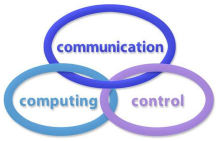
INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
Online ISSN 1841-9844, ISSN-L 1841-9836, Volume: 16, Issue: 3, Month: June, Year: 2021
Article Number: 4221, https://doi.org/10.15837/ijccc.2021.3.4221



CCC Publications

Job-shop Scheduling Over a Heterogeneous Platform

J. A. Hermosillo-Gomez, H. Benitez-Perez

Jose A. Hermosillo-Gomez*
Institute of Applied Mathematics Research and Systems (IIMAS)
National Autonomous University of Mexico, Mexico
3000 Circuito Escolar, Cd. Universitaria, Coyoacán 04510, Mexico City
*Corresponding author: jose.hermosillo@iimas.unam.mx

Hector Benitez-Prez
Institute of Applied Mathematics Research and Systems (IIMAS)
National Autonomous University of Mexico, Mexico
3000 Circuito Escolar, Cd. Universitaria, Coyoacán 04510, Mexico City
hector.benitez@iimas.unam.mx

Abstract

Real-time scheduling involves determining the allocation of platform resources in such a way tasks can meet their temporal restrictions.

This work focuses on job-shop tasks model in which a task have a finite number of nonpreemptive different instances (jobs) that share a unique hard deadline and their time requirements are known until task arrival.

Non-preemptive scheduling is considered because this characteristic is widely used in industry. Besides job-shop scheduling has direct impacts on the production efficiency and costs of manufacturing systems. So that the development of analysis for tasks with these characteristics is necessary.

The aim of this work is to propose an online scheduling test able to guarantee the execution of a new arriving task, which is generated by human interaction with an embedded system, otherwise to discart it. An extension of the schedulability test proposed by Baruah in 2006 for non-preemptive periodic tasks over an identical platform is presented in this paper. Such extension is applied to non-preemptive tasks that have hard deadlines over a heterogeneous platform. To do that, some virtual changes over both the task set and the platform are effectuated.

Keywords: Embedded systems, real-time, uniform heterogeneous platform, non-preemptive jobs.

1 Introduction

Traditionally, the job-shop problem is an optimisation problem in which many manufacturing jobs are assigned to machines at particular times while trying to minimise the makespan. However, here it is considered a platform whose machines have the same processing capabilities but different speeds, this characteristic gives certain flexibility to the traditional job-shop problem (*JSP*) [1] in such a way that there is a machine sequence-dependency setup avoidance.

The addressed task model consists of nonpreemptive *job-shops*. Let τ_i be a *job-shop*, this kind of task is composed by n_i jobs such that $n_i \in \mathbb{N}$ and $n_i < \infty$; for the propose of this work, a *job* is defined as an ordered sequence of operatios that must be executed without interruption (i.e. atomic jobs). The j^{th} job of the i^{th} task is represented as $\tau_{i,j}$ where $i = 1, \ldots$ and $j = 1, \ldots, n_i$.

A job-shop, τ_i, is an ordered sequence of jobs: let $\tau_i = \{\tau_{i,j}: i = 1, \ldots,\ j = 1, \ldots, n_i\}$ be a job-shop where $n_i \in \mathbb{N}$ denotes its number of jobs. These jobs follow *precedence constraints*, that is, the job $\tau_{i,j}$ cannot start its execution if the previous job $\tau_{i,j-1}$ has not finished yet. Job-shops are also characterised by temporal restrictions such as a unique release time instant (r_i), a critical unique deadline (d_i), and n_i execution time requirements (one per job), denoted as $c_{i,j}$ (where $j = 1, \ldots, n_i$).

