CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Federated Scheduling of Mixed-Criticality Sporadic DAG Tasks on Uniform Multiprocessors

Master's thesis in Computer science and engineering

Chengzi Huang

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Federated Scheduling of Mixed-Criticality Sporadic DAG Tasks on Uniform Multiprocessors

Chengzi Huang

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Federated scheduling of Mixed-Criticality Sporadic DAG Tasks on Uniform Multiprocessors

Chengzi Huang

Supervisor: Risat Pathan
Examiner: Jan Jonsson

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Federated Scheduling of Mixed-Criticality Sporadic DAG Tasks on Uniform Multiprocessors
Chengzi Huang
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

In designing real-time systems, there is an emerging trend in moving towards the mixed-criticality(MC) system, where functionalities with different degrees of importance (i.e., criticality) are implemented upon a shared platform, and the level of heterogeneity in the modern multiprocessors systems is gradually increasing. This thesis develops algorithms to schedule and allocate implicit-deadline sporadic mixed-criticality DAGs upon the uniform heterogeneous multiprocessors.

A two-level scheduler is designed based on the federated scheduling paradigm. Tasks are categorized into heavy and light tasks according to their utilization. Each heavy task exclusively executes on a number of dedicated processors (cluster). Light tasks are treated as sequential tasks and share the remaining processors. The work-conserving scheduler is used at the cluster level, and EDF-VD is used to schedule the light tasks. The upper bound of the response time for a heavy task under the work-conserving scheduler and utilisation bound for light tasks under EDF-VD are proposed to verify offline that the design constraints are met.

Task allocation upon the multiprocessors is known to be NP-Hard. This thesis describes an approach to solving the task allocation problem using bin-packing heuristics and simulated annealing. There are two stages for task allocation. The light tasks are assigned to processors using partitioned scheduling in the first stage. A group of bin-packing heuristics will be considered, and a metric QoP is defined to compare the quality of partitioned scheduling under different heuristics. The allocation partition with the best QoP will be used. In the second stage, simulated annealing is employed and tries to find a feasible solution by gradually minimizing the total task lateness in the system.

There is a service abrupt problem at the traditional mixed criticality system. Elastic mixed criticality task model is introduced to address this problem. This thesis also develop schedulability test and discusses the task allocation for elastic mixed criticality task.

Empirical evaluation is presented to show the effectiveness of our approach.

Keywords: Federated scheduling, Task allocation, Partitioned scheduling, Simulated annealing

Acknowledgements

First of all, I would like to give my heartfelt thanks to my supervisor Risat Pathan, for his invaluable instruction and inspiration. Without him, this thesis could not have been finished. Furthermore, I am grateful to my examiner Jan Jonsson, for piquing my interest in this topic through his excellent real-time systems courses. Finally, I would like to thank my opponents David Wilkins and Oskar Hammargren, for reviewing this thesis and giving valuable writing suggestions.

Chengzi Huang, Gothenburg, June 2022

Contents

List of Figures

List of Algorithms

1
Introduction

Mixed Criticality (MC) systems are systems that have components of two or more distinct criticality levels (for example safety-critical, mission-critical and low-critical) [1]. Criticality is a designation of the level of assurance needed against failure for a system component. A commonly used example of an MC system is an unmanned aerial vehicle. The electronic system consists of flight control system, camera control system and route planning system. The flight control system is a safety-critical system. When it fails, the unmanned aerial vehicle will crash-land. This is dangerous. Failures of safety-critical system are not acceptable. By contrast, the camera control system that captures videos and process images is a mission-critical system. Route planner, is desirable and improves the quality of service of the system, but is less critical. Those systems are needed to be hosted on the same platform by the system designers. An increasingly important trend in the design of real-time embedded systems is the integration of components with different levels of criticality onto a common computing platform. Integrating subsystems with different criticality requirements into one device that will reduce hardware costs, energy consumption and weight.

Many modern multiprocessors now have heterogeneous processing cores (e.g., ARM's big.LITTLE platform) that can execute different tasks at different speeds. Heterogeneous multiprocessors have continued to improve to meet today's requirement for high computation power in those complex embedded real-time systems [2]. In addition, applications that are implemented using a task-based parallel programming model such as MPI can effectively exploit the processing capacity of a parallel heterogeneous platform.

Scheduling algorithms play a important role in guaranteeing time predictability, i.e., computing the makespan of parallel applications. It is important to design effective scheduling algorithms for parallel tasks on a heterogeneous platform and propose schedulability tests to verify offline that the design constraints are met. Unfortunately, many of the well-known schedulability analysis techniques for homogeneous multiprocessors cannot be trivially applied to heterogeneous multiprocessors [3]. The task assignment is also a vital issue for guaranteeing system correctness. Since the problem of assigning tasks to the processors (even for sequential tasks) is NP-hard in the strong sense [4], designing an effective task assignment algorithm is not only important but also more challenging for parallel tasks in comparison to sequential tasks.

In this thesis, we study the problem of scheduling parallel mixed-criticality tasks on a uniform multiprocessors platform. Although there has been extensive research on the two related problems, namely, mixed-criticality scheduling of sequential tasks

on the homogeneous multiprocessors and single-criticality scheduling of parallel tasks on the heterogeneous multiprocessors, to our knowledge, there has been little prior work on the combined problem of mixed-criticality scheduling of parallel task on the heterogeneous platform [1, 5]. The organization and contributions of this thesis is as follow:

- Chapter 2 presents the real-time workload and platform models considered in this thesis and provides necessary background information on real-time scheduling theory as well as local search.
- Chapter 3 defines the mixed-criticality parallel real-time task model and uniform heterogeneous multiprocessor platform used in this thesis.
- Chapter
- Chapter 4 presents a two-level scheduler based on the federated scheduling paradigm. The work-conversing algorithm and a modified earliest deadline first algorithm, called EDF-VD, are used at the different levels.
- Chapter 5 develops the schedulability test for the work-conserving scheduling algorithm and EDF-VD to guarantee the system correctness on the uniform platform.
- Chapter 6 presents a practical method to find a feasible task allocation based on bin packing heuristics and local search.
- Chapter 7 develops the schedulability test under elastic mixed criticality system and the task allocation under this system.
- Chapter 8 evaluates the performance of the proposed scheduling algorithm.
- Chapter 9 contains the conclusion and some further discussion.

2
Background

This chapter presents the fundamental background to the field of real-time systems and local search.

2.1 Real time systems

In general, *real time systems* are systems in which the timing of a computation is important for the correctness of the system. Independent programs, in the field of real-time systems, are known as *tasks*. Every task τ_i is included in a *task set* with n tasks such that $\{ \tau_1, \tau_2, ..., \tau_n \}$. For each task τ_i, the worst-case execution time (WCET) C_i is known. And each task is said to *arrive* (i.e. ready to execute) at a certain instant in time. Tasks may arrive several times where each instance is know as a *job*. If the arrivals have a constant time between each job the task is said to be *periodic* and the time between consecutive arrivals of a task is called the *period* T_i. However some task's jobs may not arrive strictly periodically, but some time after the period. These tasks are called *sporadic*. Each task also has a *deadline* D_i which specifies the latest time, relative to the arrival time, that the job must complete. If the deadline is equal to the period it is said to be an *implicit deadline*.

2.2 Scheduling algorithm and task priority

For priority based scheduling algorithms, task *priorities* are used to determine in what order tasks should be executed so that all tasks meet their deadline. The priorities are either fixed or dynamic depending on what scheduling algorithm is used. A fixed priority means that the priority is decided before run-time, while for dynamic priorities the priority of a task can change during run-time.

The priority of a task also indicates how important it is that the task executed when it is dispatched. In some cases, where preemptive scheduling is used, a task with higher priority can preempt another task that already has started its execution. However, there are also cases where non-preemptive scheduling is used, which means that when a task has started its execution, it will execute until completion. Three different task priority assignment policies are presented below.

- **EDF**: The earliest deadline first (EDF) priority policy dynamically changes the priority of tasks during run-time. Task priority in EDF is decided by when in time an instance of a task has its deadline, the task with the deadline closest in time will have the highest priority while the task with the furthest deadline will have the lowest priority.

- **DM**: Deadline monotonic (DM) is a priority policy which uses static priority. For the DM policy the priorities are assigned as follows: The task with the lowest relative deadline has the highest priority and the task with the highest deadline has the lowest priority.
- **RM**: Rate monotonic (RM) is a priority policy which uses static priorities. For RM, priorities depend on the periods of the tasks. The task with the lowest period has the highest priority, and the task with highest period has the lowest priority in the task set.

2.3 Schedulability test

Schedulability tests help determine if a task set is schedulable or not, given a scheduling algorithm. A Schedulability test can be *sufficient*, *necessary*, or both, and it produces a binary outcome (positive or negative). A positive outcome from a sufficient test means that the task set is definitely schedulable and a negative outcome means nothing. Conversely, for a necessary test the positive outcome means nothing but a negative outcome means the task set is definitely not schedulable. If a Schedulability test is both sufficient and necessary, the test is said to be *exact*. A positive outcome from an exact test means that the task set is definitely schedulable and a negative outcome means that it is definitely not schedulable.

There are many existing techniques that can be used to perform Schedulability testing. Two of them are *Response time analysis* and *Guarantee bound analysis*

- Response time analysis (RTA) is a widely used Schedulability test. RTA tries to find the worst-case response time for a task, where the response time is the end time of the task while it is being interrupted by higher priority tasks. Since response time analysis will give the worst-case execution pattern for a task the result is then compared to the deadline of the task. If the result from RTA is lower than the deadline the analysis will pass for this task and it is scheduable, otherwise it is not scheduable.
- Guarantee Bound Analysis is another method to validate that all timing constraints will be met in a system. A task's utilization is expressed as C_i/T_i and if the accumulated utilization of all tasks in the system does not exceed the guarantee bound, we can guarantee that all timing constraints in the system will be met. The guarantee bound differs depending on what task priority assignment policy that is used. For example the guarantee bound for EDF is $U_{EDF} = 1$, while for RM the guarantee bound is $U_{RM} = n(2^{1/n} - 1)$, where n is the number of tasks [6].

2.4 Computing platform and workload model

The computing platform is a important part for modeling a real-time system. The worst case execution time(WCET) is related with the computing with the computing platform. It is commonly assumed in real-time systems research that a processor runs at a fixed (normalized) speed of 1. Then, the worst-case execution of a task on the unit speed is referred to as the workload of the task.

Figure 2.1 shows the execution pattern of a task τ_1, which has the workload $C_1 = 2$, deadline $D_1 = 3$ and period $P_1 = 4$. At the time interval $[0, 5]$ the processor runs with unit speed. The speed of processor changes to 0.5 at time instant $t = 5$. The height of jobs indicates the execution speed at the moment. Under this case, the total execution time for the first job $\tau_{1,1}$ and the second job $\tau_{1,2}$ is 2 and 3, respectively. However the third job need 4 unit time to finish its work. That results in a deadline miss at $t = 11$.

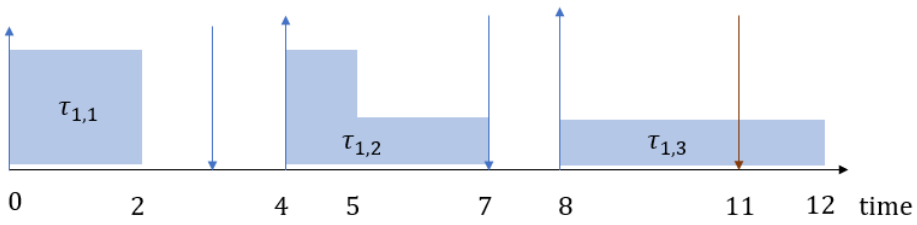

Figure 2.1: Varying execution lengths of the same task under different platform speeds

A multiprocessor is a combination of multiple uniprocessors, and can be classified into one of the following platform models depending upon the relationship between the computing capacities of those processors:

- Identical: all processors have same speed. The WCET of a task is same for all processors.
- Uniform: each processor is characterized by its own execution speed. The WCET of a task is on the processor the product of its workload (same for all processors) and the processor speed (may differ for each processor).
- Unrelated:The WCET is not necessarily related for different processors.

In this thesis, the uniform multiprocessor platform is focused.

Algorithm 1: Local_search(f, N, L, S)

Input: f: a objective function that we want to maximize/minimize.
N: a function that can generate the neighborhood for a given state s.
L: a function that get a set of legal neighbors for a given state.
S: a function that select one of the neighbors.

```
s = Generate_Initial_Solution();
s* = s;
for k = 1 to maxTrial do
    if satisfiable(s) and f(s*) is better then
        s = s*;
    end
    s* = S(L(N(s), s), s);
end
return s;
```

2.5 Local search and simulated annealing

Local search is a heuristic method for solving NP-Complete problem. Local search is useful for the problems that want to find a solution which maximize/minimum a criterion among the candidate solutions. Local search algorithms move from one solution to another solution in the searching space by applying local changes, until a optimal solution is found or upper bound of search time is reached.

A neighborhood is the set of all the solutions which can be obtained by applying minimal change on the current solution. For example, a neighborhood can be defined as one node differing from another solution for the vertex cover problem [7]. The pseudo code for a basic local search is shown as algorithm 1.

Simulated annealing is a kind of local search algorithm. It is a global optimization technique which borrows ideas from statistical physics. It has a probability to accept a worse solution when doing the local search. Accepting worse solutions makes the simulated annealing have the ability for jumping out of the local optimal area [8]. When using simulated annealing, there is a temperature parameter, which decreases as the number of searches increases. This parameter controls the probability to accept a worse solution. The pseudo code for simulated annealing is shown as algorithm 2.

Algorithm 2: simulated_annealing(f, N, L, S)

Input: E: energy (goal) function we want to maximize/minimize.
Neighbor: a function that can generate the a neighbor.
Temperature: a function that get current temperature.
P: a function that return the acceptance probability.

```
1  s = Generate_Initial_Solution();
2  s* = s;
3  Define the maximum iteration steps k_max;
4  for k = 0 to k_max − 1 do
5  |  T = Temperature(1 − (k + 1)/k_max);
6  |  Generate a new state, s* = Neighbor(s);
7  |  if P(E(s), E(s*), T)) ≥ random(0,1) then
8  |  |  s = s*;
9  |  end
10 end
11 return s;
```

The temperature function Temperature() and the acceptance probability function P() has significant impact on the performance of simulated annealing. However, there are no choices that will be always good for all problems, and there is no general way to find the best choices for a given problem. In this thesis, simulated annealing is used for assigning the processors to the tasks. The Temperature() is defined as Temperature(T) = T and the acceptance probability function P() is defined as $\mathrm{P}(E(s), E(s^*), T) = e^{(E(s)-E(s^*))/T}$.

3

System and task model

This chapter defines the mixed-criticality parallel real-time task model and uniform heterogeneous multiprocessor platform.

3.1 Uniform multiprocessor platform

A uniform multiprocessor platform of m processors can be represented by their normalized speeds $\{\delta^1, \delta^2, \cdots, \delta^m\}$, where $\delta_k \leq 1$ for all $k \in \{1, ...p\}$. And they are sorted in non-increasing order($\delta^i \leq \delta^j$ for $i \leq j$). In a time interval of length t, the amount of workload executed on a processor with speed δ^x is $t\delta^x$. Therefore, if the WCET of a task on a processor with speed 1 is c, then its WCET becomes c/δ^x on a processor of speed δ^x.

3.2 System model for dual-criticality system

In the dual-criticality system, the system has two states: *low-criticality* mode and *high-criticality* mode. A task has two kinds of WCET: *nominal* worst case execution time and *overload* worst case execution time. When the system is in the low-criticality mode, the WCET of that task is its nominal worst case execution time. When the system is in the high-criticality mode, the WCET of that task is its overload worst case execution time. The nominal worst case execution time is the less pessimistic estimate generally expected to occur during normal operation, while the overload worst case execution time is the potentially much more pessimistic estimate considered during the certification process.

3.3 Task model

Each instance of a parallel task can be modeled as a *directed acyclic graph (DAG)*, where each node is a sequential unit and each edge represents a precedence constraint. Notes that are not connected by an edge may run in parallel. A node is *ready* to be executed when all its predecessors have been finished. Without loss of generality, we assume that every DAG has one node with no incoming edges, called the *head*, and every DAG also has one node with no outgoing edges, called the *tail*.

A task set with n DAGs is defined by $\tau = \{\tau_1, ...\tau_n\}$, where each τ_i is a DAG with index $i \in \{1, 2, ..., n\}$. For a DAG τ_i with v nodes, a node with index j of τ_i is denoted by $\tau_{i,j}$, where $1 \leq j \leq v$. A node has two kinds of WCET: *nominal* worst

case execution time and *overload* worst case execution time. $c_{i,j}^{LO}$ and $c_{i,j}^{HI}$ denote the nominal worst case execution time and overload worst case execution time of node $\tau_{i,j}$ on the processor with unit speed, respectively. Figure 3.1 shows a example for two DAG tasks.

