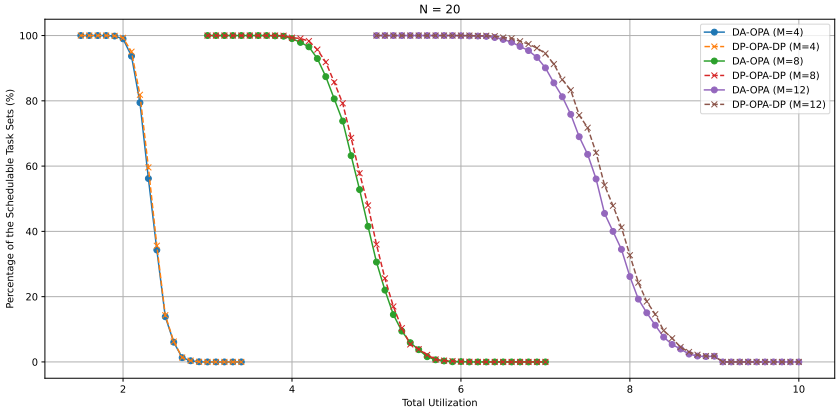

Figure 6.13: Percentage of the schedulable task sets against the total utilization when N=20, applying Heuristic 5, compare the performance of DA-OPA and DA-OPA-DP algorithms via different numbers of processors M

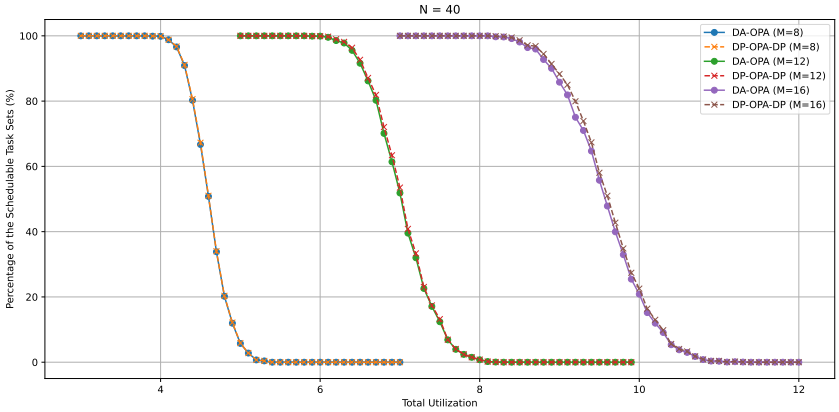

Figure 6.14: Percentage of the schedulable task sets against the total utilization when N=40, applying Heuristic 5, compare the performance of DA-OPA and DA-OPA-DP algorithms via different numbers of processors M

7

Discussion

In this chapter, we will discuss areas where this thesis is worth expanding and possible improvements.

7.1 Priority Promotion Point Assignment Strategy

As previously discussed, the assignment of PPP significantly affects the performance of the DA-OPA-DP algorithm. In many cases, especially when the number of tasks is large, the pass ratio for dual-priority scheduling on multiprocessor systems is not very high. Under traditional fixed-priority global scheduling, lower-priority tasks frequently miss deadlines due to preemption by higher-priority tasks. After introducing a priority promotion mechanism, tasks with higher utilization that promote earlier may preempt other tasks with originally higher priorities but relatively later/longer promotion times, potentially causing those higher-priority tasks to miss their deadlines. This issue becomes especially pronounced as the number of tasks increases. Consequently, it becomes challenging to identify a single, universally effective method to assign promotion points to all tasks. A feasible alternative is to apply different strategies for assigning priority promotion points based on task-specific parameters such as utilization or deadlines.

Ideally, the most effective approach would involve an adaptive strategy that dynamically adjusts promotion points based on runtime task behavior and system state. However, such dynamic adaptation would prevent predicting priority promotion points beforehand, and we need to know the priority promotion points to apply the schedulability test to know whether the task is schedulable at the assumed priority level. Thus, such dynamic adaptation will lead to unknown schedulability when a task is assigned to a priority level during the OPA algorithm, making it impossible to employ the OPA algorithm directly for assigning task priorities. Thus, priority assignment itself becomes an additional challenge that needs further investigation.

7.2 The Impact of Input on Performance

As shown in section 6.2, when we change N and M separately, the decrease of N and the increase of M will both make the DA-OPA-DP algorithm perform better. Besides the reason we discussed in section 6.2 that pessimism of the task's workload

will be increased when N increases, the difference between N and M can be another reason.