Definition 1 (Precedence constraint)**.** *For any two consecutive jobs $\tau_{i,j-1}$, $\tau_{i,j} \in \tau_i$, the job $\tau_{i,j-1}$ precedes $\tau_{i,j}$ if its finishing time $f_{i,j-1}$ meets that $f_{i,j-1} \leq b_{i,j}$, where $b_{i,j}$ denotes the $\tau_{i,j}$'s beginning instant of execution, $i = 1, \ldots$ and $j = 2, \ldots, n_i$.*

Scheduling research needs to focus on this type of tasks because of its importance in the inherent processes of industry (manufacturing, inventory costs, transportation schedules, etc.). Under the industry 4.0, the scheduling should deal with smart and distributed manufacturing systems supporting by novel and emerging manufacturing technologies such as mass customisation, cyber-physical systems, digital twins, and SMAC (Social, Mobile, Analytics, and Cloud), big data, etc. [12].

Embedded systems are an ideal platform to implement projects in terms of the requirements of *IoT* and Industry 4.0 regarding different resource capacities (communication, computing, memory, etc.), energy efficiency, etc. Embedded systems must operate in dynamic environments where human activities occur at any moment, then some tasks, such as emergency tasks, external event tasks, human interaction tasks, etc., arrive aperiodically [11]. Such aperiodic interactions might be the cause of generating new entering tasks (in this case job-shops) that should be completed within their respective deadline by the embedded systems.

Embedded platforms for dynamic environment domains are expected to handle several tasks at the same time, such as the task model previously described.

Non-preemptive scheduling is widely used in industry, and it may be preferable to preemptive scheduling for certain reasons [8]: non-preemptive scheduling algorithms are easier to implement and have lower run-time overhead than preemptive scheduling algorithms; the overhead of preemptive scheduling algorithms is more difficult to characterise and predict than of non-preemptive scheduling due to inter-task interference caused by caching and pipelining. These benefits of non-preemptive scheduling are more important on multiprocessor since the task migration overhead is higher and more difficult to predict [7].

A result that gives the possibility of thinking that for a considerable part of real-life applications on multiprocessor platforms, non-preemptive scheduling could be a better choice with respect to real time performance is specified in [7], since in it is experimentally showed that for task sets, in which the range of execution time is not very wide, the performance of EDF_{np} (short for non-preemptive EDF) is very close to *EDF*, and DM_{np} (short for non-preemptive DM) performs better than *DM*.

This work addresses the problem of scheduling sets of arriving non-preemptive job-shops that have to be scheduled over a heterogeneous platform by their temporal restrictions. By means of extending [3] it is possible to determine if a set of arriving job-shops is schedulable online.

Contrary to [3] that holds for static[1] conditions; in this work, the attention is focused on those tasks that arrive at unknown time instants. The managed tasks, job-shops [2] also have their respective unique [6] but critical deadlines.

The platform considered for scheduling these tasks is a *uniform heterogeneous* [4] one. Let $\pi = (s_1,\ s_2,\ \ldots, s_m)$ denote such multiprocessor platform of size m with the i^{th} processor having a speed rate s_i, where $s_i \geq s_{i+1}$ for $i = 1, \ldots, m-1$, as bigger the value of s_i the slower the i^{th} processor is.

The problem addressed in this work is about making the decision of assigning new released job-shops into a heterogeneous platform at certain instant t, then a schedulability test is defined in function of the active tasks as from time t.

[1] Not only the number of tasks still the same but also their time computing requirements, during execution.

[2] There are also other names like coordinated tasks[10] in the literature to name this tasks which, at the same time, are a particular case of DAG tasks

The rest of this paper presents a review of the analysis in which this work is based (section 2); defines the transformation operations that let apply the proposed modified *BAR06* analysis to schedule new entering job-shops (section 3); describes the possible transitions that the jobs may have from its arriving onto the embedded system to its execution or discard (section 4); describes the server construction (section 5); introduces an important added component to analyze entering tasks (section 6); finally enunciates a theorem that extends the usability of *BAR06* over job-shops in a uniform heterogeneous platform (section 7).

2 Related work

There is a few work dedicated to non-preemptive task schedulability analysis, one of the most known papers about this topic is [3], and [7] which is an improvement of the first.