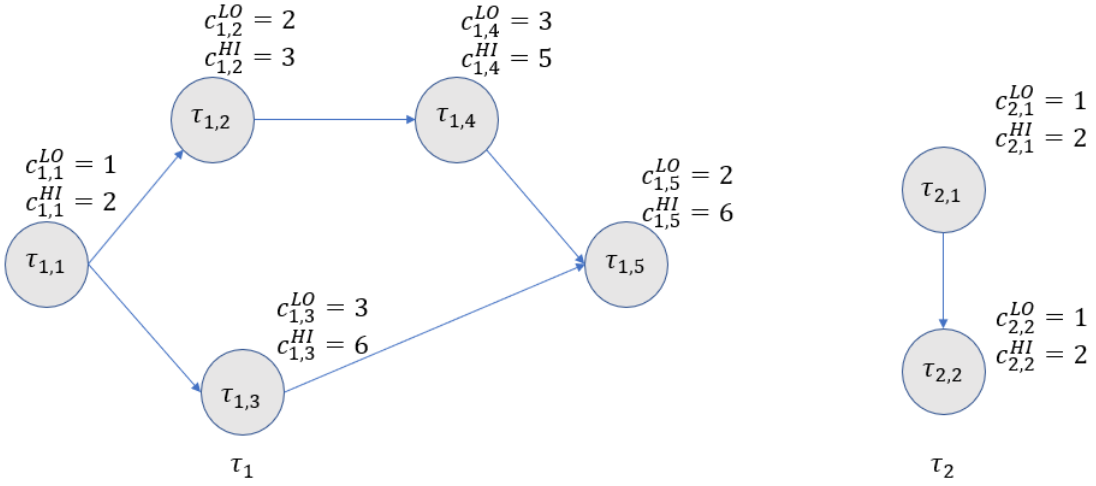

Figure 3.1: An example for two DAG tasks

The tuple $(Z_i, T_i, D_i, C_i^{HI}, C_i^{LO}, L_i^{HI}, L_i^{LO})$ characterizes a task τ_i. $Z_i \in \{HI, LO\}$ is the criticality of the task τ_i, where HI and LO specify that task τ_i is a HI-criticality task and LO-criticality task, respectively[1]. T_i is the minimum inter-arrival time of the jobs (called the period) of the task τ_i and D_i is the relative deadline of the task τ_i. In this thesis, we focus on *implicit-deadline sporadic tasks*, where $D_i = T_i$. C_i^{HI} and C_i^{LO} are the nominal and overload total workload of task τ_i. Similarly, L_i^{HI} and L_i^{LO} are, respectively, the nominal and overload workload of the critical-path of task τ_i. They are defined as definition 1 and 2.

Definition 1. *Total workload of task τ_i with v nodes under a certain criticality Z_i is defined as follows:*

$$C_i^{Z_i} = \sum_{j=1}^{v} c_{i,j}^{Z_i} \tag{3.1}$$

Definition 2. *A head-to-tail path γ_i of a task τ_i is a chain of nodes in which the first node is the head node, and the end node is the tail node. The set of all head-to-tail paths of task τ_i is denoted by $paths_i$. The critical path is a head-to-tail path with the longest length. The length of the critical path of task τ_i under a specific criticality Z_i is defined as follows:*

$$L^{Z_i} = arg \max_{\gamma_i \in paths_i} \sum_{\tau_{i,j} \in \gamma_i} c_{i,j}^{Z_i} \tag{3.2}$$

From the definition of nominal and overload worst case execution time in the section 3.2, set $C_i^{LO} = C_i^{HI}$, $L_i^{LO} = L_i^{HI}$ for a low-critical task τ_i and $C_i^{LO} = C_i^{HI}$, $L_k^{LO} < L_k^{HI}$ for a high-critical task τ_k.

[1]All the nodes of a DAG have the same criticality.

When a task τ_i executes on a processor p with speed δ^p, its *inflated* nominal and overload utilization is denoted as $u_i^{LO,p} = \frac{C_i^{LO}}{\delta^p D_i}$ and $u_i^{HI,p} = \frac{C_i^{HI}}{\delta^p D_i}$, respectively. The concepts of *light* and *heavy* task are based on the inflated utilization.

Definition 3. *Task τ_i is called a heavy task upon processor p with speed δ^p, it satisfies the relationship* $\max\left(u_i^{LO,p}, u_i^{HI,p}\right) > 1$*, otherwise it is called a light task upon processor p.*

Based on the utilization and the criticality, tasks are categorized into following four types on a specific processor p:

- Heavy and HI-criticality (HH) task: $Z_i = HI$ and $\max\left(u_i^{LO,p}, u_i^{HI,p}\right) > 1$
- Heavy and LO-criticality (HL) task: $Z_i = LO$ and $\max\left(u_i^{LO,p}, u_i^{HI,p}\right) > 1$
- Light and LO-criticality (LL) task: $Z_i = LO$ and $\max\left(u_i^{LO,p}, u_i^{HI,p}\right) \leq 1$
- Light and HI-criticality (LH) task: $Z_i = HI$ and $\max\left(u_i^{LO,p}, u_i^{HI,p}\right) \leq 1$

3.4 Virtual deadline and system behavior

Each HI-criticality task will be assigned a virtual deadline [2]. The idea of *virtual deadline* is first proposed by Baruah et al. in [9]. At a high level, the virtual deadline D_i^v for a high-criticality task τ_i is less than its deadline D_i. Generally, the virtual deadline has two purposes: (1) It gives the high-criticality job a higher priority so that the scheduler can detect whether it has exhibited overload behavior early. (2) furthermore, it provides enough time for the job to finish the overload work after the transition.

The behavior of the system is as follows: Firstly, the system is in LO-criticality mode. The time instant for checking whether the system should transition from Low-criticality mode to High-criticality mode is at the HI-criticality job's absolute virtual deadline. If some HI-criticality job does miss its absolute virtual deadline, the system will transition from LO-criticality mode to HI-criticality mode and all LO-criticality jobs are immediately discarded and no LO-criticality job will receive any execution at HI-criticality mode [3].

3.5 Target goal

This thesis proposes scheduling algorithms and corresponding schedulability tests to verify that the correctness (according to Definition 4) of a set of implicit mixed-criticality parallel tasks on a uniform machine is guaranteed. Chapter 4 presents the scheduling algorithms used in this thesis and Chapter 5 presents the schedulability test.

Definition 4. *A dual-criticality scheduler is MC-correct if, it satisfies the following two conditions:*

- *During the low-criticality mode, all the tasks can meet their deadlines.*

[2] The value of the virtual deadlines for the light and heavy task are presented in section 4.3 and section 5.1, respectively.

[3] This assumption will be relaxed in the chapter 7

- *After the system transitions into the high-criticality mode, all HI-criticality tasks can meet their deadlines.*

4

Scheduling algorithm

This chapter presents the scheduling algorithm for heavy and light tasks.

4.1 Federated scheduling

Our algorithm is based on federated scheduling, which is firstly presented by Li in [10]. It is a generalized partitioned scheduling algorithm for parallel tasks. When using federated scheduling algorithm, each heavy task is assigned a set of dedicated processors, called cluster, such that it can execute its cluster exclusively and will not be interfered by other tasks. For the light tasks, it shares the rest processors as partitioned scheduling.

This thesis uses the work-conserving scheduling algorithm to schedule each heavy DAG on its dedicated cluster. And a special earliest deadline first scheduling algorithm, called EDF-VD [9], is used to schedule the light tasks on their allocated processors.

4.2 Scheduling algorithm for heavy task

We use a work-conserving algorithm to schedule the heavy task on its dedicated cluster P.

On homogeneous multiprocessors, a work-conserving scheduling algorithm for parallel tasks, never makes any processor idle if some node is waiting for execution. In [11], Xu et al. extend the concept of work-conserving scheduling for parallel tasks to uniform multiprocessors. They define a scheduling algorithm for parallel tasks is work-conserving on m uniform processors if it satisfies both of the following conditions:

- No processor is idle when there exists some node's job waiting for execution.
- If at some time instant, the number of active node's job is less than the number of processor m, then the active jobs can migrate to the fastest processors for execution.

The pseudo code for the work-conserving scheduling on uniform multiprocessors is shown as algorithm 3.

The node's job, which is waiting for execution, is stored in the ReadyQ, and the RunQ keeps track of the active jobs. If the head node of the task release a job or one of the nodes' jobs finishes, the scheduler is invoked (line 1). The jobs in the RunQ can migrate to the fastest idle processor. After the migration, the processor it was executing on is marked as idle. We check all the active jobs to see whether

Algorithm 3: Work-conserving scheduler on uniform multiprocessors

```
1  if Release or Completion event then
2      forall τ_{i,j} ∈ RunQ do
3          Find the idle processor p_k with smallest k;
4          Execute τ_{i,j} to processor p_k, update platform's status;
5      end
6      while There are idle processors and the ReadyQ is not empty do
7          τ_{i,j} = ReadyQ.pop;
8          Find the idle processor p_k with smallest k;
9          Execute τ_{i,j} to processor p_k, update platform's status;
10     end
11 end
```

they can migrate (lines 2 - 5). If there exists an idle processor and some job waiting for execution, the ReadyQ will dispatch a job from its head. The dispatched job will execute on the fastest idle processor(lines 6 - 10).

Figure 4.1 shows a possible scheduling sequence of the DAG, which is shown as τ_1 in figure 3.1, on two processors with speeds $\{1, 0.5\}$, when system is in LO-criticality mode. Node $\tau_{1,3}$ and node $\tau_{1,4}$ migrate to the fastest processor at time 3 and 5 respectively.

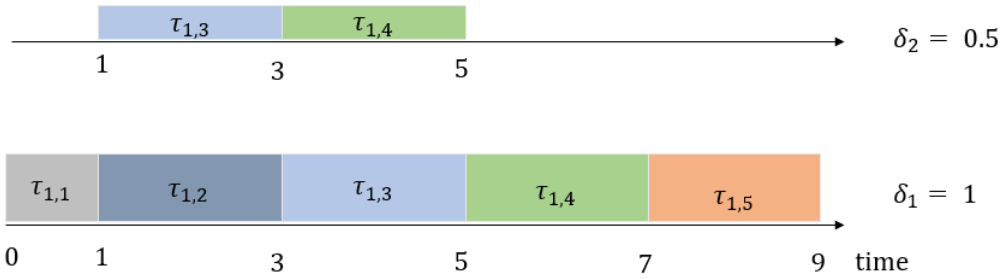

Figure 4.1: A work-conserving scheduling sequence on a uniform platform with 2 processors.

It has been proved that if the migration is forbidden. In the worst case, the critical path of the DAG will be executed on the lowest processor, which makes the response time of that DAG become very large [11]. To make it schedulable, we need to allocate more processors, which is a significant computing resource waste.

4.3 Scheduling algorithm for light task

Baruah et al. propose an algorithm, EDF-VD, to schedule mixed-criticality implicit-deadline sporadic tasks on a preemptive uniprocessor in [9]. When given a set of tasks τ and for each task $\tau_i \in \tau$, it is a light task on processor p, which means $\max\left(u_i^{LO,p}, u_i^{HI,p}\right) \leq 1$, we can apply earliest deadline first with virtual deadline algorithm, `EDF-VD`, to schedule them.

Definition 5. *Let the task set τ executed on the processor p with speed δ^p. For each of x and y in $\{LO, HI\}$, we define a utilization parameter on the processor p as follows:*

$$U_x^{y,p}(\tau) = \sum_{\tau_i \in \tau \wedge Z_i = x} \frac{C_i^y}{\delta^p T_i} \tag{4.1}$$

For example, $U_{LO}^{LO,p}(\tau)$ and $U_{HI}^{LO,p}(\tau)$ denote the sum of the utilizations of the LO-criticality tasks and HI-criticality tasks in τ at LO-criticality mode on the processor p, respectively.

When using EDF-VD to schedule a set of task τ, there is a pre-processing phase. At the pre-processing stage, a strategy is used to determine whether the task set is schedulable. If the task set is deemed schedulable, the *modified period* (the $\hat{T}_i$) is computed for the each HI-criticality task. As shown in algorithm 4, EDF-VD first computes a parameter x and then assigns values to the $\hat{T}_i$ parameters for all HI-criticality tasks as equation 4.2:

$$\hat{T}_i \leftarrow x \times T_i \tag{4.2}$$

Algorithm 4: The pre-processing stage of EDF-VD

Input: The task set τ, the processor p with speed δ_p

compute x as follows:

$$x \leftarrow \frac{U_{HI}^{LO,p}(\tau)}{1 - U_{LO}^{LO,p}(\tau)}$$

if $xU_{LO}^{LO,p}(\tau) + U_{HI}^{HI,p}(\tau) \leq 1$ **then**
  $\hat{T}_i \leftarrow xT_i$ for each HI-criticality task τ_i
  return **Schedulable**
end
return **Unschedulable**

The reason why assigning the modified period as this value and why this pre-processing stage can check whether the task set is schedulable are explained at Chapter 5 Lemma 2 and Lemma 3.

The EDF-VD still uses the EDF scheduler. The difference between EDF-VD and EDF is that the priority set by the EDF-VD scheduler is according to the `virtual deadline`.

1. At LO-criticality mode, the priorities of jobs are determined by the virtual deadline. The job with earliest absolute deadline is selected for execution will have the highest priority. Suppose that a job of task τ_i arrives at time-instant t, the virtual deadline is assigned as follows:
   - If it is a LO-criticality job, its absolute virtual deadline equals $t + T_i$.
   - If it is a HI-criticality job, its absolute virtual deadline equals $t + \hat{T}_i$.
2. At HI-criticality mode, all the LO-criticality jobs are discarded. Use EDF to scheduling all the HI-criticality jobs. The priority of job is determined by its absolute deadline that is its release time plus its original deadline (the $D_i = T_i$, not the $\hat{T}_i$).

4.4 Summary

This chapter presents the scheduling algorithm used by our algorithm. Our algorithm is based on federated scheduling. Tasks are categorized as heavy and light tasks. Heavy tasks are executed on its dedicated cluster in isolation, and the working-conserving scheduler is used at the cluster level. For light tasks, use partitioned scheduling, and EDF-VD is used at each uniprocessor level.

Chapter 5 gives the upper bound of the response time of a heavy task under the working-conserving scheduler and the utilization bound for checking whether a set of light tasks is schedulable on a uniprocessor.

5

Schedulability test

Mapping tasks to processors relies on the schedulability test. This chapter develops the upper bound of response time for the work-conserving scheduling algorithm and shows the utilization bound for EDF-VD.

5.1 Response time analysis for the working conserving scheduling algorithm

In [12], Funk et al. first introduce the concept *uniformity* of the uniform platform.
Definition 6. *The uniformity of m processors with speeds* $\{\delta^1, \cdots, \delta^m\}$ $(\delta^x \geq \delta^{x+1})$ *is defined as*

$$\lambda = \max_{x=1}^{m} \left\{ \frac{S_m - S_x}{\delta^x} \right\} \tag{5.1}$$

where S_x *is the sum of the speeds of the x fastest processors:*

$$S_x = \sum_{j=1}^{x} \delta^j \tag{5.2}$$

The *uniformity* intuitively measures how similar a uniform multiprocessor system and corresponding homogeneous multiprocessor system, which has the same number of processors are. If a platform is formed by m identical processors, the *uniformity* of that platform is $(m-1)$ and becomes progressively smaller as the speeds of the processors differ from each other by greater amounts.

Xu et al. give an upper bound of response time of a DAG task τ_i executing on m processors with speeds $\{\delta^1, \delta^2, \cdots, \delta^m\}$ $(\delta^x \geq \delta^{x+1})$ under work-conserving scheduling in [11], which is as equation 5.3 shown.

$$R \leq \frac{C_i + \lambda L_i}{S_m} \tag{5.3}$$

where C_i is the total worst-case execution time of all vertices of DAG task τ_i on the fastest processor, L_i is the sum of the worst-case execution time of each vertex along the critical path of τ_i, and S_m is the total capacity of the platform.

Let m_i^{HI} and m_i^{LO} denote the cluster assigned to task τ_i in HI-criticality and LO-criticality mode respectively. S_i^{HI} and S_i^{LO} denote the total capacity of the cluster m_i^{HI} and m_i^{LO} respectively, which is the sum of processor speeds. And the uniformity of m_i^{HI} and m_i^{LO} are denoted as λ_i^{HI} and λ_i^{LO} respectively. Without loss of generality, we assume the processors in the cluster are sorted in non-increasing speed order $(\delta^x \geq \delta^{x+1})$.

$$
\begin{aligned}
S_i^{HI} &= \sum_{x=1}^{|m_i^{HI}|} \delta_i^{HI,x}, \lambda_i^{HI} = \max_{x=1}^{|m_i^{HI}|} \left\{ \frac{S_i^{HI} - S_i^{HI,x}}{\delta_i^{HI,x}} \right\} \\
S_i^{LO} &= \sum_{x=1}^{|m_i^{LO}|} \delta_i^{LO,x}, \lambda_i^{LO} = \max_{x=1}^{|m_i^{LO}|} \left\{ \frac{S_i^{LO} - S_i^{LO,x}}{\delta_i^{LO,x}} \right\}
\end{aligned}
\tag{5.4}
$$

where $S_i^{LO,x}$ and $S_i^{HI,x}$ are the sum of the speeds of x faster processors of the cluster m_i^{LO} and m_i^{HI}. $\delta_i^{LO,x}$ and $\delta_i^{HI,x}$ are the x^{th} processor speed of the cluster m_i^{LO} and m_i^{HI} respectively.