As it can be noticed from Figure 6.10, Figure 6.11, and Figure 6.12, the input with best improvement is $(M = 4, N = 10)$, $(M = 8, N = 15)$ and $(M = 16, N = 30)$. We can use $\frac{M}{N}$ to indicate the average utilization for each task. Then, in Figure 6.10, the performance of DA-OPA-DP when the input is $(M = 4, N = 10)$ is better than the input is $(M = 4, N = 15)$ and $(M = 4, N = 20)$. If we consider these three cases: $(M = 4, N = 10)$, $(M = 4, N = 15)$, and $(M = 4, N = 20)$, the results of these three cases are shown in Figure 6.10. Case $(M = 4, N = 10)$ will have the largest value of $\frac{M}{N}$, and also the DA-OPA-DP algorithm has the best performance in this case. If we compare these three cases, the larger the value of $\frac{M}{N}$, the better performance the DA-OPA-DP algorithm has. A similar trend is evident in the results presented in Figure 6.11 and Figure 6.12.

What's more, we can also compare the results between different inputs with both M and N changes. For example, when the input is $(M = 4, N = 10)$, the result is shown in the red and blue lines in Figure 6.10, $\frac{M}{N}$ is equal to 0.4, the task sets' schedulable percentage of the DA-OPA-DP algorithm is about 5% higher than that of the DA-OPA algorithm in the middle part. When the input is $(M = 8, N = 15)$, the result is shown in the red and blue lines in Figure 6.11, the value of $\frac{M}{N}$ is around 0.53 which is larger than the value of $(M = 4, N = 10)$ case, the improvement of the percentage is grown to around 7%. This further illustrates that $\frac{M}{N}$ has an impact on the performance of the DA-OPA-DP algorithm.

It seems that the higher the value of $\frac{M}{N}$ is, the more improvement that DA-OPA-DP has, compared with the DA-OPA. Additionally, the higher the value of $\frac{M}{N}$ means the higher the average utilization for each task. This also shows the potential that the priority promotion mechanism has when the load is high in global scheduling. When the load of each task is high, the schedule of the task will become difficult, and the classical GFP priority scheduling cannot handle it, but by introducing the priority promotion mechanism can have a big potential to schedule these difficult task sets.

7.3 Future Improvement

Several promising research directions arise from this study, as discussed below:

Improving the Accuracy of the DA-OPA-DP Test: Although the proposed DA-OPA-DP algorithm demonstrates approximately a 5% improvement over the existing DA-OPA test, the theoretical potential gains from dual-priority scheduling mechanisms may be much greater. Future research could explore methods to further reduce pessimism, including more accurate modeling of carry-in interference and heuristic approaches for assigning PPP.

Analyzing Effective Application of Dual-Priority Scheduling in Real-Time Systems: While this thesis mainly focuses on schedulability analysis and theoretical aspects, practical considerations such as inter-processor synchronization and task migration overhead are not thoroughly examined. However, the increased dynamic

scheduling behavior introduced by dual-priority mechanisms inherently increases task migration costs. Therefore, further research is necessary to analyze and mitigate task migration overhead associated with the DA-OPA-DP algorithm.

Limited the Carry-in and Carry-out Interference: The DA-LC schedulability test in study [1] considers only carry-in interference, and the "LC" in DA-LC means that it considers the number of processors in the schedulability test to limit carry-in interference, which is the interference caused by a job that was released before the start of the scheduling window. This can also be considered in our schedulability test. In the previously introduced DA-OPA-DP algorithm, carry-in and carry-out interference from all other tasks is fully accounted for in the total interference. However, in reality, only as many tasks as the number of processors can run concurrently, implying that at most $M - 1$ tasks could realistically cause carry-in and carry-out interference for a specific task. To reduce this pessimism further, we can adopt the approach described by the DA-LC schedulability test in reference [1], which explicitly limits carry-in and carry-out interference.

The specific approach to limit the carry-in interference is as follows: Calculate the individual carry interference of each task toward the tested task, then select and sum only the largest $M-1$ interference values as the total carry interference. Adding this total to the non-carry interference will yield a more accurate and less pessimistic schedulability test.