Recent work focuses their efforts on developing analysis to determine schedulability of a set of fixed set of periodic tasks over a homogeneous multiprocessor platform: [9], [2]. [9] seeks to bound the worst-case response time of a set of jobs which access shared resources, and a non-work-conserving nonclairvoyant and non-preemptive scheduling framework is proposed in [2], which is able to find feasible schedules for work-conserving-infeasible task sets. Although, they do not consider the constant interaction with users that produces new entry tasks in a system.

The schedulability analysis of sufficiency proposed by Baruah [3], henceforth called *BAR06*, consists of validating the restriction (1) for a set of n known periodic tasks, τ, on an identical m-multiprocessor platform.

$$V_{sum}(\tau) \leq m - (m-1) \cdot V_{max}(\tau) \tag{1}$$

where

$$V_{sum}(\tau) := \sum_{\tau_i \in \tau} V(\tau_i, \tau) \tag{2}$$

$$V_{max}(\tau) := \max_{\tau_i \in \tau} \{V(\tau_i, \tau)\} \tag{3}$$

and $V(\tau_i, \tau)$ is the utilization of the periodic non-preemptive task τ_i:

$$V(\tau_i, \tau) := \frac{c_i}{T_i - e_{max}(\tau)} \tag{4}$$

$e_{max}(\tau)$ is the maximum of the execution requirements in τ, as appreciated at (4), it is the worst case interference time scenario caused by non-preemption:

$$e_{max}(\tau) := \max_{i=1,\ldots,n} \{c_i\} \tag{5}$$

where c_i and T_i are the execution requirement and the interrelease times of the periodic task, $\tau_i \in \tau$, respectively. An obvious observation is that a task $\tau_i \in \tau$ with low utilization can not pass (1) if $c_i > T_{min}$, where T_{min} denotes the minimal period T_i of the tasks in τ.

3 Transformations

Global EDF_{np} try to assign as many tasks with the promptest deadline as it can to available processors, then schedules the instances with the next earliest deadlines as it can and on and on, until either all processors are busy or there are no more waiting task instances.

With the aim of applying the schedulability analysis proposed by Baruah in [3] over job-shops scheduled by global EDF_{np} in a heterogeneous platform, the next series of *virtual* transformations is realized:

1. Transformation of a heterogeneous platform into a homogeneous one.

2. Transformation of the incoming aperiodic non-preemptive job-shops into periodic tasks.

Lemma 1. *A uniform heterogeneous platform has associated an identical (homogeneous) platform.*

Proof. The proof is by construction. Let $\pi = (s_1,\ s_2,\ \ldots, s_m)$ denote the speed rates of the uniform multiprocessor platform of size m with the k^{th} processor having a rate of s_k, where $0 < s_k \leq 1$ (note that rate $s_{max} = 1$ is the slowest one). To convert the current heterogeneous platform into a homogeneous one is necessary to meet that all processors speeds are the same. To do that it is necessary to find a factor for every processor in the platform, let α_k be such value, $k = 1, \ldots, m$, such that $\alpha_k \cdot s_k = s_{max}$ where s_{max} is the rate of the slowest processor

$$s_{max} := \underset{k=1,\ldots,m}{max}\{s_k\}$$

then

$$\alpha_k = \frac{s_{max}}{s_k}$$

□

This new homogeneous platform is a virtual platform over which the schedulability test will be applied for the new entering tasks. Note that the factor α_k indicates how much the k^{th} virtual processor must slow down to be part of the slowest homogeneous platform. Also note that the bigger α_k is, the slower the k^{th}-processor will turn.

By lemma 2, the execution requirements of the pseudo-periodic tasks are built under the assumption of having an underlying homogeneous platform mentioned in lemma 1. It is assumed that for every $\tau_{i,j} \in \tau_i$ there exists an execution requirement $c_{i,j}$ $(i = 1, \ldots,\ n)$ that is given *in terms of the slowest processor in* π. It means that a job, $\tau_{i,j}$, executed on the k^{th}-processor of the heterogeneous platform last less than on a homogeneous one: $c_{i,j} \geq c_{i,j} \cdot s_k$ for any $k = 1, \ldots, m$.