Now we will present the schedulability analysis and the corresponding tests for HH and HL tasks.

1) Schedulability analysis for HL task

The virtual deadline for each HL task τ_i is $D_i^v = D_i$. Since the LO-criticality task is discarded during the HI-criticality mode, the cluster m_i^{HI} assigned to it at this mode is an empty set.

Theorem 1. *The execution of each HL task τ_i is correct using the work-conserving scheduler, if the processor cluster m_i^{LO} assigned to it, satisfies the condition 5.5.*

$$
\frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{S_i^{LO}} \leq D_i \tag{5.5}
$$

Proof. We can apply the equation 5.3 to get the upper bound of the response time and let this bound be less than the DAG's deadline. □

2) Schedulability analysis for HH task

To utilize the processor efficiently, we set that $m_i^{LO} \subseteq m_i^{HI}$. The virtual deadline D_i^v for HH task τ_i is assigned as follows:

$$
D_i^v = \frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{S_i^{LO}} \tag{5.6}
$$

Lemma 1. *If $m_i^{LO} \subseteq m_i^{HI}$, then $S_i^{LO} \leq S_i^{HI}$ and $\lambda_i^{LO} \leq \lambda_i^{HI}$.*

Proof. Let $\Delta m_i = m_i^{HI} \setminus m_i^{LO}$. Since $m_i^{LO} \subseteq m_i^{HI}$, $|\Delta m_i| \geq 0$. Let $\Delta\delta_i^x$ be the x^{th} processor speed of the cluster Δm_i. It is clear that $\Delta\delta_i^x > 0$. $S_i^{HI} - S_i^{LO} = \sum_{x=1}^{|\Delta m_i|} \Delta\delta_i^x \geq 0$. So $S_i^{LO} \leq S_i^{HI}$.

The uniformity can be rewritten as

$$
\lambda = \max_{i=1}^{m} \frac{\sum_{j=i+1}^{m} \delta^j}{\delta^i} \tag{5.7}
$$

Since the processor speeds in the cluster are sorted in non-increasing order and $m_i^{LO} \subseteq m_i^{HI}$, for each $\delta_i^{LO,x} \in m_i^{LO}$, the cluster m_i^{HI} also contains all $\delta_i^{LO,(x+\Delta)}(0 \leq \Delta \leq |m_i^{LO}| - x)$. Assume $\lambda_i^{LO} = \frac{\sum_{j=k+1}^{|m_i^{LO}|} \delta_i^{LO,j}}{\delta_i^{LO,k}}$ and $\delta_i^{LO,k} = \delta_i^{HI,k'}$ we can have

$$\begin{aligned} \sum_{j=k+1}^{|m^{LO}|} \delta_i^{LO,j} &\leq \sum_{j=k'+1}^{|m^{HI}|} \delta_i^{HI,j} \\ \Leftrightarrow \frac{\sum_{j=k+1}^{|m^{LO}|} \delta_i^{LO,j}}{\delta_i^{LO,k}} &\leq \frac{\sum_{j=k'+1}^{|m^{HI}|} \delta_i^{HI,j}}{\delta_i^{HI,k'}} \leq \lambda_i^{HI} \end{aligned} \tag{5.8}$$

So we can get $\lambda_i^{LO} \leq \lambda_i^{HI}$.

□

Theorem 2. *Consider a pair of cluster* $\left(m_i^{LO}, m_i^{HI}\right)$*, such that the HH task* τ_i *is assigned dedicated cluster* m_i^{LO} *and* m_i^{HI} *for the LO- and HI-criticality mode, respectively, where* $m_i^{LO} \subseteq m_i^{HI}$*. Each job of task* τ_i *meets its deadline in all correct states if the following condition is satisfied:*

$$\left(C_i^{LO} + \lambda_i^{LO} L_i^{LO}\right)\left(\frac{1}{S_i^{LO}} - \frac{1}{S_i^{HI}}\right) + \frac{C_i^{HI} + \lambda_i^{HI} L_i^{HI}}{S_i^{HI}} \leq D_i \tag{5.9}$$

Proof. Without loos of generality, we consider a job J_i, which is released at time instant 0, of HL task τ_i. The execution of job J_i happens in three possible scenarios: (1) Stable LO-criticality mode, (2) Stable HI-criticality mode, and (3) mode switching. A stable mode means that there is no system state change during the execution of job J_i. This theorem is proved by showing that job J_i can meet its deadline for all these three possible execution scenarios if condition 5.9 satisfied.

Stable LO-criticality mode. During the stable LO-criticality, the task τ_i is executed on cluster m_i^{LO}. Since its job J_i executes entirely in stable mode, it should signal completion at or before its virtual deadline. So the upper bound of the response time R_{LO} for a job of HH task in the stable LO-criticality should satisfy condition 5.10 .

$$R_{\mathrm{LO}} \leq \frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{S_i^{LO}} \tag{5.10}$$

From Lemma 1, we can get $\lambda_i^{LO} \leq \lambda_i^{HI}$. Since for each node $c_{i,j}^{LO} \leq c_{i,j}^{HI}$, $C_i^{LO} \leq C_i^{HI}$ and $L_i^{LO} \leq L_i^{HI}$. So we can have

$$\frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{S_i^{LO}} \leq \frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{S_i^{LO}} + \frac{C_i^{HI} - C_i^{LO} + \lambda_i^{HI} L_i^{HI} - \lambda_i^{LO} L_i^{LO}}{S_i^{HI}} \leq D_i \tag{5.11}$$

Combining equation 5.10, we can get

$$R_{\mathrm{LO}} \leq \frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{S_i^{LO}} = D_i^v \leq D_i \tag{5.12}$$

Stable HI-critical mode. During the stable HI-criticality mode, the task τ_i is executed on cluster m_i^{HI}. Since its job J_i executes entirely in stable mode, it should signal completion at or before its deadline. So the upper bound of the response time R_{HI} for a job of HH task in the stable HI-criticality mode should satisfy condition 5.13.

$$R_{\text{HI}} \leq \frac{C_i^{HI} + \lambda_i^{HI} L_i^{HI}}{S_i^{HI}} \leq D_i \tag{5.13}$$

From Lemma 1, we can get $S_i^{LO} \leq S_i^{HI}$ and $(1/S^{LO} - 1/S^{HI}) > 0$. So we can have

$$\frac{C_i^{HI} + \lambda_i^{HI} L_i^{HI}}{S_i^{HI}} \leq \frac{C_i^{HI} + \lambda_i^{HI} L_i^{HI}}{S_i^{HI}} + \left(C_i^{LO} + \lambda_i^{LO} L_i^{LO}\right)\left(\frac{1}{S_i^{LO}} - \frac{1}{S_i^{HI}}\right) \leq D_i \tag{5.14}$$

Combining equation 5.13, we can get

$$R_{\text{HI}} \leq \frac{C_i^{HI} + \lambda_i^{HI} L_i^{HI}}{S_i^{HI}} \leq D_i \tag{5.15}$$

Mode Switching The transition occurred during the execution of job J_i. Assume the transition to HI-criticality mode occurred at time t^*. It is clear that $0 < t^* \leq D_i^v$. (If $t^* > D_i^v$, then J_i has finished executing already. That contradicts that assumption.) So the job J_i executes on cluster m_i^{LO} during the interval $[0, t^*)$ and on cluster m_i^{HI} after t^*.

During the time interval $[0, t^*]$, let $\alpha_i^{LO,x}$ denote the total length of intervals during which the x^{th} processor of cluster m_i^{LO} (with speed $\delta_i^{LO,x}$) is busy. Since the work-conserving scheduling does not let any processor be idle and migration is allowed, if the x^{th} processors is busy in a time interval then all the processors which has smaller index are also always busy. Because they have a higher processor speed. Therefore, we know $t^* = \alpha_i^{LO,1}$. We define a parameter β as equation 5.16.

$$\beta_i^{LO,x} = \begin{cases} \alpha_i^{LO,x} - \alpha_i^{LO,x+1}, & 1 \leq x < |m_i^{LO}| \\ \alpha_i^{LO,x}, & x = |m_i^{LO}| \end{cases} \tag{5.16}$$

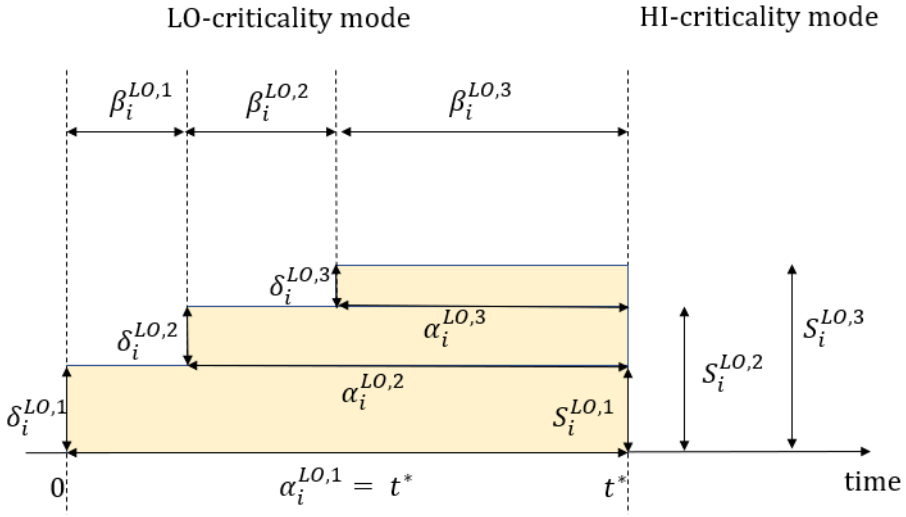

Figure 5.1: Illustration of $\alpha_i^{LO,x}$ and $\beta_i^{LO,x}$

Figure 5.1 illustrates the definition of $\alpha_i^{LO,x}$ and $\beta_i^{LO,x}$. So we can rewrite $t^* = \alpha_i^{LO,1}$ as

$$t^* = \sum_{x=1}^{|m_i^{LO}|} \beta_i^{LO,x} \tag{5.17}$$

The total workload executed on all the processors at the LO-criticality mode is

$$C_i^{LO} = \sum_{x=1}^{|m_i^{LO}|} \beta_i^{LO,x} S_i^{LO,x} \tag{5.18}$$

The remaining amount of total work at time t^*, denoted by C_{remain}, from the equation 5.18, we can have

$$C_{\text{remain}} = C_i^{HI} - \sum_{x=1}^{|m_i^{LO}|} \beta_i^{LO,x} S_i^{LO,x} \tag{5.19}$$

During the time interval $\beta_i^{LO,x}$, for $1 \leq x \leq |m_i^{LO}| - 1$, the two proprieties are satisfied:

- at least one processor is idle
- the slowest processor speed among the busy processor is $\delta_i^{LO,x}$

We use $w(\delta_i^{LO,x})$ to denote the total amount of workload executed on the cluster m_i^{LO} before D_i^v at time interval $\beta_i^{LO,x}$ for $1 \leq x \leq |m_i^{LO}| - 1$. At such time interval, the length of the critical path is decreased. Because there are at least one processor is idle. Since the slowest busy processor has speed $\delta_i^{LO,x}$, we can have

$$\begin{aligned} & w\left(\delta_i^{LO,x}\right) \geq \beta_i^{LO,x} \delta_i^{LO,x} \\ \Rightarrow & \sum_{x=1}^{|m_i^{LO}|-1} w\left(\delta_i^{LO,x}\right) \geq \sum_{x=1}^{|m_i^{LO}|-1} \beta_i^{LO,x} \delta_i^{LO,x} \end{aligned} \tag{5.20}$$

So the length of critical path is reduced at least $\sum_{x=1}^{|m_i^{LO}|-1} \beta_i^{LO,x} \delta_i^{LO,x}$. The remaining workload of the critical path at time t^*, denoted by L_{remain}, can have

$$L_{\text{remain}} \leq L_i^{HI} - \sum_{x=1}^{|m_i^{LO}|-1} \beta_i^{LO,x} \delta_i^{LO,x} \tag{5.21}$$

After t^*, we can model a new DAG with total workload C_{remain} and the critical path length L_{remain} assigned on the cluster m_i^{HI}. So the upper bound of the response time R_s of job J_i is shown as equation 5.22.

$$R_s \leq t^* + \frac{C_{\text{remain}} + \lambda_i^{HI} L_{\text{remain}}}{S_i^{HI}} \tag{5.22}$$

From equation 5.19 and 5.21, we can have

$$R_s \leq t^* + \frac{C_i^{HI} - \sum_{x=1}^{|m_i^{LO}|} \beta_i^{LO,x} S_i^{LO,x} + \lambda_i^{HI} \left(L_i^{HI} - \sum_{x=1}^{|m_i^{LO}|-1} \beta_i^{LO,x} \delta_i^{LO,x} \right)}{S_i^{HI}} \tag{5.23}$$

From Lemma 1, we can have $\lambda_i^{HI} \geq \lambda_i^{LO}$. Combine the definition of λ in equation 5.1, we know for all $1 \leq x \leq |m_i^{LO}|$, the condition 5.24 is satisfied.

$$\frac{S_i^{LO} - S_i^{LO,x}}{\delta_i^{LO,x}} \leq \lambda_i^{LO} \leq \lambda_i^{HI} \tag{5.24}$$

Let $B = \sum_{x=1}^{|m_i^{LO}|} \beta_i^{LO,x} S_i^{LO,x} + \lambda_i^{HI} \sum_{x=1}^{|m_i^{LO}|-1} \beta_i^{LO,x} \delta_i^{LO,x}$. From condition 5.24, we can get

$$\begin{aligned} B &\geq \sum_{x=1}^{|m_i^{LO}|} \beta_i^{LO,x} S_i^{LO,x} + \sum_{x=1}^{|m_i^{LO}|-1} \beta_i^{LO,x} \left(S_i^{LO} - S_i^{LO,x} \right) \\ &\Leftrightarrow \beta_i^{LO,|m_i^{LO}|} S_i^{LO} + \sum_{x=1}^{|m_i^{LO}|-1} \beta_i^{LO,x} S_i^{LO} \\ &\Leftrightarrow S_i^{LO} \sum_{x=1}^{|m_i^{LO}|} \beta_i^{LO,x} \end{aligned} \tag{5.25}$$

Since $\sum_{x=1}^{|m_i^{LO}|} \beta_i^{LO,x} = t^*$, we can get

$$-B \leq -S_i^{LO} t^* \tag{5.26}$$

Combining condition 5.23 and 5.26 gives

$$\begin{aligned} R_s &\leq t^* + \frac{C_i^{HI} + \lambda_i^{HI} L_i^{HI} - S_i^{LO} t^*}{S_i^{HI}} \\ &\text{(since } 0 < t^* \leq D_i^v\text{)} \\ &\leq D_i^v + \frac{C_i^{HI} + \lambda_i^{HI} L_i^{HI} - S_i^{LO} D_i^v}{S_i^{HI}} \end{aligned} \tag{5.27}$$

From equation 5.6, we get

$$R_s \leq \left(C_i^{LO} + \lambda_i^{LO} L_i^{LO} \right) \left(\frac{1}{S_i^{LO}} - \frac{1}{S_i^{HI}} \right) + \frac{C_i^{HI} + \lambda_i^{HI} L_i^{HI}}{S_i^{HI}} \leq D_i \tag{5.28}$$

Therefore, the generic job j_i of HH task τ_i meet its deadline in all the three scenarios, the theorem is proved. □

5.2 Utilization bound for EDF-VD

In this section, we show a utilization bound of the EDF-VD on a specific processor. Assume a set of light task τ is scheduled by EDF-VD on the processor p with speed δ^p and for each task $\tau_i \in \tau$, it is a light task upon processor p.

Lemma 2. ***(Theorem 1 from [9])****. The following condition is a sufficient condition for ensuring that the task set τ is schedulable by EDF-VD on the processor p with speed δ^p at LO-criticality mode:*

$$x \geq \frac{U_{HI}^{LO,p}(\tau)}{1 - U_{LO}^{LO,p}(\tau)} \tag{5.29}$$

Proof. Since at LO-critcality mode, each HI-criticality task τ_i will be scheduled by its modified period $\hat{T}_i = xT_i$. Scaling down the period of each HI-criticality task by a factor x is equivalent to inflating its utilization by a factor $1/x$. From the utilization bound result of EDF [6], we can get that

$$\begin{aligned} & U_{LO}^{LO,p}(\tau) + \frac{U_{HI}^{LO,p}(\tau)}{x} \leq 1 \\ \Leftrightarrow & \frac{U_{HI}^{LO,p}(\tau)}{x} \leq 1 - U_{LO}^{LO,p}(\tau) \\ \Leftrightarrow & x \geq \frac{U_{\mathrm{HI}}^{LO,p}(\tau)}{1 - U_{LO}^{LO,p}(\tau)} \end{aligned} \tag{5.30}$$

is sufficient for guaranteeing the task set $\tau$is schedulable by EDF-VD at LO-criticality mode. □

EDF-VD chooses for x the smallest value such that Lemma 2 is satisfied:

$$x = \frac{U_{\mathrm{HI}}^{LO,p}(\tau)}{1 - U_{LO}^{LO,p}(\tau)} \tag{5.31}$$

This is the reason why assigning the modified period as this value as equation 4.2.