Improving the task generation algorithm: In the task generator, the utilization allocation algorithm is the UUnifast-discard algorithm proposed in [1]. It should be noted that the UUnifast-discard algorithm requires restricting the number of processors and tasks to maintain task-set validity, particularly when discarding up to 1000 invalid sets (Details shown in Figure A.1 from [1]). It means that when the input number of tasks and the number of processors are very close, the task set generated by the UUnifast-discard algorithm may be invalid. For example, when we use $(M = 3, N = 4)$ as the input, most of the time, there will be task's utilization will be higher than 1. Then, each time we detect that the task's utilization is higher than 1, we will discard this task set and generate a new task set. If the new task set still has this problem. We will generate again until we discard 1000 times. Since we generate the task set randomly, sometimes the task set still has tasks with utilization higher than 1 even after we re-generate for 1000 times. This happens when the value of M is close to N. According to [1]), if we want to avoid this problem, we need to follow the constraint shown in Figure A.1 to set the input. Thus, we have not tested the performance of the DA-OPA-DP for some inputs with a small difference of M and N. But it is worth testing the performance of the DA-OPA-DP under the small difference of M and N. Further work can be done to improve the utilization allocation algorithm for generating the task sets, and test the DA-OPA-DP algorithm when using inputs has a small difference of M and N, such as $(M = 3, N = 4)$, $(M = 4, N = 6)$, and so on.

7.4 Ethical Consideration

From a societal perspective, this research aims to improve the performance and reliability of real-time applications in critical domains such as autonomous vehicles, industrial automation, and medical systems. Enhancing system responsiveness and predictability contributes to safer and more efficient services, benefiting a broader user base. It is crucial to ensure that these technologies remain accessible to all users, regardless of socioeconomic status.

Ecologically, the proposed scheduling method is designed to minimize context switching, thereby improving processor efficiency and reducing unnecessary resource consumption while maintaining system schedulability. This can lead to reduced energy consumption, particularly in embedded and battery systems. By advocating energy-efficient computing, this research contributes to the development of sustainable computing infrastructures and reduces the environmental impact associated with large-scale real-time implementations.

Ethically, the design and implementation of scheduling algorithms should prioritize system safety and fairness. Real-time decisions can have significant implications for critical systems such as traffic control or medical devices, thus making correctness and robustness essential. Transparent documentation of algorithmic limitations and rigorous testing under various conditions are necessary to prevent unintended consequences. Additionally, in systems managing sensitive data, it is important that real-time mechanisms do not compromise privacy or data integrity.

8

Conclusion

In this thesis, we analyzed the potential of dual-priority scheduling in multiprocessor systems and proposed a deadline analysis for dual-priority (DA-DP) schedulability test specifically designed for multiprocessor dual-priority scheduling. We first propose an approach with excessive pessimism to calculate the workload for the schedulability test. After identifying and analyzing the pessimism, we developed an improved approach to calculate the workload. Furthermore, we combined DA-DP with the existing deadline analysis with optimal priority assignment (DA-OPA) algorithm, proposing a hybrid DA-OPA with dual priority (DA-OPA-DP) algorithm. This combined approach achieves roughly a 5% improvement over the existing DA-OPA algorithm, enabling more precise predictions of task set schedulability.

Our exploration of various priority promotion assignment strategies revealed that assigning priority promotion points based on task utilization provides effective results, though alternative strategies may further enhance system performance. Additionally, addressing limited carry-in interference through refined analytical methods represents another valuable path toward reducing pessimism in schedulability tests.

Overall, the methods and insights presented in this thesis not only enrich existing real-time scheduling theory but also offer practical contributions toward the effective application and further development of dual-priority scheduling mechanisms in multiprocessor environments.

Bibliography

[1] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, Oct. 2011. [Online]. Available: https://doi.org/10.1145/1978802.1978814

[2] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, ser. Mathematical Sciences Series. Freeman, 1979. [Online]. Available: https://books.google.se/books?id=fjxGAQAAIAAJ

[3] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Operations Research*, vol. 26, no. 1, pp. 127–140, 1978. [Online]. Available: http://www.jstor.org/stable/169896

[4] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Systems*, vol. 47, no. 1, pp. 1–40, 2011. [Online]. Available: https://doi.org/10.1007/s11241-010-9106-5

[5] B. Andersson and J. Jonsson, "The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%," in *15th Euromicro Conference on Real-Time Systems, 2003. Proceedings.*, 2003, pp. 33–40.

[6] A. Burns and A. Wellings, "Dual priority assignment: A practical method for increasing processor utilisation," in *Fifth Euromicro Workshop on Real-Time Systems*, 1993, pp. 48–53.

[7] A. Burns, "Dual priority scheduling: Is the processor utilisation bound 100%?" in *Proceedings of the 1st International Real-Time Scheduling Open Problems Seminar (RTSOPS 2010)*, R. I. Davis and N. Fisher, Eds., 2010, pp. 3–4.