Lemma 2 (Pseudo-periodic task)**.** *A job-shop, τ_i, has its equivalent associated periodic task, which is called the associated pseudo-periodic task, let denote it as τ_i'.*

Proof. The proof is by construction. An associated pseudo-periodic task, τ_i', can be represented as the tuple $(r_i, C_i, T_i(t))$, where r_i is the same release time as in τ_i; C_i, short for $C(\tau_i)$, is the *virtual* execution requirement, which is computed as

$$C(\tau_i) := \max_{\tau_{i,j} \in \tau_i}\{c_{i,j}\} \tag{6}$$

thus, for each $c_{i,j}$ of every non-executed job $\tau_{i,j} \in \tau_i$, it is necessary to add $C_i - c_{i,j}$ units of *unoccupied* time in order to have n_i instances with the same execution requirements (just like periodic task instances); $T_i(t)$, short for $T(\tau_i, t)$, is the imposed *virtual* period of the task τ_i that is computed as

$$T(\tau_i, t) := \left\lfloor \frac{d_i - t}{n_i} \right\rfloor \tag{7}$$

where d_i is the absolute deadline of τ_i and t is the time instant in which T_i is calculated. It must be met that $T_i(t) \cdot n_i \leq d_i$ and $C_i \leq T_i(t)$. □

In this studied setting, the jobs of $\tau_i \in \tau$, for all $i = 1, \ldots$, are finite; their execution requirements are not necessary equal; and they are consumed during run time. Also, the arrival instants of new job-shops are considered during run time (online). Hence the necessity to modify *BAR06* analysis: Considering the evolution in time of the execution of the "accepted" job-shops.

Once $\tau_{i,j} \in \tau_i$ was consumed, is *discarded*, therefore $v(t, \tau_i, \tau)$ must be calculated by means of substituting definition (6) and (7) (see lemma 2) into (4) every time the server task is active at instant t.

$$v(t, \tau_i, \tau) := \frac{C_i}{T_i(t) - e_{max}(\tau)} \tag{8}$$

where $e_{max}(\tau)$ is the maximum execution requirement of the periodic task set τ:

$$e_{max}(\tau) := \max_{\tau_i \in \tau}\{C_i\} \tag{9}$$

once considered all this run time dynamism to the new model, innequation (1) has been modified as showed next:

$$v_{sum}(t,\tau) \leq m - (m-1) \cdot v_{max}(t,\tau) \tag{10}$$

where

$$v_{max}(t,\tau) := \begin{cases} \max_{\tau_i \in \tau}\{v(t,\tau_i,\tau)\} & \text{if } \tau \neq \phi \\ 0 & \text{if } \tau = \phi \end{cases} \tag{11}$$

and

$$v_{sum}(t,\tau) := \begin{cases} \sum_{\tau_i \in \tau} v(t,\tau_i,\tau) & \text{if } \tau \neq \phi \\ 0 & \text{if } \tau = \phi \end{cases} \tag{12}$$

note the dependence on time for this new acceptance analysis.

So far, it is known how to tailor the temporal restrictions of a fixed set of job-shops into a set of periodic tasks but not how to go through with new incoming job-shops to the system. Therefore, a systematic way of converting and verifying the released tasks is described below.

4 Task transitions

Global EDF_{np} is the underlying algorithm for this approach and it also utilizes a task *server* (τ^s), they are adapted to a multiprocessor platform to allocate resources for released incoming tasks (here task means a job-shop).

There are four possible states tasks can go through, namely *released, wait, active*, and *dead*; each state disposes of its own data structure (i.e. a heap) that helps to make reference to the treated tasks. As showed in Figure 1, for instance, the state *released* has associated the data structure $\tau_{released}$, that receives all new execution requests of arriving tasks at any time to the system.