With this value of x, we now determine a sufficient condition for ensuring that all HI-criticality tasks can meet their deadlines at HI-criticality mode under EDF-VD.

Lemma 3. ***(Theorem 2 from [9])****. The following condition is a sufficient condition for ensuring that task set τ is schedulable by EDF-VD on processor p with speed δ^p at HI-criticality mode:*

$$xU_{LO}^{LO,p}(\tau) + U_{HI}^{HI,p}(\tau) \leq 1 \tag{5.32}$$

Proof. Suppose that τ satisfies condition 5.29, all deadlines can be meet of τ at LO-criticality mode, but EDF-VD cannot meet all deadlines of τ at HI-criticality mode.

Consider a instance I of set of jobs, which satisfied the following condition:

- The earliest release time of job in I is 0
- Only one job will miss its deadline and others can meet their deadline
- All the jobs are schedulable at LO-criticality mode

Let t_f denote the time instant of the deadline miss. Because all the jobs in the I must be schedulable at LO-criticality mode, t_f must be the deadline of a HI-criticality job. Let t^* denote the time instant at which the system switch to HI-criticality mode. Let J_1 denote the job with the earliest release time amongst all those that execute in $[t^*, t_f)$. Assume the release time of J_1 is a_1 and its deadline is d_1. Assume that the total workload over the interval $[0, t_f]$ of a job J_i is denoted as η_i.

Proposition 1. ***(Fact 1 from [9])***

All jobs in the I that execute in $[t^*, t_f)$ have deadline $\leq t_f$.

Proposition 2. ***(Fact 2 from [9])***

All LO-criticality jobs in the I have deadline $\leq a_1 + x(t_f - a_1)$.

Proposition 3. ***(Fact 3 from [9])***
All HI-criticality jobs in the I with release time $< a_1$*, have modified deadline* $\leq a_1 + x(t_f - a_1)$.

1) The workload for LO-criticality task τ_i:
From Proposition 2, for any LO-criticality task τ_i, it workload has

$$\eta_i \leq u_i^{LO,p} \left(a_1 + x \left(t_f - a_1\right)\right) \delta^p \tag{5.33}$$

2) The workload for HI-criticality task τ_i:
Case a: If τ_i does not release a job at or after a_1. According to Proposition 2, each job has a modified deadline $\leq (a_1 + x\,(t_f - \ a_1)\ )$, their actual deadlines are all $\leq \frac{a_1}{x} + (t_f - a_1)$. Their cumulative workload requirement is at most

$$\begin{aligned} &\frac{a_1}{x} u_i^{LO,p} \delta^p + \left(t_f - a_1\right) u_i^{LO,p} \delta^p \\ \leq &\frac{a_1}{x} u_i^{LO,p} \delta^p + \left(t_f - a_1\right) u_i^{HI,p} \delta^p \end{aligned} \tag{5.34}$$

Case b: If τ_i releases a job at or after a_1. Let a_i denote the first release $\geq a_1$. The cumulative workload requirement of all jobs of τ_i is at most

$$a_i u_i^{LO,p} \delta^p + \left(t_f - a_i\right) u_i^{HI,p} \delta^p \tag{5.35}$$

Since $a_1 \leq a_i$ and $u_i^{LO,p} \leq u_i^{HI,p}$ and $x \leq 1$, we can have

$$\begin{aligned} &a_i u_i^{LO,p} \delta^p + \left(t_f - a_i\right) u_i^{HI,p} \delta^p \\ \leq &\, a_1 u_i^{LO,p} \delta^p + \left(t_f - a_1\right) u_i^{HI,p} \delta^p \\ \leq &\frac{a_1}{x} u_i^{LO,p} \delta^p + \left(t_f - a_1\right) u_i^{HI,p} \delta^p \end{aligned} \tag{5.36}$$

So for any LO-criticality task τ_i has

$$\eta_i \leq \frac{a_1}{x} u_i^{LO,p} \delta^p + \left(t_f - a_1\right) u_i^{HI,p} \delta^p \tag{5.37}$$

Let us sum the cumulative workload of all tasks over $[0, t_f)$:

$$\begin{aligned} &\sum_{\tau_i \in \tau \wedge Z_i = LO} \eta_i + \sum_{\tau_i \in \tau \wedge Z_i = HI} \eta_i \\ \leq &\sum_{\tau_i \in \tau \wedge Z_i = LO} u_i^{LO,p} \left(a_1 + x \left(t_f - a_1\right)\right) \delta^p + \sum_{\tau_i \in \tau \wedge Z_i = HI} \frac{a_1}{x} u_i^{LO,p} \delta^p + \left(t_f - a_1\right) u_i^{HI,p} \delta^p \\ = &a_1 \left(U_{LO}^{LO,p}(\tau) + \frac{U_{HI}^{LO,p}(\tau)}{x}\right) \delta^p + \left(t_f - a_1\right) \left(x U_{LO}^{LO,p}(\tau) + U_{HI}^{HI,p}(\tau)\right) \delta^p \end{aligned} \tag{5.38}$$

Since $U_{LO}^{LO,p}(\tau) + \frac{U_{HI}^{LO,p}(\tau)}{x} \leq 1$,

$$\sum_{\tau_i \in \tau \wedge Z_i = LO} \eta_i + \sum_{\tau_i \in \tau \wedge Z_i = HI} \eta_i \leq \delta^p a_1 + \left(t_f - a_1\right) \left(x U_{LO}^{LO,p}(\tau) + U_{HI}^{HI,p}(\tau)\right) \delta^p \tag{5.39}$$

We can form a necessary infeasibility condition of this instance that

$$\begin{aligned} &\delta^p a_1 + (t_f - a_1)\left(xU_{LO}^{LO,p}(\tau) + U_{HI}^{HI,p}(\tau)\right)\delta^p > t_f\delta^p \\ \Leftrightarrow &(t_f - a_1)\left(xU_{LO}^{LO,p}(\tau) + U_{HI}^{HI,p}(\tau)\right) > (t_f - a_1) \\ \Leftrightarrow &xU_{LO}^{LO,p}(\tau) + U_{HI}^{HI,p}(\tau) > 1 \end{aligned} \tag{5.40}$$

From equation 5.40, it follows that $xU_{LO}^{LO,p}(\tau)+U_{HI}^{HI,p}(\tau) \leq 1$ is sufficient to ensure task set is schedulable by EDF-VD at HI-criticality mode. □

Theorem 3. ***(Theorem 4 from [9]).*** *Any set of light tasks τ satisfying the property*

$$\max\left(U_{LO}^{LO,p}(\tau) + U_{HI}^{LO,p}(\tau), U_{HI}^{HI,p}(\tau)\right) \leq \frac{3}{4} \tag{5.41}$$

is successfully scheduled by EDF-VD on a preemptive processor p with speed δ^p.

Proof. Let b denote an upper bound on both LO-criticality utilization and the HI-criticality utilization of task set τ:

$$b \geq \max\left(U_{LO}^{LO,p}(\tau) + U_{HI}^{LO,p}(\tau), U_{HI}^{HI,p}(\tau)\right) \tag{5.42}$$

By Lemma 2 and Lemma 3, we know that if an x satisfying both lemma, there will be no deadline miss. Since Lemma 2 and Lemma 3 require that

$$\begin{aligned} x &\geq \frac{U_{HI}^{LO,p}(\tau)}{1 - U_{LO}^{LO,p}(\tau)} \\ x &\leq \frac{1 - U_{HI}^{HI,p}(\tau)}{U_{LO}^{LO,p}(\tau)} \end{aligned} \tag{5.43}$$

we can get equation 5.44 as a sufficient condition for τ to successfully scheduled by EDF-VD:

$$\frac{U_{HI}^{LO,p}(\tau)}{1 - U_{LO}^{LO,p}(\tau)} \leq \frac{1 - U_{HI}^{HI,p}(\tau)}{U_{LO}^{LO,p}(\tau)} \tag{5.44}$$

Since $U_{LO}^{LO,p}(\tau) + U_{HI}^{LO,p}(\tau) \leq b$, $U_{HI}^{LO,p}(\tau) \leq b - U_{LO}^{LO,p}(\tau)$. we can have

$$\frac{b - U_{LO}^{LO,p}(\tau)}{1 - U_{LO}^{LO,p}(\tau)} \leq \frac{1 - U_{HI}^{HI,p}(\tau)}{U_{LO}^{LO,p}(\tau)} \tag{5.45}$$

Since $U_{HI}^{HI,p} \leq b$, we can get

$$\begin{aligned} &\frac{b - U_{LO}^{LO,p}(\tau)}{1 - U_{LO}^{LO,p}(\tau)} \leq \frac{1 - b}{U_{LO}^{LO,p}(\tau)} \\ \Leftrightarrow &(U_{LO}^{LO,p}(\tau))^2 - U_{LO}^{LO,p}(\tau) + (1 - b) \geq 0 \end{aligned} \tag{5.46}$$

If we set $b = \frac{3}{4}$, equation 5.46 becomes:

$$\begin{aligned} &(U_{LO}^{LO,p}(\tau))^2 - U_{LO}^{LO,p}(\tau) + \frac{1}{4} \geq 0 \\ \Leftrightarrow &\left(U_{LO}^{LO,p}(\tau) - \frac{1}{2}\right)^2 \geq 0 \end{aligned} \tag{5.47}$$

which is true for all values of $U_{LO}^{LO,p}$.

So when $\frac{3}{4} \geq \max\left(U_{LO}^{LO,p}(\tau) + U_{HI}^{LO,p}(\tau), U_{HI}^{HI,p}(\tau)\right)$, Lemma 2 and Lemma 3 will be satisfied. That means task set τ is schedulable. □

5.3 Summary

The summary for scheduling algorithm and schedulability test for four different types of tasks is shown as table 7.1. Here assume that the LH and LL tasks are assigned on the processor p with speed δ^p.

Task Type	Criticality	Virtual Deadline	Schedulability test	Scheduling algorithm
HL	Low	-	Theorem 1	Work-conserving scheduling
HH	High	$(C_i^{LO} + \lambda_i^{LO} L_i^{LO})/S_i^{LO}$	Theorem 2	
LL	Low	-	Theorem 3	EDF-VD
LH	High	$\frac{U_{\mathrm{HI}}^{LO,p}(\tau)}{1-U_{LO}^{LO,p}(\tau)} D_i$		

Table 5.1: Scheduling algorithm and scheduability test for four different types of tasks

6

Task allocation

The task-to-processor assignment algorithm is crucial for guaranteeing the MC-correctness for federated scheduling on multiprocessors. But the problem of searching the clusters of processors that would make all the DAGs meet their deadlines is NP-hard in the strong sense [4]. This chapter builds a heuristics task allocation framework.

6.1 Overview

In this chapter, we assume the initial task set is denoted as τ, and the initial available processor set is P.

Definition 7. *When given a processor set P, assume processor p_{max} is the processor with the highest speed in the processor set P. If a task set τ, for each task $\tau_i \in \tau$ satisfies the relationship* $\max\left(u_i^{LO,p_{max}}, u_i^{HI,p_{max}}\right) \leq \delta^{p_{max}}$*, the task set τ is a light task set for the processor set P.*

The task allocation has two main stages:

- In the first stage, we select a set processors P_{ps} and get its light task set τ_{light} according to definition 7. We use *partitioned scheduling algorithm* to schedule the task set τ_{light} on the processor set P_{ps}. EDF-VD is used upon every uniprocessor which is in the processor P_{ps}. We do not require any task in the task set τ_{light} to be successfully scheduled on any processor P_{ps} at the stage. After the first stage, assume the task set τ_{success} is successfully scheduled and the processor set $P_{\text{allocation}}$ is used to host the task set τ_{success}. Let $\tau_{\text{remain}} = \tau \setminus \tau_{\text{success}}$ and $P_{\text{remain}} = P \setminus P_{\text{allocation}}$.
- In the second stage, use τ_{remain} and P_{remain} as the input. Treat each task in τ_{remain} as a heavy task. For a task $\tau_i \in \tau_{\text{remain}}$, if $Z_i = HI$, it will be assigned cluster m_i^{LO} and m_i^{HI}, when system is in LO-criticality mode and HI-criticality mode, respectively. If $Z_i = LO$, it will be assigned cluster m_i^{LO} when the system is in LO-criticality mode. The work-conserving scheduler is used at the cluster level. In the second stage, we use *simulated annealing* to find such an allocation of m_i^{LO} and m_i^{HI}.

The general task allocation framework is shown as algorithm 5. Because the simulated annealing algorithm takes relatively longer time than the partitioned scheduling algorithm, we decided to first apply partitioned scheduling algorithm to allocate light tasks (line 1). When performing partitioned scheduling in the first stage, we need to select part of subset of processors to form P_{ps}, here we use simulated anneal-

Algorithm 5: Task_allocation_framework(τ, P)

Input: The task set τ and the processor set P.

1 $P_{\text{allocation}}$, τ_{success} = Allocate_light_task(P_{ps}, τ, π); // Algorithm 7
2 $P_{\text{remain}} = P \setminus P_{\text{allocation}}$;
3 $\tau_{\text{remain}} = \tau \setminus \tau_{\text{success}}$;
4 **if** $P_{\text{remain}} == \emptyset$ and $\tau_{\text{remain}} == \emptyset$ **then**
  // All tasks are successfully scheduled by partitioned scheduling algorithm
5 return success;
6 **end**
7 isFeasible = Allocate_heavy_task(P_{remain}, τ_{remain}); // Algorithm 9
8 **if** isFeasible == True **then**
  // Find a feasible task allocation by simulated annealing algorithm
9 return success;
10 **end**
11 return failure;

ing to find a best one [1]. The partitioned framework takes a set of heuristic policies π to make sure the processors are used efficiently. The tasks that can not be assigned a processor by using this framework, we will treat it as a "heavy" task and put it into the remaining assignment framework.

Line 2 to line 6 is to get the input of the second stage and check whether the task set τ is already schedulable by performing the partitioned framework. Then simulated annealing algorithm is used to find a feasible allocation (line 8). If a feasible allocation is found, declare the task set is schedulable and return (lines 8 - 10). If all the works have been done but still can not find a feasible allocation, declare the task set is unschedulable and return (line 11).

6.2 Partitioned scheduling for light tasks on uniform platform

We use partitioned scheduling for schedule light tasks. However, find an feasible allocation pattern for partitioned scheduling is NP-complete [13]. So there cannot be any polynomial-time algorithm for finding an optimal partition of a set of tasks unless $P = NP$. Some heuristic policies are used to find an approximate solution.

We want to know whether one heuristic is better than the other or not, we use a concept called *Quality of Partition*, to determine which heuristic to apply from a set of different alternative heuristics at each stage of the partitioned algorithm. Section 6.2.1 presents the definition of the Quality of Partition. Section 6.2.2 presents the details of the each heuristic policy and Section 6.2.4 presents how to form the subset of processors to perform the partitioned scheduling.

[1] This part is described in detail at section 6.2.4

6.2.1 Quality of partition

A metric *QoP* is defined to measure the quality of partitioned scheduling under specific heuristics.

Definition 8. *Assume given a task set τ and a processor set P. The task set τ is a light task set for the processor set P according to the definition 7 and is scheduled successfully on the processor set P by EDF-VD. All the processors in the process set P are occupied by at least one task. For each processor $p_i \in P$, let $\tau(p_i)$ denote the set of tasks that are assigned to it. The QoP is defined as:*

$$\begin{aligned} QoP &= \frac{|\tau|}{|P|} \sum_{i=1}^{|P|} \frac{U_i^{use}}{0.75\delta_i} \\ &\Leftrightarrow \frac{4|\tau|}{3|P|} \sum_{i=1}^{|P|} \frac{U_i^{use}}{\delta_i} \end{aligned} \tag{6.1}$$

where $U_i^{use} = \max\left\{U_{LO}^{LO,p_i}(\tau(p_i)) + U_{HI}^{LO,p_i}(\tau(p_i)), U_{HI}^{HI,p_i}(\tau(p_i))\right\}$.

Since the tasks assigned to the processor are required to be schedulable, $U_i^{use} \leq 3/4$ (due to Theorem 3). The QoP measure the degree the processor cluster is utilized. If a heuristic policy can allocate as many light tasks as possible with as few processors, it will have a large value of QoP.

6.2.2 Heuristics

The heuristic policies aim to make the partitioned scheduling algorithm get the maximum QoP on a given processor set. The heuristic policies are combined by two sub-heuristic strategies. The first sub-heuristics concerns the order in which the processors are selected for testing whether another task can be added. The second sub-heuristics concerns the initial ordering of the taskset, that is, the order in which the tasks will be selected for schedulability testing on each processor.

6.2.2.1 Processor selection heuristics

The following heuristics will be considered to guide the processor selection:

- First Fit in increasing order (FFI): The processors are sorted in order of increasing speed, and the selected task is assigned to the first processor in which it fits according to the used schedulability test.
- First Fit in decreasing order (FFD): The opposite of FFI, the processors are ordered according to their speed in decreasing order.
- Best Fit (BF): The processor with the highest utilization is tested first, then the one with the second-highest utilization, etc.
- Worst Fit (WF): The opposite of BF, the processors, are ordered according to the highest remaining capacity.