[8] B. Andersson and J. Jonsson, "Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition," in *Proceedings Seventh International Conference on Real-Time Computing Systems and Applications*, 2000, pp. 337–346.

[9] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Department of Computer Science, University of York, Tech. Rep. YCS-164, 1991.

[10] R. M. Pathan, "Unifying fixed- and dynamic-priority scheduling based on priority promotion and an improved ready queue management technique," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015, pp. 209–220.

[11] X. Gu, A. Easwaran, and R. Pathan, "Design and analysis for dual priority scheduling," in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, 2018, pp. 164–173.

[12] T. Baker, "Multiprocessor EDF and deadline monotonic schedulability analysis," in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, 2003, pp. 120–129.

[13] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Oper. Res.*, vol. 26, no. 1, p. 127–140, Feb. 1978. [Online]. Available: https://doi.org/10.1287/opre.26.1.127

[14] C. L. Liu and J. W. Layland, *Scheduling algorithms for multiprogramming in a hard-real-time environment.* USA: Kluwer Academic Publishers, 2001, p. 179–194.

[15] D.-I. Oh and T. P. Bakker, "Utilization bounds for n-processor rate monotonescheduling with static processor assignment," *Real-Time Syst.*, vol. 15, no. 2, p. 183–192, Sep. 1998. [Online]. Available: https://doi.org/10.1023/A:1008098013753

[16] N. Guan, M. Stigge, W. Yi, and G. Yu, "Fixed-priority multiprocessor scheduling with liu and layland's utilization bound," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010, pp. 165–174.

[17] Y. Sun, G. Lipari, N. Guan, and W. Yi, "Improving the response time analysis of global fixed-priority multiprocessor scheduling," in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, 2014, pp. 1–9.

[18] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 149–160.

[19] A. Burns and A. Wellings, "Dual priority assignment: A practical method for increasing processor utilisation," in *Fifth Euromicro Workshop on Real-Time Systems*, 1993, pp. 48–53.

[20] F. Fauberteau, S. Midonnet, and L. George, "Improvement of schedulability bound by task splitting in partitioning scheduling," *1st International Real-Time Scheduling Open Problems Seminar (RTSOPS'10)*, 07 2010.

[21] R. M. Pathan, "Unifying fixed- and dynamic-priority scheduling based on priority promotion and an improved ready queue management technique," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015, pp. 209–220.

[22] X. Gu, A. Easwaran, and R. Pathan, "Design and analysis for dual priority scheduling," in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, 2018, pp. 164–173.

[23] J. Xiao, S. Altmeyer, and A. D. Pimentel, "Schedulability analysis of global scheduling for multicore systems with shared caches," *IEEE Transactions on Computers*, vol. 69, no. 10, pp. 1487–1499, 2020.

[24] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of edf on multiprocessor platforms," in *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, 2005, pp. 209–218.