When an interruption occurs at instant t and there is enough budget for the server task (let C^s denote the budget capacity of the server), tasks referenced by $\tau_{released}$ begin to be transformed (see lemma 2) and then they change their status to *wait*, it means that they were allocated to τ_{wait} by τ^s. Formally $\tau_{released}$ is defined as follows

$$\tau_{released} := \{\tau_i : 0 \leq r_i \leq t\}$$

When global EDF_{np} selects τ^s as the highest priority task and all tasks from $\tau_{released}$ have been transformed, τ^s *validates* which tasks from τ_{wait} are still schedulable.

Definition 2 (Schedulable task)**.** *At instant t, given the total utilization of the currently performing pseudo-periodic tasks, $v_{sum}(t,\tau_{active})$, it is said that a task $\tau_i \in \tau_{wait}$ is still schedulable at instant t if it can still be executed by its deadline, d_i:*

$$t + n_i \cdot C_i + v_{sum}(t,\tau_{active}) \cdot \frac{(d_i - t)}{m} \leq d_i \tag{13}$$

where n_i is the number of jobs in τ_i that have not been scheduled yet and m is the number of processors.

If a schedulable task in $\tau_k \in \tau_{wait}$ passes *BAR06* analysis, then its status changes to *active* and is moved towards τ_{active} that is the set of tasks currently assigned to the platform; otherwise, it is either *discarded* (put towards τ_{dead}) or *rejected* (placed back to τ_{wait}). Note that, at instant t, τ_{wait} will keep schedulable tasks:

$$\tau_{wait} := \{\tau_i : satisfies\ (13)\ and\ v_{sum}(t,\tau) > m - (m-1) \cdot v_{max}(t,\tau)\}$$

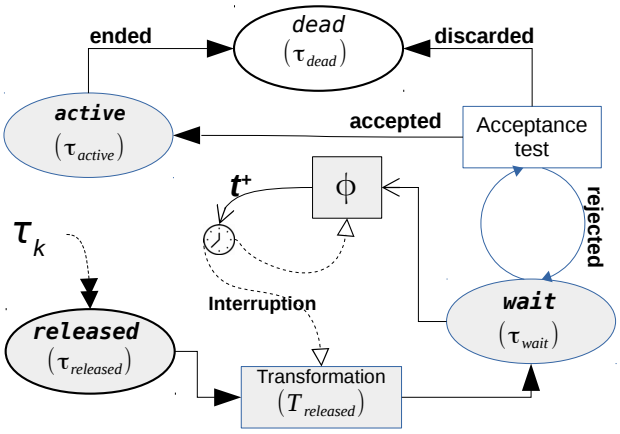


Figure 1: States of tasks during run time

where $\tau = \tau_{active} \cup \{\tau_i\}$.

As it is possible to observe in Figure 1, *rejected* tasks are still schedulable, while *discarded* tasks are thrown away because they cannot meet restriction (13), besides according to [3], it must be satisfied the next restriction before verifying inequality (10): The maximum of the execution requirements in any task set τ, $e_{max}(\tau)$, must be less than or equal to the minimum of the periods of τ: $e_{max}(\tau) \leq T_{min}$.

When a task satisfies (10) is added to τ_{active} which is a global set of pseudo-periodic tasks being scheduled by EDF_{np}.