6.2.2.2 Initial task ordering heuristics

The following ordering of task heuristics will be evaluated:

- Increasing utilization (IU): The tasks are ordered by increasing utilization at LO-criticality mode.
- Decreasing utilization (DU): The opposite of IU, the order of tasks decreases according to their utilization at HI-criticality mode.
- Decreasing criticality (DC): The tasks are ordered according to decreasing criticality level. In the case of equal criticality levels, decreasing utilization is used.
- Increasing criticality (IC): The tasks are ordered according to the increasing criticality level. In case of equal criticality levels, increasing utilization is used.
- Random (RAND): The tasks are not ordered at all.

6.2.3 Partitioned framework

The partition framework is shown as algorithm 6. This algorithm takes the set of given processors P_{ps}, the task set τ and the heuristic policy set π as the input. Every heuristic strategy $\pi_i \in \pi$ is combined by the processor selection heuristics and task initial orderings heuristics which are presented in section 6.2.2. We do not require any task that can be assigned a processor by this partitioned framework. The tasks that can not be assigned a processor by using this framework, we will treat it as a "heavy" task and put it into the remaining assignment framework.

At first, get the light task set τ_{exam} for the given processor set P_{ps} according to the definition 7 (line 1). Line 4 to line 18 is used a specific heuristic policy to perform the partitioned scheduling algorithm. Sort the processor set P_{ps} and task set τ_{exam} according to the specific heuristic policy. For each task $\tau_{i,\text{exam}}$, traverse the processor set P_{ps}. Once find a processor p_j that can host the task $\tau_{i,\text{exam}}$, store the information. When the partitioned scheduling algorithm under the specific policy is finished, check whether it has a better QoP (lines 20 - 24). Finally, return the $P_{\text{allocation}}$ and τ_{success}, which are obtained by the heuristics with the best QoP.

6.2.4 Heuristics for forming processor set

In the first stage, we need to select a set processor P_{ps} to do partitioned scheduling. However, there are total $2^{|P|}$ different processor set P_{ps}, when given a initial processor set P. Here, we use simulated annealing to find the "best" processor set P_{ps}.

The initial solution for P_{ps} is set as P. A neighbor of a processor set P_{ps} is defined as randomly deleting a processor from P_{ps} or adding a new processor into P_{ps}. The objective function is defined as the maximum value of QoP. The pseudo-code is shown as algorithm 7. We want to find an allocation scheme with the best QoP. If a better solution is generated, $(\text{QoP}' - \text{QoP}) > 0$, $e^{(\text{QoP}'-\text{QoP})/T} > 1$ is satisfied. That means a better allocation scheme is always accepted. If a worse solution is generated, there is a slight possibility of accepting it (lines 8 - 13).

Algorithm 6: Partitioned_framework(P_{ps}, τ, π)

Input: The set of processor to be checked P_{ps}, with speed $\left\{\delta^1, \delta^2, ...\delta^{|P|}\right\}$, the task set τ and the heuristic policy set π

Output: The processor set $P_{\text{allocation}}$ that is used to host the light tasks, the task set τ_{success} which has been successfully partitioned and the quality of partition QoP

```
Traverse the processor set P_ps and the task set τ to get the light task set
  τ_exam according to the definition 7;
QoP = 0;
for k = 1 to |π| do
    Sort the processor set P_ps and the light task set τ_exam according to the
     heuristic policy π_k;
    P_allocation_tmp = ∅;
    τ_success_tmp = ∅;
    for i = 1 to |τ_exam| do
        for j = 1 to |P_ps| do
            if Equation 5.41 is satisfied then
                // processor p_j can host task τ_{i,exam} according to Theorem 3
                assign task τ_{i,exam} to processor p_j;
                if p_j ∉ P_allocation_tmp then
                    Add the processor p_j to P_allocation;
                end
                Add the task τ_{i,exam} to τ_success;
                break;
            end
        end
    end
    Based on P_allocation and τ_success compute the QoP_current according to
     equation 6.1;
    if QoP_current > QoP then
        QoP = QoP_current;
        P_allocation = P_allocation_tmp;
        τ_success = τ_success_tmp;
    end
end
```

Algorithm 7: Allocate_light_task(P, τ, π)

Input: The inital processor set P, task set τ, the heuristic policy set π
Output: The processor set $P_{\text{allocation}}$ that is used to host the light tasks and the task set τ_{success} which has been successfully partitioned

```
P_ps = P;
P_allocation, τ_success, QoP = Partitioned_framework(P_ps, τ, π); // Algorithm 6
Define maximum iteration steps k_max;
for k = 0 to k_max − 1 do
    T = 1 − k/k_max;
    P'_ps = Pick a random neighbour of P_ps;
    P'_allocation, τ'_success, QoP' = Partitioned_framework(P'_ps, τ, π);
    if e^((QoP'−QoP)/T) ≥ random(0,1) then
        P_ps = P'_ps;
        P_allocation = P'_allocation;
        τ_success = τ'_success;
        QoP = QoP';
    end
end
```

6.3 Search an allocation solution for heavy tasks based on simulated annealing

Assume after the partitioned framework, the task set is $\tau_{\text{remain}} = \tau \setminus \tau_{\text{success}}$ and the processor set is $P_{\text{remain}} = P \setminus P_{\text{allocation}}$. At this stage, for each task $\tau_i \in \tau_{\text{remain}}$, if $Z_i = HI$, it will be assigned cluster m_i^{LO} and m_i^{HI}, when system is in LO-criticality mode and HI-criticality mode respectively. If $Z_i = LO$, it will be assigned cluster m_i^{LO} when the system is in LO-criticality mode. m_i^{LO} and m_i^{HI} are initialized by an empty set. We use a simulated annealing to determine m_i^{LO} and m_i^{HI}.

When using simulated annealing, objective function, initial solution and neighbourhood space of a solution are needed. Section 6.3.1 presents how to formulated the problem allocate heavy tasks as a optimization problem and the objective function. Section 6.3.2 presents how to generate the initial solution. The neighbourhood space is defined in section 6.3.3. Finally, section 6.3.4 gives the total framework of the algorithm for the second stage.

6.3.1 Objective function

Searching a allocation solution which makes the task set schedulable is a feasibility problem. We use the total lateness of tasks to define the objective function.

$$cost = \sum_{\tau_i \in \tau_{\text{remain}}} |R_i - D_i| \tag{6.2}$$

where R_i is the response time bound for the task τ_i computed according to equation 5.10 and equation 5.28 for LO-criticality task and HI-criticality task, respectively,

based currently given cluster m_i^{LO} and m_i^{HI}. D_i is the deadline of task τ_i.

Allocating heavy tasks can be formulated as equation 6.3.

$$\begin{aligned} \text{minimize} \quad & \sum_{\tau_i \in \tau_{\text{remain}}} |R_i - D_i| \\ \text{subject to} \quad & m_i^{LO} \subseteq m_i^{HI} \qquad \text{if } Z_i = \text{HI} \\ & \bigcup_{\tau_i \in \tau_{\text{remain}} \wedge Z_i = \text{HI}} m_i^{HI} = P_{\text{remain}} \\ & \bigcup_{\tau_i \in \tau_{\text{remain}} \wedge Z_i = \text{HI}} \Delta m_i^{HI} = \bigcup_{\tau_i \in \tau_{\text{remain}} \wedge Z_i = \text{LO}} m_i^{LO} \end{aligned} \tag{6.3}$$

where $\Delta m_i^{HI} = m_i^{HI} \setminus m_i^{LO}$.

The first condition is to make the processors be used sufficiently. When the system is transformed to HI-criticality mode, the LO-criticality tasks will be discarded, and the clusters assigned to them will be reassigned to the HI-criticality tasks. The second condition and the third condition are used to guarantee that resource constraint is not exceeded, and the system behavior is satisfied.

6.3.2 Initial solution

The initial solution is obtained by the greedy algorithm. The algorithm is shown as algorithm 8.

The tasks are sorted based on their total workloads at LO-criticality mode, and the processors are sorted based on their speeds (line 1). For each task τ_i, add the processor gradually to its m_i^{LO} until it becomes schedulable when the system is in LO-criticality mode. If we have travesed all the processors but the task τ_i is still unschedulable, those processors will also assigned to form the cluster m_i^{LO}. If a task τ_i is LO-criticality task, every time a processor p is added into its m_i^{LO}, select a random HI-criticality task τ_j and add the processor p to the cluster Δm_j^{HI}, to make sure that the third condition is always satisfied (lines 6 - 8). When all tasks are schedulable and there is some processor unused, assign those processors to the cluster m_j^{LO} of a random HI-criticality task τ_j, to make sure the second condition in the equation 6.3 is satisfied (lines 15 - 18).

After the greedy algorithm, if all tasks are schedulable. Return schedulable directly. If some task is still unschedulable, perform the simulated annealing algorithm, which is shown as algorithm 9.

6.3.3 Generate a legal neighbor

To make sure the three conditions in equation 6.3 are always satisfied and follow the design principle that a neighborhood is the set of all potential solutions that differ from the current state by the minimum possible extend. For a allocation scheme s, we define follow five rules to generate a legal neighbor:

1. For an arbitrary pair of HI-criticality tasks τ_i and τ_j: $m_i^{LO} \xrightarrow{a} m_j^{LO}$
2. For an arbitrary pair of LO-criticality tasks τ_i and τ_j: $m_i^{LO} \xrightarrow{a} m_j^{LO}$
3. For an arbitrary pair of HI-criticality tasks τ_i and τ_j: $\Delta m_i^{HI} \xrightarrow{a} \Delta m_j^{HI}$

Algorithm 8: Initial_solution($P_{\text{remain}}, \tau_{\text{remain}}$)

Input: The task set τ_{remain} and the processor set P_{remain}

```
1  Sort the available processors based on their speed and the tasks based on
     their total workloads at LO-criticality mode in descending order;
2  k = 0; // use index k to track available processor
3  foreach τ_i ∈ τ_remain do
4      while k < |P_remain| do
5          Add processor p_k to the cluster m_i^LO;
6          if Z_i == LO then
               // Make sure the third condition in equation 6.3 is always
                  satisfied
7              Choose a HI-criticality task τ_j randomly, and add processor p_k to
                the cluster Δm_j^HI;
8          end
9          k = k + 1;
10         if Task τ_i is schedulable then
               // If Z_i = LO, use theorem 1. If Z_i = HI, use theorem 2
11             break;
12         end
13     end
14 end
15 while k < |P_remain| do
16     Choose a HI-criticality task τ_j randomly, and add processor p_k to the
        cluster m_j^LO;
17     k = k + 1;
18 end
```

4. For an arbitrary pair of HI-criticality task τ_i and LO-criticality task τ_j: $m_i^{LO} \xrightarrow{a} m_j^{LO}$ and select an arbitrary HI-criticality task τ_k, add processor a into Δm_k^{HI}
5. For an arbitrary pair of LO-criticality task τ_i and HI-criticality task τ_j: $m_i^{LO} \xrightarrow{a} m_j^{LO}$ and find the HI-criticality task τ_k, where the processor $a \in \Delta m_k^{HI}$, delete processor a from Δm_k^{HI}

The operator $\xrightarrow{a}$ is defined as equation 6.4:

$$A \xrightarrow{a} B \iff A = A \setminus \{a\} \text{ and } B = B \cup \{a\} \tag{6.4}$$

Where A and B is a set, and $A \xrightarrow{a} B$ is valid when $a \in A$.

The following two rules can also generate a legal neighbor, but the neighbors generated by rule 6 and rule 7 are as same as rule 4 and rule 5. So we just use five rules to generate neighbors. Those five rules are shown in figure 6.1.

6. For an arbitrary pair of HI-criticality task τ_i and τ_j: $m_i^{LO} \xrightarrow{a} \Delta m_j^{HI}$ and select an arbitrary LO-criticality task τ_k, add processor a into m_k^{LO}.
7. For an arbitrary pair of HI-criticality task τ_i and τ_j: $\Delta m_i^{HI} \xrightarrow{a} m_j^{LO}$ and find the LO-criticality task τ_k, where the processor $a \in m_k^{LO}$, delete processor a from m_k^{LO}.

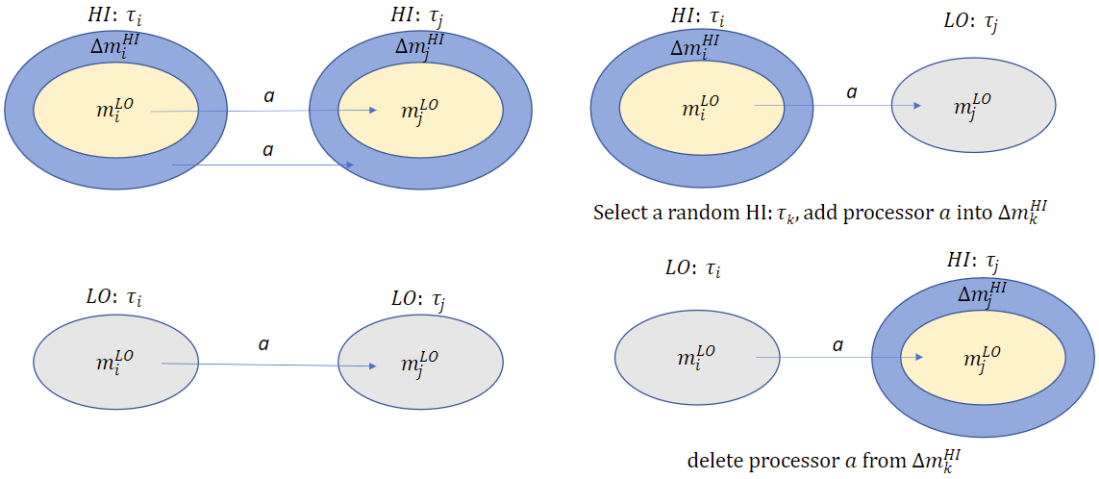

Figure 6.1: Neighbor Generation

6.3.4 Framework for the simulated annealing algorithm

The framework of simulated annealing is shown as algorithm 9. We want to find a feasible solution, and the value of the objective function for a feasible solution might be small. If a better solution is generated, $\Delta cost < 0$, $e^{-\Delta Cost/T} > 1$ is satisfied. That means a better solution is always accepted. If a worse solution is generated, there is a slight possibility of accepting it (lines 18 - 21).

Algorithm 9: Allocate_heavy_task(P_{remain}, τ_{remain})

Define maximum iteration step k_{max};
Getting an initial solution S by the greedy algorithm; // Algorithm 8
if The initial solution S is a feasible solution **then**
    // According to the schedulibility test (Theorem 1 and Theorem 2)
    Declare the task set is schedulable;
    return;
end
Mincost = cost(S) ;
k = 0;
while $k < k_{\max}$ **do**
    $T = 1 - k/k_{\max}$;
    // The 5 rules presented in section 6.3.3
    $S^{'}$ = randomly chosen from the neighbor of S;
    if The $S^{'}$ is a feasible solution **then**
        // According to the schedulibility test (Theorem 1 and Theorem 2)
        Declare the task set is schedulable;
        return;
    end
    CurrentCost = cost($S^{'}$);
    $\Delta cost$ = CurrentCost - MinCost;
    if $e^{-\Delta Cost/T} \geq$ random(0,1) **then**
        $S = S^{'}$;
        MinCost = CurrentCost;
    end
end

7

Extension to elastic mixed criticality model

In the previous chapter, all the LO-criticality task are assumed to be discarded when system switches to high criticality mode. This causes a service abrupt problem, since some LO-criticality tasks may provide very useful function to the system. . In this chapter, this assumption is relaxed and the schedulability tests and task allocation are discussed under a new model called elastic mixed-criticality model, , which allows the low-critical tasks to continue execution at HI-criticality mode.

7.1 Elastic mixed criticality model

To address the service abrupt problem for low-criticality tasks in mixed-criticality scheduling algorithms, Su and Zhu in [14] introduce an *Elastic Mixed-Criticality* task model, where the *elastic model* [15] is used to model low criticality tasks. When the mixed criticality system transforms to HI-criticality mode, the period of low-criticality tasks will become larger such that low-criticality tasks continue to provide service (less frequently). That makes the systems has the graceful degradation.

The major difference between the elastic mixed criticality task model and the traditional mixed criticality task model is how to model the low-criticality task. For each low-criticality task τ_i in the elastic mixed criticality system, it has a additional *maximum* period $T_i^{max} \geq T_i$ to represent its minimum service requirements.

In this thesis, we consider the implicit deadline task. That means, in the low criticality model, low criticality task τ_i has a deadline $D_i = T_i$ and in the high criticality model, low criticality task τ_i has a extended deadline $D_i^{max} = T_i^{max}$.

Definition 9. *A set of elastic mixed criticality tasks is said to be schedulable, if the following two conditions can be satisfied:*

- *During the low-criticality mode, all the tasks meet their deadlines.*
- *After the system transitions into the high-criticality mode, all high-criticality tasks can meet their deadlines and all low-criticality tasks can meet their extended deadline.*

7.2 Schedulability test

We need to develop new schedulablity test for HL, LH and LL tasks. Note that LH is considered even if it is a high-critical task since EDF-VD uses a schedulability test for each uniprocessor where some light HI-criticality task may be allocated. The

schedulability test for HH task (Theorem 2) is still usable under elastic mixed criticality model because its behavior under elastic mixed criticality systems is remain same.