A

Appendix 1

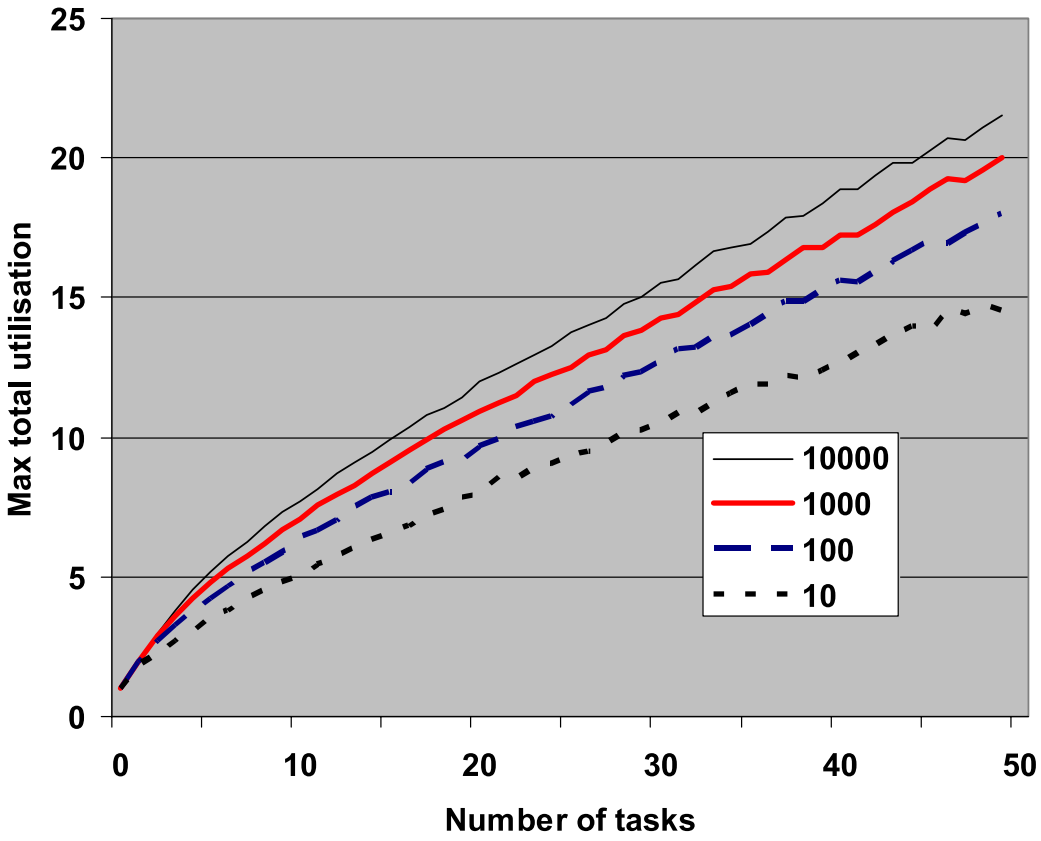

Figure A.1: Utilisation limits for the UUnifast-Discard algorithm as a function of taskset cardinality and the discard limit used from [1]

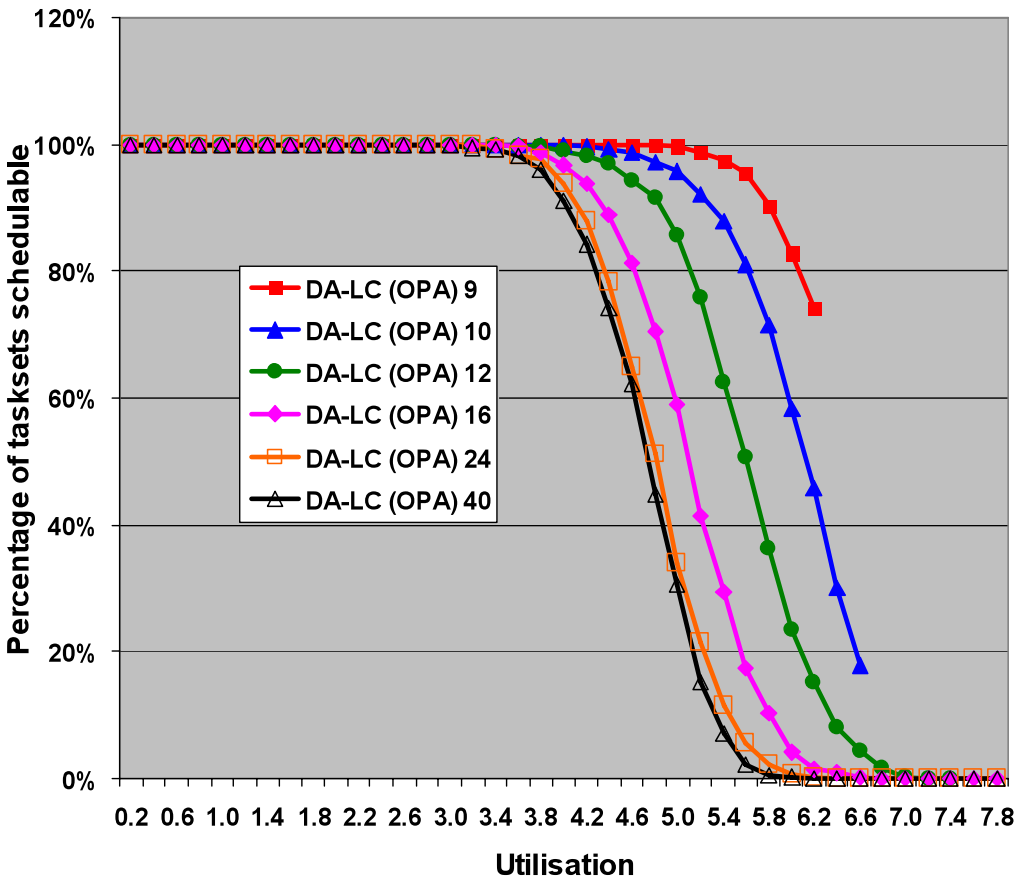

Figure A.2: Taskset cardinality from 9 to 40 from [1]