Algorithm 1 ACCEPTANCE TEST

1: $T_{released} \longleftarrow convert2Pseudoperiodic(\tau_{released})$
2: $\tau_{wait} \longleftarrow \tau_{wait} \cup T_{released}$
3: **for all** $\tau'_k \in \tau_{wait}$ or c^s is not used up **do**
4: $\quad \tau \longleftarrow \tau_{active} \cup \{\tau'_k\}$
5: $\quad$ **if** $t + n_k \cdot C_k + v_{sum}(t, \tau_{active}) \cdot \frac{(d_k - t)}{m} \leq d_k$ **and** $C_k \leq min_{\tau_i \in \tau_{active}}\{T_i\}$ **and** $T_k \geq \max_{\tau_i \in \tau_{active}}\{C_i\}$ **and** $v_{sum}(t, \tau) \leq m - (m-1) \cdot v_{max}(t, \tau)$ **then**
6: $\quad\quad \tau_{active} \longleftarrow \tau$
7: $\quad$ **else if** $t + n_k \cdot C_k + v_{sum}(t, \tau_{active}) \cdot \frac{(d_k - t)}{m} > d_k$ **or** $C_k > min_{\tau_i \in \tau_{active}}\{T_i\}$ **or** $T_k < \max_{\tau_i \in \tau_{active}}\{C_i\}$ **then**
8: $\quad\quad$ discard(τ'_k)
9: $\quad$ **end if**
10: **end for**

As showed in algorithm 1, every time an interruption is triggered at instant t, every $\tau_k \in \tau_{released}$ is stored in an auxiliary data structure, $T_{released}$, and then converted into its associated periodic task, τ'_k, before it becomes part of τ_{wait} (see line 1) and then restrictions (13) and (10) are verified for each $\tau'_k \in \tau_{wait}$ (see lines 3-10).

5 Server design

Generally, the server, denoted by τ^s, is a special periodic task whose intent is to handle new incoming aperiodic requests as soon as possible and without causing previously accepted tasks to miss their deadlines. In this case, two types of requests are considered: first, those tasks that are waiting until instant t in $\tau_{released}$ to be transformed; second, those tasks that are stored in τ_{wait} and need to be validated, in case to be accepted they are moved towards τ_{active}.

τ^s adapts itself to the current platform conditions, it means that it is able to change its budget capacity *at every triggered interruption* (see section 6). Note that task server has not restricted execution so that it can be run on any processor.

The utilization for τ^s, U^s, is bounded according to the capacity left by the tasks in τ_{active}. At instant t, when an interruption has been triggered, U^s is calculated as follows:

$$U^s(t) = m - v_{sum}(t, \tau_{active}) \tag{14}$$

since the addressed problem copes with non-preemptive tasks, U^s is treated as the upper bound for τ^s's utilization value (u^s), this is $0 < u^s \leq U^s$.

It is necessary to regard τ^s as one extra task in the set, then inequality (10) must satisfy:

$$v_{sum}(t, \tau_{active}) + u^s \leq m - (m-1) \cdot v_{max}(t, \tau_{active})$$

thus, for the non-preemptive task system, u^s must meet

$$u^s \leq m - (m-1) \cdot v_{max}(t, \tau_{active}) - v_{sum}(t, \tau_{active}) \tag{15}$$

It is supposed the existence of a user defined value that delimits the minimum value the τ^s's utilization can take, let u^s_{min} be that value, such that $0 < u^s_{min} \leq u^s$. From (15) and u^s_{min}, the value for u^s is defined:

$$u^s := \max\{u^s_{min},\ m - (m-1) \cdot v_{max}(t, \tau_{active}) - v_{sum}(t, \tau_{active})\} \tag{16}$$

with this is pretended to assign the highest possible utilization value for u^s.

The budget capacity for τ^s, C^s, is upper bounded by $e_{max}(\tau_{active})$, (i.e. $0 < C^s \leq e_{max}(\tau_{active})$), this is to guarantee that there will be no need to calculate again the utilization value for all the tasks in $\tau_{acepted}$. To guarantee predictability, it is also necessary to meet that there will be always enough budget to execute, at least, one request in the system, then C^s is bounded in the next way:

$$max\{c_1, c_2\} \leq C^s \leq e_{max}(\tau_{active}) \tag{17}$$

Once that C^s was set, it is necessary to define the absolute deadline calculation for τ^s, d^s. It is calculated in such a way that the interference of the task with the highest job execution requirement, $e_{max}(\tau_{active})$, is considered, and all the allocated budget is used up by such deadline:

$$d^s(t) = t + \left\lceil \frac{C^s}{u^s} + e_{max}(\tau_{active}) \right\rceil \tag{18}$$

Although the considered task set consists of jobs with precedence constraints, the analysis is executed at job level. Under the established conditions and the proposed transformations it is possible to guarantee the execution of a task through their jobs.