7.2.1 Response time analysis for HL task

Assume task τ_i is a HL task. Let m_i^{LO} and m_i^{HI} denote the cluster assigned to it in LO-criticality and HI-criticality mode respectively. To utilize the processor efficiently, we set that $m_i^{HI} \subseteq m_i^{LO}$. From Lemma 1, we can have: $\lambda_i^{LO} \geq \lambda_i^{HI}$ and $S_i^{LO} \geq S_i^{HI}$. The virtual deadline D_i^v for it is assigned as follows:

$$D_i^v = \frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{S_i^{LO}} \tag{7.1}$$

Theorem 4. *Consider a pair of cluster $\left(m_i^{LO}, m_i^{HI}\right)$, such that the HL task τ_i is assigned dedicated cluster m_i^{LO} and m_i^{HI} for the LO- and HI-critical mode, respectively, where $m_i^{HI} \subset m_i^{LO}$. Each job of task τ_i meets its deadline in all correct states if the following condition is satisfied:*

$$\frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{\min\left\{\omega_i S_i^{HI}, S_i^{LO}\right\}} \leq D_i \tag{7.2}$$

where $\omega_i = T_i^{max}/T_i$.

Proof. Without loos of generality, we consider a job J_i, which is released at time instant 0, of HL task τ_i. The execution of job J_i happens in three possible scenarios: (1) Stable LO-criticality mode, (2) Stable HI-criticality mode, and (3) mode switching. A stable mode means that there is no system state change during the execution of job J_i. This theorem is proved by showing that job J_i can meet its deadline for all these three possible execution scenarios if condition 7.2 satisfied.

Stable LO-criticality mode. During the stable LO-criticality, the task τ_i is executed on cluster m_i^{LO}. Since its job J_i executes entirely in stable mode, it should signal completion at or before its virtual deadline. So the upper bound of the response time R_{LO} for HL task in the stable LO-criticality should satisfy condition 7.3.

$$R_{LO} \leq \frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{S_i^{LO}} = D_i^v \tag{7.3}$$

Stable HI-criticality mode. During the stable HI-criticality mode, the task τ_i is executed on cluster m_i^{HI}. Since its job J_i executes entirely in stable mode, it should signal completion at or before its extended deadline. So the upper bound of the response time R_{HI} for HL task in the stable HI-criticality mode should satisfy condition 7.4.

$$R_{HI} \leq \frac{C_i^{HI} + \lambda_i^{HI} L_i^{HI}}{S_i^{HI}} \leq \omega_i D_i \tag{7.4}$$

Mode Switching The transition occurred during the execution of job J_i. Let R_s denote the upper bound of response time for this case. Assume the execution time for J_i at Hi-criticality mode is $t^{'}$. It is clear that $0 < t^{'} < R_s$.

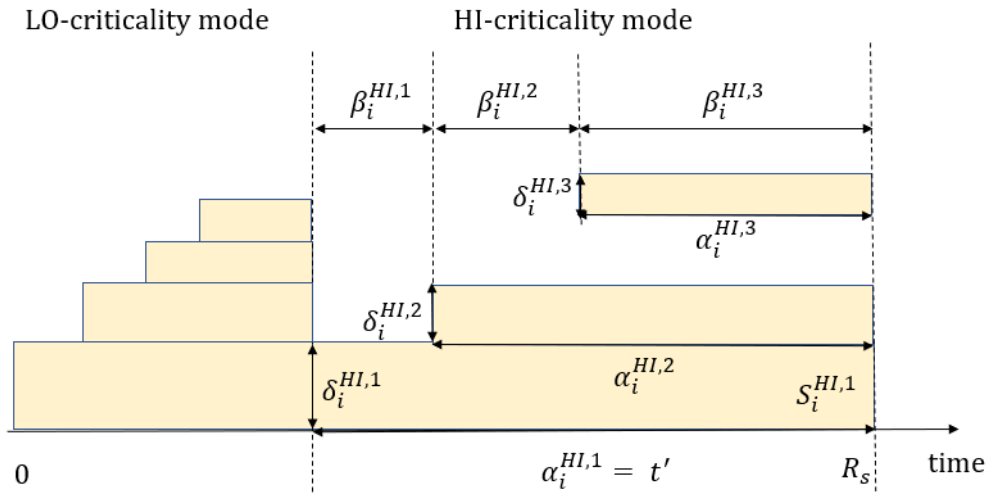

Figure 7.1: A possible allocation scheme

As shown in Figure 7.1, we can get the total workload and the critical length which have been finished at LO-criticality mode as equation 7.5 shown.

$$\begin{aligned} C_{\text{finished}} &= C_i^{LO} - \sum_{x=1}^{|m_i^{HI}|} \beta_i^{HI,x} S_i^{HI,x} \\ L_{\text{finished}} &\leq L_i^{LO} - \sum_{x=1}^{|m_i^{HI}|-1} \beta_i^{HI,x} \delta_i^{HI,x} \end{aligned} \tag{7.5}$$

So we can get

$$\begin{aligned} R_s &\leq t' + \frac{C_{\text{finished}} + \lambda_i^{LO} L_{\text{finished}}}{S_i^{LO}} \\ &\leq t' + \frac{C_i^{LO} - \sum_{x=1}^{|m_i^{HI}|} \beta_i^{HI,x} S_i^{HI,x} + \lambda_i^{LO} \left(L_i^{LO} - \sum_{x=1}^{|m_i^{HI}|-1} \beta_i^{HI,x} \delta_i^{HI,x} \right)}{S_i^{LO}} \end{aligned} \tag{7.6}$$

From Lemma 1, we can have $\lambda_i^{LO} \geq \lambda_i^{HI}$. Combine the definition of λ in equation 5.1, we know for all $1 \leq x \leq |m_i^{HI}|$, the condition 7.7 is satisfied.

$$\frac{S_i^{HI} - S_i^{HI,x}}{\delta_i^{HI,x}} \leq \lambda_i^{HI} \leq \lambda_i^{LO} \tag{7.7}$$

Let $B = \sum_{x=1}^{|m_i^{HI}|} \beta_i^{HI,x} S_i^{HI,x} + \lambda_i^{LO} \sum_{x=1}^{|m_i^{HI}|-1} \beta_i^{HI,x} \delta_i^{HI,x}$. From condition 7.7, we can get

$$\begin{aligned} B &\geq \sum_{x=1}^{|m_i^{HI}|} \beta_i^{HI,x} S_i^{HI,x} + \sum_{x=1}^{|m_i^{HI}|-1} \beta_i^{HI,x} \left(S_i^{HI} - S_i^{HI,x} \right) \\ &\Leftrightarrow \beta_i^{HI,|m_i^{HI}|} S_i^{HI} + \sum_{x=1}^{|m_i^{HI}|-1} \beta_i^{HI,x} S_i^{HI} \\ &\Leftrightarrow S_i^{HI} \sum_{x=1}^{|m_i^{HI}|} \beta_i^{HI,x} \end{aligned} \tag{7.8}$$

Since $\sum_{x=1}^{|m_i^{HI}|} \beta_i^{HI,x} = t^{'}$, we can get

$$-B \leq -S_i^{HI} t^{'} \tag{7.9}$$

Combining condition 7.6 and 7.9 gives

$$\begin{aligned} R_s &\leq t^{'} + \frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO} - S_i^{HI} t^{'}}{S_i^{LO}} \\ &\Leftrightarrow \frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{S_i^{LO}} + (1 - \frac{S_i^{HI}}{S_i^{LO}}) t^{'} \\ &\text{(since } 0 < t^{'} < R_s \text{ and } S_i^{HI} \leq S_i^{LO}\text{)} \\ &\leq \frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{S_i^{LO}} + (1 - \frac{S_i^{HI}}{S_i^{LO}}) R_s \end{aligned} \tag{7.10}$$

So we can get

$$R_s \leq \frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{S_i^{HI}} \tag{7.11}$$

To make the task τ_i become schedulable, we can make

$$\max\{R_{LO}, R_{HI}/\omega_i, R_s/\omega_i\} \leq D_i \tag{7.12}$$

Since it is a HL task, $C_i^{HI} = C_i^{LO}$ and $L_i^{HI} = L_i^{LO}$. And $\lambda_i^{HI} \leq \lambda_i^{LO}$, we can have

$$\frac{C_i^{HI} + \lambda_i^{HI} L_i^{HI}}{\omega_i S_i^{HI}} \leq \frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{\omega_i S_i^{HI}} \tag{7.13}$$

That means $R_{HI}/\omega_i \leq R_s/\omega_i$.

So when $\frac{C_i^{LO} + \lambda_i^{LO} L_i^{LO}}{\min\{\omega_i S_i^{HI}, S_i^{LO}\}} \leq D_i$, each job of task τ_i meets its deadline in all correct states.

□

7.2.2 Utilization bound for LH and LL tasks

Liu et al. analysis the elastic mixed criticality system under EDF-VD scheduling in [16]. They gives two sufficient tests for sequential elastic mixed criticality tasks. Because LH and LL tasks are light tasks, those two sufficient tests are still usable here.

Lemma 4. ***(Theorem 3 from [16])*** *Given an elastic mixed criticality task set τ and a processor p with speed δ^p, if*

$$\begin{gathered} \frac{U_{HI}^{LO,p}(\tau)}{1 - U_{LO}^{LO,p}(\tau)} \leq \frac{1 - \left(U_{HI}^{HI,p}(\tau) + U_{LO}^{HI,p}(\tau)\right)}{U_{LO}^{LO,p}(\tau) - U_{LO}^{HI,p}(\tau)} \\ U_{HI}^{HI,p}(\tau) + U_{LO}^{HI,p}(\tau) < 1 \text{ and } U_{LO}^{LO,p}(\tau) < 1 \text{ and } U_{LO}^{LO,p} > U_{LO}^{HI,p}(\tau) \end{gathered} \tag{7.14}$$

then this elastic mixed criticality task set τ can be scheduled by EDF-VD with a deadline scaling factor x arbitrarily chosen in the following range

$$x \in \left[\frac{U_{HI}^{LO,p}(\tau)}{1 - U_{LO}^{LO,p}(\tau)}, \quad \frac{1 - \left(U_{HI}^{HI,p}(\tau) + U_{LO}^{HI}(\tau)\right)}{U_{LO}^{LO,p}(\tau) - U_{LO}^{HI,p}(\tau)} \right] \tag{7.15}$$

where $U_{HI}^{HI,p}(\tau)$, $U_{HI}^{LO,p}(\tau)$ *and* $U_{LO}^{LO,p}(\tau)$ *are computed according to definition 5 and*

$$U_{LO}^{HI,p} = \sum_{\tau_i \in \tau \wedge Z_i = LO} \frac{C_i^{HI}}{\delta_p T_i^{max}} \tag{7.16}$$

Theorem 5. ***(Theorem 3 from [16])*** *An elastic mixed criticality task set* τ *is schedulable upon a processor* p *with* δ^p*, if the following condition is satisfied.*

$$\max\left\{U_{LO}^{LO,p}(\tau) + U_{HI}^{LO,p}(\tau), U_{LO}^{HI,p}(\tau) + U_{HI}^{HI,p}(\tau)\right\} \leq \frac{3}{4} \tag{7.17}$$

The Lemma 4 can be used for determining the modified period and virtual deadline when using EDF-VD to schedule an elastic mixed criticality task set. And the Theorem 5 can be used to offline verify whether the task set is schedulable.

7.2.3 Summary

The summary for scheduling algorithm and schedulability test for four different types of elastic mixed criticality tasks is shown as table 7.1. Here assume that the LH and LL tasks are assigned on the processor p with speed δ^p.

Task Type	Criticality	Schedulability test	Scheduling algorithm
LH	High	Theorem 3	EDF-VD
LL	Low		
HH	High	Theorem 2	Work-conserving scheduling
HL	Low	Theorem 4	

Table 7.1: Scheduling algorithm and scheduability test for four different types of elastic mixed criticality tasks

7.3 Task allocation

The task allocation algorithm for elastic mixed criticality tasks is similar to traditional mixed criticality tasks. There are following difference:

The calculation for QoP. When computing the QoP according definition 8 in section 6.2.1, the way for calculating the U_i^{use} changes to

$$U_i^{\text{use}} = \max\left\{U_{LO}^{LO,p_i}(\tau(p_i)) + U_{HI}^{LO,p_i}(\tau(p_i)), U_{HI}^{HI,p_i}(\tau(p_i)) + U_{LO}^{HI,p_i}(\tau(p_i))\right\} \tag{7.18}$$

where p_i is the processor and $\tau(p_i)$ denotes the tasks that are assigned to it.

Schedulability test. In line 9 of algorithm 6, use equation 7.2 to check whether the task is schedulable upon the uniprocessor. In line 12 of algorithm 9, use Theorem 4 to check whether HL tasks are schedulable or not. When computing the value of the objective function for an allocation scheme, use equation 7.2 to get R_i if $Z_i = LO$.

The rules for generate a neighbor. Assume we want to allocate a set of elastic mixed criticality task τ upon the processor set P. After the first stage(algorithm 7),

the task set τ_{success} is successfully scheduled and the processor set $P_{\text{allocation}}$ is used to host the task set τ_{success}. Let $\tau_{\text{remain}} = \tau \setminus \tau_{\text{success}}$ and $P_{\text{remain}} = P \setminus P_{\text{allocation}}$.

In the second stage (algorithm 9), each LO-criticality task $\tau_i \in \tau_{\text{remain}}$ is assigned on the cluster m_i^{HI} and the cluster $m_i^{LO} = m_i^{HI} \cup \Delta m_i^{LO}$ at HI- and LO-criticality mode, respectively. And each HI-criticality task $\tau_j \in \tau_{\text{remain}}$ is assigned on the cluster m_j^{LO} and the cluster $m_j^{HI} = m_j^{LO} \cup \Delta m_j^{HI}$ at LO- and HI-criticality mode, respectively. The equation 6.3 in the section 6.3.1 is changed as equation 7.19.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{\tau_i \in \tau_{\text{remain}}} |R_i - D_i| \\
\text{subject to} \quad & m_i^{HI} \subseteq m_i^{LO} \qquad \text{if } Z_i = \text{LO} \\
& m_j^{LO} \subseteq m_j^{HI} \qquad \text{if } Z_j = \text{HI} \\
& \bigcup_{\tau_i \in \tau_{\text{remain}} \wedge Z_i = \text{LO}} m_i^{LO} = P_{\text{remain}} \setminus \bigcup_{\tau_j \in \tau_{\text{remain}} \wedge Z_j = \text{HI}} m_j^{LO} \\
& \bigcup_{\tau_i \in \tau_{\text{remain}} \wedge Z_i = \text{LO}} \Delta m_i^{LO} = \bigcup_{\tau_j \in \tau_{\text{remain}} \wedge Z_j = \text{HI}} \Delta m_i^{HI}
\end{aligned}
\tag{7.19}
$$

The first and second condition in equation 7.19 is to make the processors be used sufficiently. When the system is transformed to HI-criticality mode, part of the processors which are assigned to LO-criticality tasks at LO-criticality mode will be reassigned to HI-criticality tasks. The third and forth condition are used to guarantee that resource constraint is not exceeded, and the system behaviour is satisfied.

As figure 7.2 shown, we define eight rules to generate a legal neighbor.