The rules to replenish the budget are enunciated next:

1. Initially, at $t = 0$, u^s is set according to (16), note that if there is no task in τ_{active} ($\tau_{active} = \phi$) then $u^s = m$, see definitions (11) and (12), it means that at the beginning τ^s could have the platform at its complete disposal.

2. Every time the deadline of τ^s (d^s) has been reached, C^s is set to the lowest necessary value it is able to take: $C^s = max\{c_1, c_2\}$ (see (17)) and the next deadline d^s is again calculated according to equation (18).

6 Interruptions

Interruptions are important for two reasons. First, in every interruption occurrence, at time t, the server's budget is reset according to the number of tasks in τ_{active}. Second, the next interruption is set at instant t^+ to assist the released tasks ($\tau_{released}$).

It is assumed that the first interruption occurs at $t = 0$ and there is also a minimum utilization value, u^s_{min}, defined by the user for the server task τ^s, then t^+ is calculated as follows

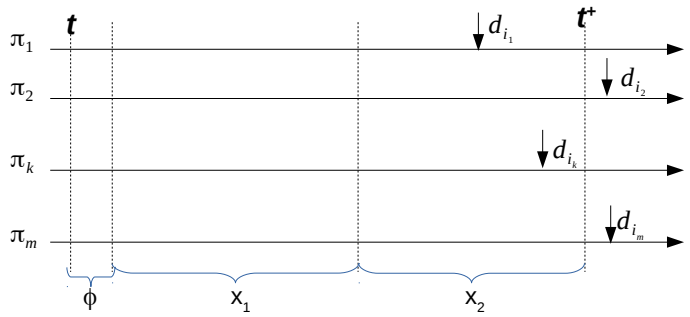


Figure 2: Different operation subintervals in which the interval $[t, t^+)$ may be divided: x_1, x_2, and ϕ

$$t^+ = t + \left\lceil \frac{\phi + x_1 + x_2}{u^s} \right\rceil \tag{19}$$

where ϕ is the time interval that will take the platform to calculate the next interruption, t^+, and the budget for the server; while x_1 and x_2 are the time intervals that will take the transformation and the validation operations (through condition (10)), respectively, see Figure 2.

To know how long it takes to calculate t^+ the next suppositions are made:

S0 It is considered that the server task τ^s always works under the worst case scenario it may have, it is $u^s = u^s_{min}$ (see section 5).

S1 The required time to calculate ϕ is constant. When implementing, it is possible to endow the receiving data structures of counters that permit to know at every time the number of total jobs or tasks in a data structure as required, let $c^{\tau_x}_{jobs}$ and c^{τ_x} be the counters that store the number of jobs and tasks for the data structure τ_x, respectively.

S2 Both transformation and validation are operations that depend on the number of jobs and tasks, respectively.

S3 Neither transformation nor validation operations take zero time. The amount of time used by these operations is described by two constants: Let $c_1 > 0$ and $c_2 > 0$ be such constants for the transformation and validation, respectively.

From **S2** and **S3**, it is easy to deduce that the complexity of calculating t^+ only depends on the number of jobs in $\tau_{released}$ and the number of tasks in τ_{wait}.

Lemma 3 (Task transformation time)**.** *The needed time to transform the tasks in $\tau_{released}$ depends on the number of jobs it has*

$$x_1 = c_1 \cdot \ c^{\tau_{released}}_{jobs}$$

that can be rewritten as

$$x_1 = c_1 \cdot \sum_{\tau_i \in \tau_{released}} n_i$$

where n_i is the number of jobs in the task $\tau_i \in \tau_{released}$.