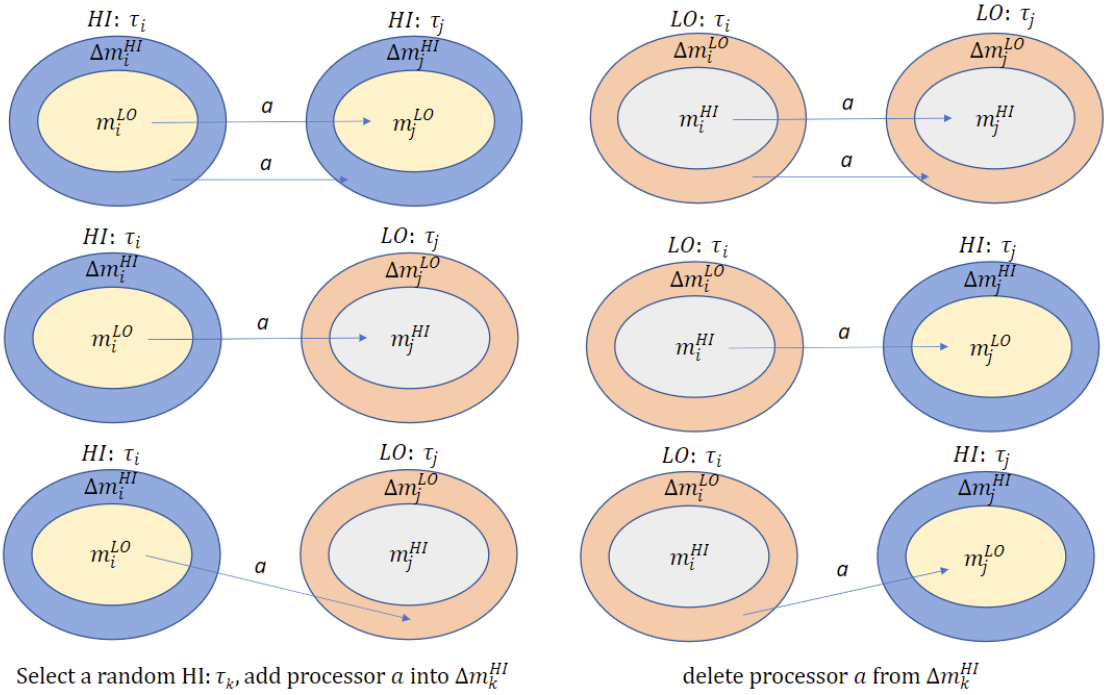

Figure 7.2: Neighbor Generation for elastic mixed criticality tasks

1. For an arbitrary pair of HI-criticality tasks τ_i and τ_j: $m_i^{LO} \xrightarrow{a} m_j^{LO}$

2. For an arbitrary pair of HI-criticality tasks τ_i and τ_j: $\Delta m_i^{HI} \xrightarrow{a} \Delta m_j^{HI}$
3. For an arbitrary pair of LO-criticality tasks τ_i and τ_j: $m_i^{HI} \xrightarrow{a} m_j^{HI}$
4. For an arbitrary pair of LO-criticality tasks τ_i and τ_j: $\Delta m_i^{LO} \xrightarrow{a} \Delta m_j^{LO}$
5. For an arbitrary pair of HI-criticality task τ_i and LO-criticality task τ_j: $m_i^{LO} \xrightarrow{a} m_j^{HI}$
6. For an arbitrary pair of LO-criticality task τ_i and HI-criticality task τ_j: $m_i^{HI} \xrightarrow{a} m_j^{LO}$
7. For an arbitrary pair of HI-criticality task τ_i and LO-criticality task τ_j: $m_i^{LO} \xrightarrow{a} \Delta m_j^{LO}$ and select an arbitrary HI-criticality task τ_k, add processor a into Δm_k^{HI}
8. For an arbitrary pair of LO-criticality task τ_i and HI-criticality task τ_j: $\Delta m_i^{LO} \xrightarrow{a} m_j^{LO}$ and find the HI-criticality task τ_k, where the processor $a \in \Delta m_k^{HI}$, delete processor a from Δm_k^{HI}

7.4 Summary

This chapter presents the elastic mixed criticality systems, and develops a up bound of the parallel task's response time. The difference in the task allocation algorithm for elastic and traditional mixed criticality tasks is also shown in this chapter. In the next chapter, we will evaluate the performance of the task allocation algorithm.

8

Evaluation

To quantitatively evaluate our algorithm, we generate synthetic DAG sets and check whether all the DAGs of a set of DAGs can meet their deadlines. Section 8.1 and section 8.2 introduces the task set generation algorithm, and section 8.3 presents the simulation results.

8.1 Uniform platform generation

To generate the uniform platform, two parameters are needed. The first one is the number of processors m. We set $m \in \{16, 32, 64, 128\}$. The second one is the total capacity of the platform S_m. We set $S_m = 0.8m$.

Once the number of processor m and the total capacity of the platform S_m is determined, we need to generate the processor speed δ for each processor. The sum of the processor speed should be equal to the total capacity of the platform. The Dirichlet-Rescale (DRS) algorithm [17] can be used.

To make sure that a processor with the unit speed is in the results, we use $(m-1)$, $(S_m - 1)$ as the input of the DRS and each value is limited in $[0, 1]$. After it outputs a sequence of $(m-1)$ values, we add the value 1 into the sequence to form the final results.

8.2 Task set generation

After generation of the platform, we can generate the task set. Since our schedulability test depends on the total work, the critical-path length. We directly generate theses two parameters. Let U^{LO} and U^{HI} respectively denote the total nominal utilization of all the tasks and total overload utilization of all the HI-criticality tasks in a randomly generated taskset τ such that $U^{LO} = \sum_{\tau_i \in \tau} u_i^{LO}$ and $U^{HI} = \sum_{\tau_i \in \tau \wedge Z_i = HI} u_i^{HI}$. Let $U_B = \max\left\{U^{HI}/S_m,\ U^{LO}/S_m\right\}$ denote the upper bound of the total system utilization. Note that $U_B \leq 1$ is a necessary condition for taskset τ is schedulable on the given platform.

To generate a task set τ, we control 6 parameters, which is similar to the task set generation algorithm presented by Pathan in [18]. They are shown as fellow:

- p^{hu}: This parameter controls the proportion of heavy tasks in a task set. Set $p^{hu} \in \{0.4, 0.5, ..., 1.0\}$.
- p^{hc}: This parameter controls the proportion of HI-criticality tasks in a task set. Set $p^{hc} \in \{0.4, 0.5, ..., 1.0\}$.

- u_{max}: This parameter defines the upper bound of overload utilization of each heavy task. Set $u_{max} \in \{2, 4, ..., 10\}$.
- P_{max}: This parameter defines the upper bound of the ratio of the deadline and overload critical-path length of a task $\tau_i \in \tau$, such that $1 \leq D_i/L_i^{HI} \leq P_{max}$. Set $P_{max} \in \{2.0, 2.5, ..., 5.0\}$.
- R_{max}: This parameter defines the upper bound of the ratio of overload and nominal utilization of a task $\tau_i \in \tau$, such that $1 \leq u_i^{HI}/u_i^{LO} \leq R_{max}$. Set $R_{max} \in \{2.0, 2.5, ..., 5.0\}$.
- U_B: Total system utilization U_B is setting as $U_B \in \{0.2, 0.3, ..., 1.0\}$.

We consider different combinations of the those 6 parameters to generate a task set. For each combination, we generate 500 task sets to form the test set. The task $\tau_i \in \tau$ is generated as follows (each parameter is uniformly selected from the specific range, and the generated the total workload, as well as the critical path length, is the value on the processor p_1 with unit speed):

- Task period $D_i = T_i$ is chosen in the range $[10, 1000]$.
- Generate a random number p_i^u in the range $[0, 1]$. If $p_i^u \leq p^{hu}$, then τ_i will be a heavy task on the processor p_1 and its overload utilization u_i^{HI,p_1} is drawn in the range $[1.0, u_{\max}]$; otherwise, it will be a light task on the processor p_1 and its overload utilization u_i^{HI,p_1} is chosen in the range $[0.02, 1]$. The overload total workload of τ_i is computed as $C_i^{HI} = u_i^{HI,p_1} \times T_i$.
- Generate a random number P_i in the range $[1, P_{\max}]$. Then the overload critical path length is computed as $L_i^{HI} = T_i/P_i$.
- Generate a random number p_i^c in the the range $[0, 1]$. If $p_i^c \leq p^{hc}$, then τ_i will be a HI-criticality task; otherwise it will be a LO-criticality task. If τ_i is HI-criticality task, then generate a random number R_i in the range $[1, R_{\max}]$; otherwise $R_i = 1$.
- The nominal total workload and critical path length are computed as $C_i^{LO} = C_i^{HI}/R_i$ and $L_i^{LO} = L_i^{HI}/R_i$.

Generate a task according to above steps repeatedly and add it to the task set until $\max\left\{U^{HI}/S_m,\ U^{LO}/S_m\right\} > U_B - 0.1$. If add a generated task to the task set make $\max\left\{U^{HI}/S_m,\ U^{LO}/S_m\right\} > U_B$, discard it and generate a new one.

According to the above task set generation procedure, the U_B of each task set is in the range $(U_B - 0.1, U_B]$. It is accepted because we set the increment step of U_B in our experiment is 0.1.

8.3 Results

We use *acceptance ratio* to measure the performance. The acceptance ratio is the fraction of task sets that can be schedulable on the given uniform multiprocessor platform out of the test set, which contains 500 task sets. Note in this section that each time we vary one parameter, the other parameters are fixed. The fixed values of the parameters are set as $m = 64$, $p^{hc} = 0.5$, $p^{hu} = 0.5$, $u^{max} = 2.0$, $P_{max} = 4.0$ and $R_{max} = 2.0$.

8.3.1 Impact of uniform platform and different DAGs

In this section, we evaluate the performance of our algorithm by varying the parameters for generating the platform and task set.

1) Impact of m

Figure 8.1 shows the change of acceptance ratio by varying the number of processors.

Generally, as the number of processors increases, the acceptance ratio decreases. Because the total capacity is set as $S_m = 0.8m$ and $U_B = \max\left\{U^{HI}/S_m,\ U^{LO}/S_m\right\}$, so when m becomes larger, the number of tasks in the task set will also become larger. The number of different allocation patterns becomes larger. It makes the search space of our algorithm become larger, which makes it harder for our algorithm to find a feasible solution.

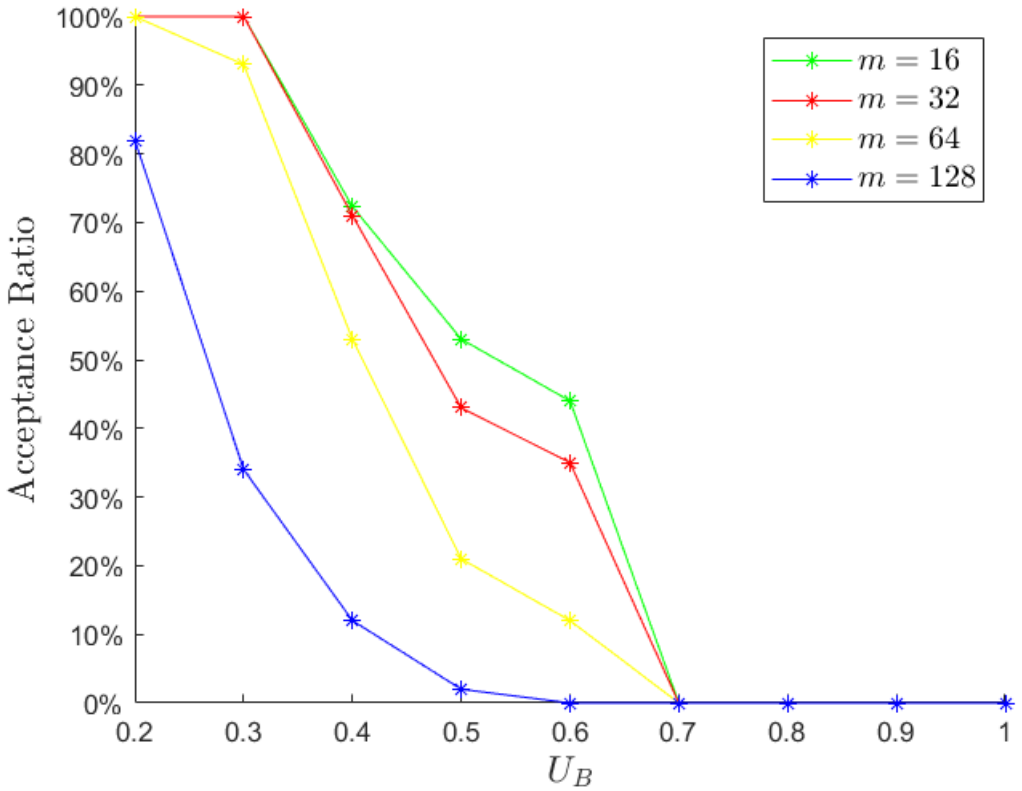

Figure 8.1: Comparison of acceptance ratios for different number of processors

2) Impact of P_{max}

P_{max} is the upper bound of the ratio of the deadline and the overload critical path length. It has a huge impact on the schedulability of the parallel task because it affects the structure (parallelism) of tasks. The P_{max} becomes larger means that the task has larger parallelism. When it closer to 1, the task is more close to a sequential task.

When P_{max} increases, tasks have more slack to complete the work, which can be executed parallelly, so they do not require a large computing resource. That makes

our algorithm much easier to find a feasible solution. Figure 8.2 shows the change of acceptance ratio by varying P_{max}.

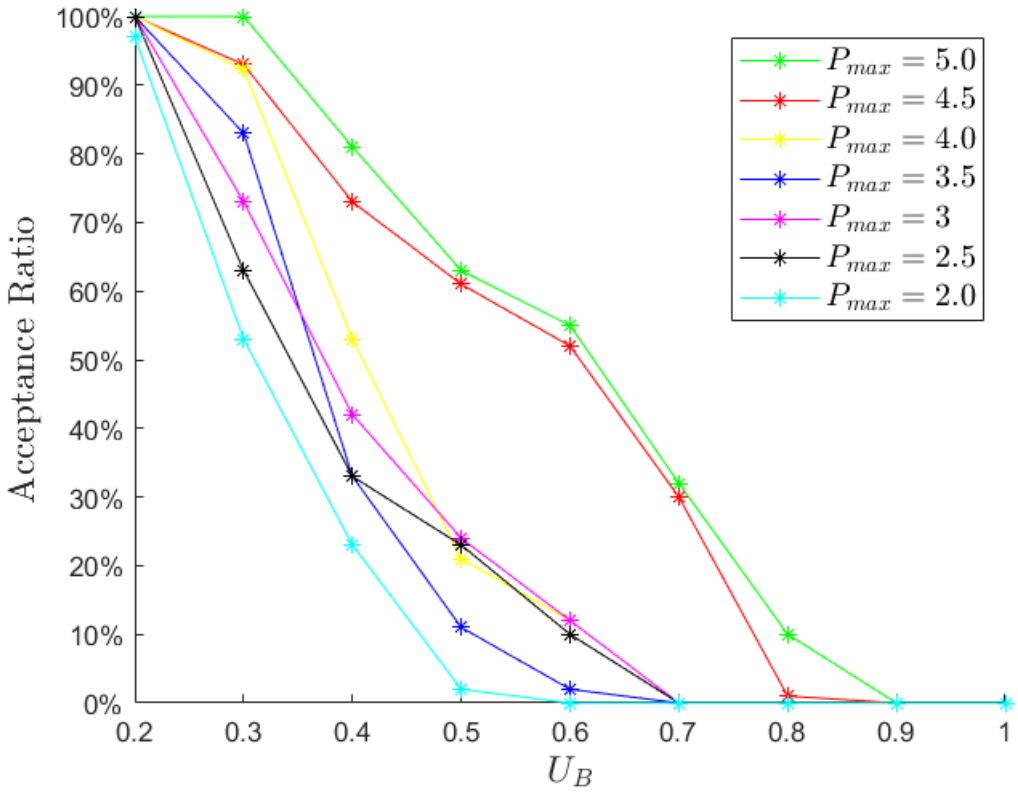

Figure 8.2: Comparison of acceptance ratios for different P_{max}

3) Impact of p^{hu}

p^{hu} controls the proportion of heavy tasks in a task set. Figure 8.3 shows the change of acceptance ratio by varying p^{hu}. When it becomes larger, the task set will contain more heavy tasks.

The heavy tasks will be allocated a set of processors, and they exclusively own the assigned processors. This way has a drawback in that it may waste computing resources. To make it clear, we consider the situation under the homogeneous platform. Under the homogeneous platform, assume a heavy task needs $k + \epsilon$(k is an integer and $0 < \epsilon < 1$) computing resource to become schedulable. According to the federated scheduling, we need to allocate $\lceil x + \epsilon \rceil$ processors to it. So $1 - \epsilon$ computing resources are wasted. The proportion of wasted resources is positively correlated with P^{hu}. So the acceptance ratio goes down as the p^{hu} goes up.

4) Impact of u_{max}

u_{max} is the upper bound of overload utilization of each heavy task. Figure 8.4 shows the change of acceptance ratio by varying u_{max}.

Each time, we add one task into the task set until the specific U_B is met. When u_{max} becomes larger, that makes the task set contains more heavy tasks. So why does the acceptance ratio decrease as u_{max} increases for the same reason as p^{hu}, which is shown in section 8.3.1.

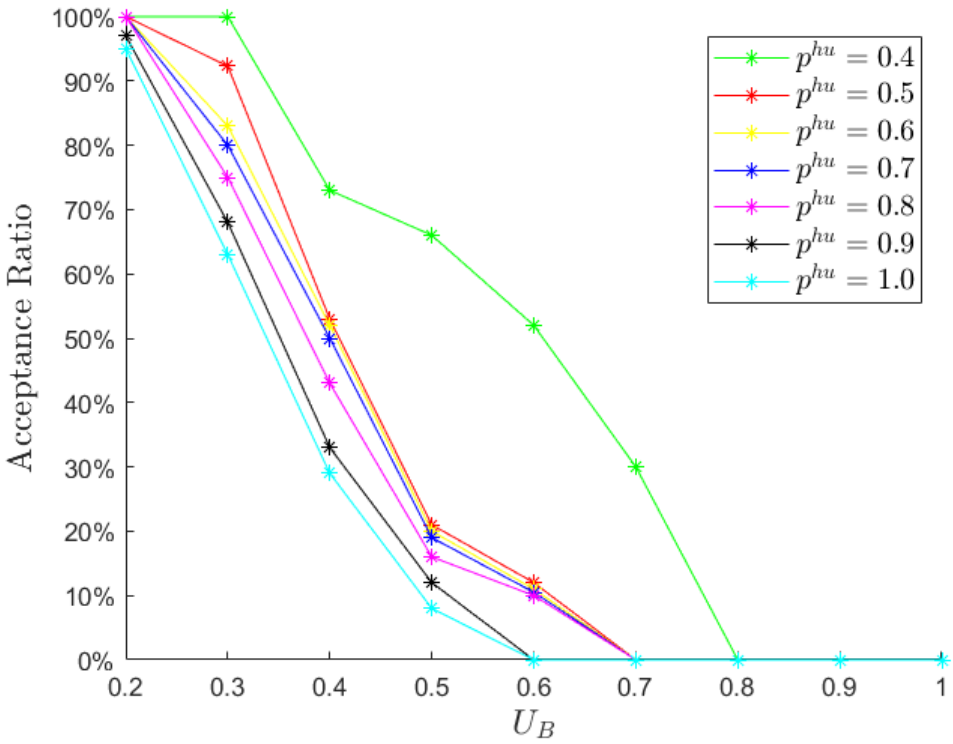

Figure 8.3: Comparison of acceptance ratios for different p^{hu}

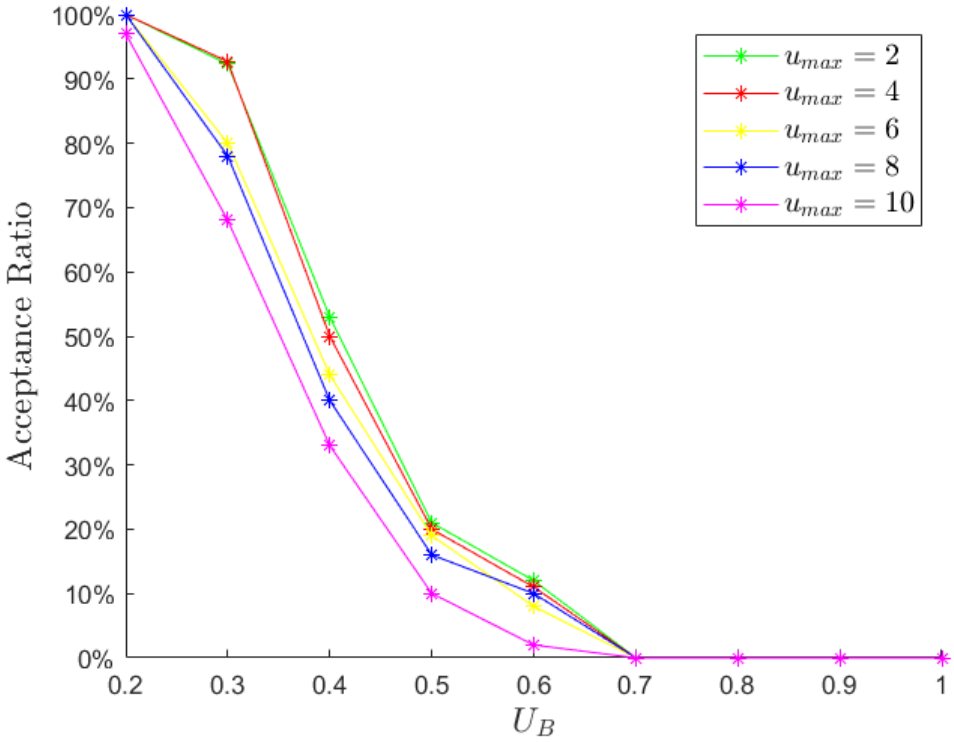

Figure 8.4: Comparison of acceptance ratios for different u_{max}

5) Impact of R_{max} and p^{hc}

R_{max} is the upper bound of the ratio of overload and nominal utilization, and P^{hc} is the proportion of HI-criticality tasks in a task set. They do not have a significant impact on the acceptance ratio because they do not have a significant impact on the schedulability test.

We use weighted acceptance ratio to show their effects. Let $R(U_B)$ denote the acceptance ratio under a specific U_B. The weighted acceptance ratio is computed as $\sum R(U_B)U_B / \sum U_B$. Figure 8.5 shows the change of weighted acceptance ratio by varying R_{max} and p^{hc}.

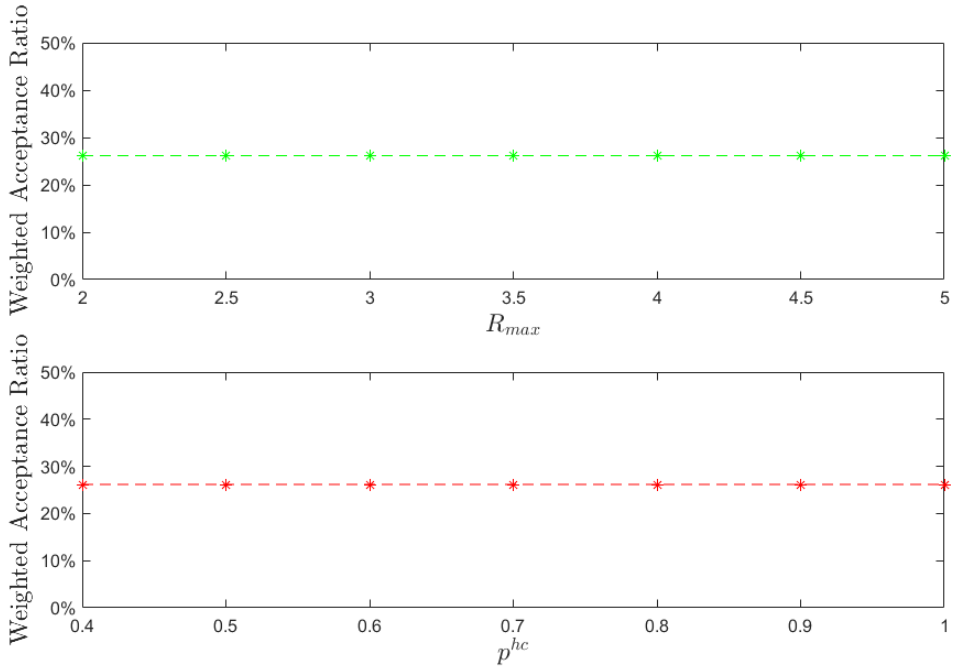

Figure 8.5: The weignted acceptance ratio under different R_{max} and p^{hc}

8.3.2 Performance under homogeneous platform

The homogeneous platform can be seen as a kind of special uniform platform. All the processors have the unit speed. We can set $S_m = m$ to generate a homogeneous platform.

Figure 8.6 compares the acceptance ratio of our algorithm with `MCFS-Bound`, which is proposed by Li in [19], with different number of processors.

The acceptance ratio of our algorithm is better than the aceptance ratio of `MCFS-Bound` for $m = 16, 32, 64, 128$. For example, the acceptance ratio at $U_B = 0.3$ for $m = 128$ is around 35% and the acceptance ratio for `MCFS-Bound` is around 10%. And when U_B increases, the acceptance ratio of the `MCFS-Bound` decreases more quickly.

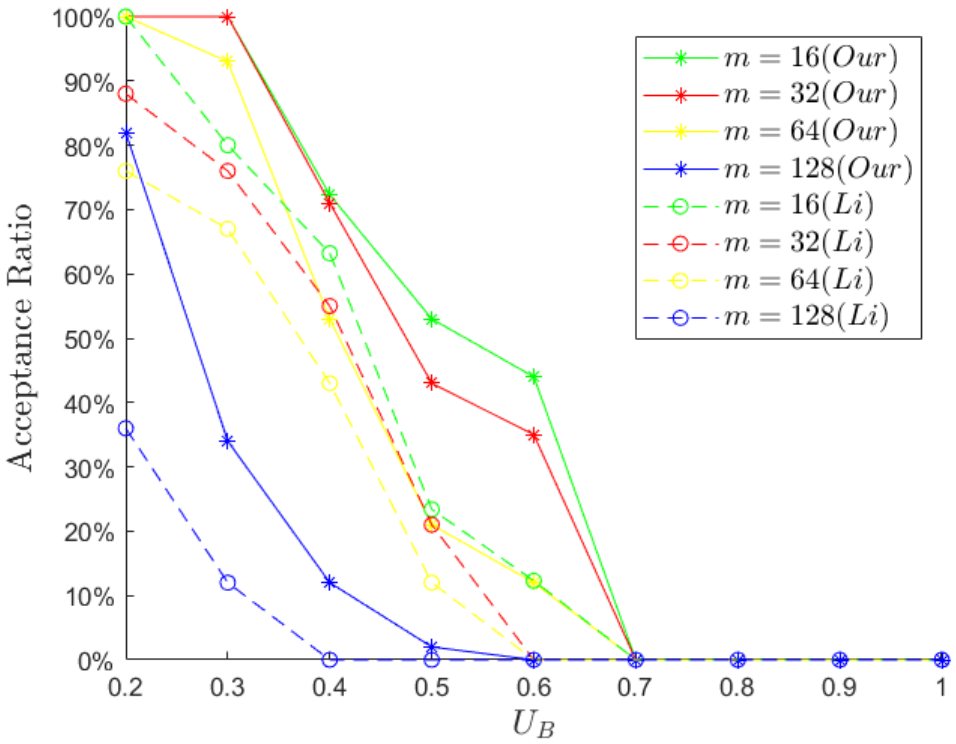

Figure 8.6: Performance under homogeneous platform

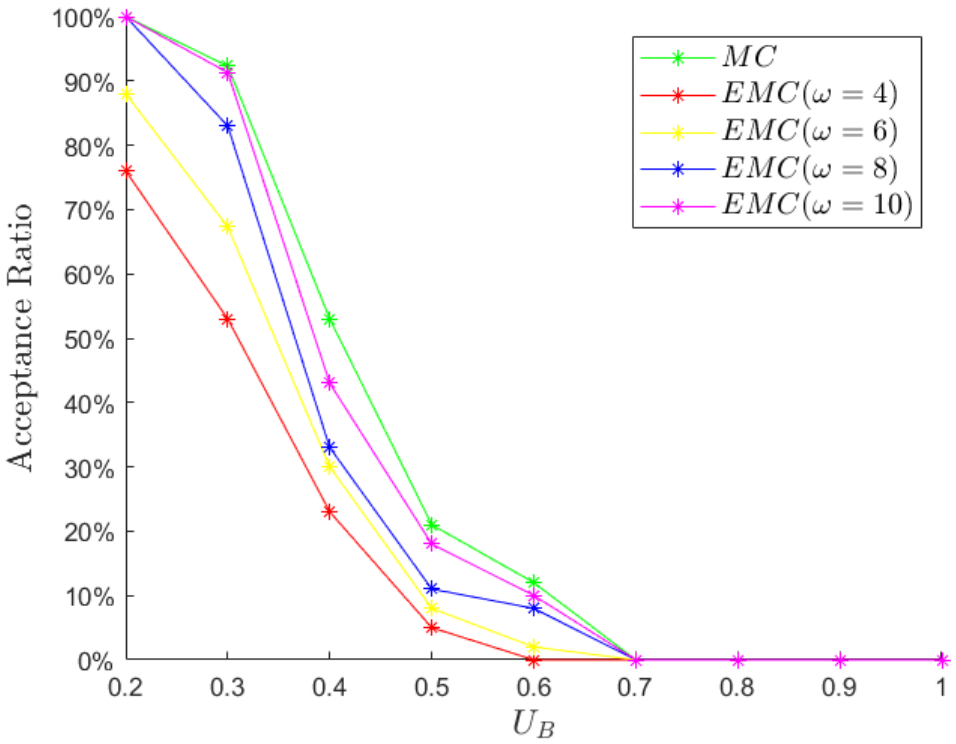

Figure 8.7: Performance under elastic mixed-criticality systems

8.3.3 Performance under elastic mixed-criticality systems

In elastic mixed-criticality systems, each LO-criticality task τ_i has a elastic degree $\omega_i = T_i^{max}/T_i$. We use a parameter average elastic degree $\omega \in \{4, 6, .., 10\}$ to generate the ω_i for each LO-criticality task. After generating a task set for the traditional mixed-criticality systems. The number of LO-criticality tasks is determined. Assume it is $|LO|$. Since $\omega = \sum_{z_i=LO} \omega_i/|LO|$, we can use $|LO|\omega$ and $|LO|$ as the input of the DRS and each value is limited in $[\omega - 2, 10]$ to generate the ω_i for each the LO-criticality task.

Figure 8.7 shows the performance of our algorithm under elastic mixed-criticality systems. Since we generate the elastic degree for each LO-criticality task after we generate the task set for traditional mixed-criticality systems, here, the value of U_B is still computed under the traditional mixed-criticality system. Actually, the true U_B for the elastic mixed-criticality systems should be larger than the U_B for traditional mixed-criticality systems because the LO-criticality tasks are not discarded at HI-criticality mode under elastic mixed-criticality systems.

The traditional mixed-criticality systems can be seen as a special case of elastic mixed-criticality systems, the elastic degree for the LO-criticality task is infinite. When the average elastic degree increases, the true U_B for elastic mixed-criticality systems is more close to the U_B for traditional mixed-criticality systems. So when the average elastic degree increases, the acceptance ratio of the elastic mixed-criticality systems is more close to it under the traditional mixed-criticality systems.

9
Conclusion and future work

This thesis presents the scheduling algorithm for dual-criticality parallel tasks on uniform multiprocessor platform. The algorithm is based on federated scheduling. An upper bound of response time for mixed-criticality parallel tasks on a uniform multiprocessor platform is given. A heuristics based on bin-packing and simulated annealing is developed to allocate tasks on processors. To address the serve abrupt problem in the traditional mixed-criticality systems, a discussion on how to extend the task allocation algorithm to elastic mixed-criticality systems is presented.

There are several directions for future work. We assume that the system will transition to HI-criticality mode at a HI-criticality task miss its virtual deadline. Suppose the system can have some indications as to whether a job will overrun during runtime. A good direction to consider is how we can use this information to improve the schedulability test to increase the system utilization. Under federated scheduling, each heavy task exclusively executes on its dedicated cluster. This makes getting the schedulability test become easy. But this property also makes federated scheduling suffer significant resource waste. An interesting future research direction could be to develop the schedulability test in case that sharing processors among the clusters is allowed. Note that the task can have a shorter response time on a cluster with a smaller uniformity λ. The uniformity intuitively measures how similar a uniform multiprocessor system and corresponding homogeneous multiprocessor system. Consider two clusters $P_1 = \{0.5, 0.5, 0.5\}$ and $P_2 = \{0.6, 0.5, 0.4\}$. They have the same total capacity as 1.5. But $\lambda_1 = 2$, $\lambda_2 = 1.5$. A DAG with $C = 10$ and $L = 5$. Its response time is $40/3$ on cluster P_1 and $35/3$ on cluster P_2. That means that maybe we let the processor speeds differ from each other by greater amounts in the cluster to make more task sets can find a feasible task allocation scheme. This feature is worth considering when designing task allocation algorithm. The schedulability test for the system with more than two criticality levels and on the unrelated multiprocessor platforms is to be done.

Bibliography

[1] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–37, 2017.

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

[3] A. Gupta, S. Im, R. Krishnaswamy, B. Moseley, and K. Pruhs, "Scheduling heterogeneous processors isn't as easy as you think," in *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, (USA), p. 1242–1253, Society for Industrial and Applied Mathematics, 2012.

[4] M. R. Garey and D. S. Johnson, *Computers and intractability*, vol. 174. freeman San Francisco, 1979.

[5] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys*, vol. 43, no. 4, pp. 1–44, 2011.

[6] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[7] S. Khuri and T. Bäck, "An evolutionary heuristic for the minimum vertex cover problem," in *Genetic Algorithms within the Framework of Evolutionary Computation–Proc. of the KI-94 Workshop*, pp. 86–90, Citeseer, 1994.

[8] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

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

[10] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *2014 26th Euromicro Conference on Real-Time Systems*, pp. 85–96, 2014.

[11] X. Jiang, N. Guan, X. Long, and W. Yi, "Semi-federated scheduling of parallel real-time tasks on multiprocessors," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, pp. 80–91, 2017.

[12] S. Funk, J. Goossens, and S. Baruah, "On-line scheduling on uniform multiprocessors," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, RTSS '01, (USA), p. 183, IEEE Computer Society, 2001.

[13] S. Baruah, "Scheduling periodic tasks on uniform multiprocessors," in *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000*, pp. 7–13, 2000.

[14] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 147–152, 2013.

[15] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Transactions on Computers*, vol. 51, no. 3, pp. 289–302, 2002.

[16] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi, "Edf-vd scheduling of mixed-criticality systems with degraded quality guarantees," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, pp. 35–46, 2016.

[17] D. Griffin, I. Bate, and R. I. Davis, "Generating utilization vectors for the systematic evaluation of schedulability tests," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, pp. 76–88, IEEE, 2020.

[18] R. M. Pathan, "Improving the schedulability and quality of service for federated scheduling of parallel mixed-criticality tasks on multiprocessors," in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[19] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu, "Mixed-criticality federated scheduling for parallel real-time tasks," *Real-time systems*, vol. 53, no. 5, pp. 760–811, 2017.

A

Source code

The source code for the task allocation algorithm and task set generation can be found at `https://github.com/ChengziHuang/taskAllocation`.