Proof. Proof follows from the suppositions **S2** and **S3**. □

After performing the transformation of the tasks in $T_{released}$, they are added to τ_{wait} where they wait until acceptance or removal. Note also that it is possible to know in a delimited time interval if a task cannot be performed by its deadline, namely $[t, t + \phi + x_1)$, this gives the possibility to the user of changing the task to other system.

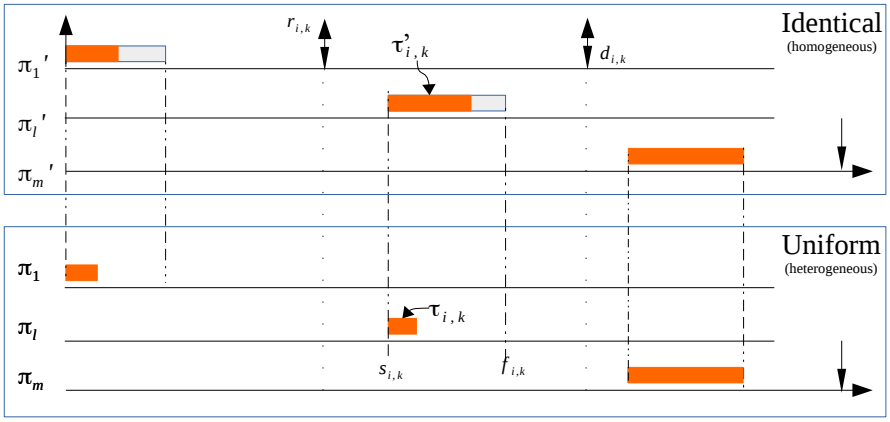


Figure 3: Correspondence between a job-shop (down) and its pseudo-periodic associated task (up)

Lemma 4 (Task validation time)**.** *Once the tasks have been transformed, the needed time for validating the tasks in τ_{wait} by BAR06 is*

$$x_2 = c_2(2 \cdot c^{\tau active} + c^{\tau wait})$$

where $c^{\tau active}$ and $c^{\tau wait}$ represent the number of stored tasks in the data structures τ_{active} and τ_{wait}, respectively.

Proof. Let τ_{wait} be the data structure in which the tasks wait until either execution or removal (to discard a task).

For *BAR06*, it is needed to calculate two values from the tasks that are currently assigned to the platform, τ_{active}; first, the maximum execution requirement, $v_{max}(t, \tau_{active})$, this operation takes proportional time to the number of tasks in τ_{active}, $c^{\tau active}$; second, $v_{sum}(t, \tau_{active})$ that also takes proportional time to $c^{\tau active}$.

The tasks in τ_{wait} are validated in a sequential way, thus every other task trying to be part of $\tau_{acvtive}$ do not increase the time complexity. It is still having linear time, according to the number of tasks that need to be validated, then the time, that take the validation of all tasks in τ_{wait}, is added. □

As mentioned above, in supposition **S1**, through the use of counters in every data structure, it is possible to obtain the information of the number of tasks and jobs from $\tau_{released}$, τ_{wait}, and τ_{active} in constant time, thus the calculations of x_1 and x_2 can be considered as negligible and so do for ϕ. Therefore, equation (19) can be reduced to

$$t^+ = t + \left\lceil \frac{x_1 + x_2}{u^s} \right\rceil . \tag{20}$$

7 Theorem

After all of those transformations over both the platform and the task set; the validation of tasks, and interruptions, the next result is enunciated.

Theorem 1 (Heterogeneous schedulability guarantee)**.** *If a set of pseudo-periodic tasks is scheduled on the associated homogeneous platform, then it is also scheduled on the underlying heterogeneous platform.*

Proof. The job-shops are treated as periodic tasks, due to their previous transformation (see lemma 2), and their respective jobs will be chosen by global EDF_{np} at run time. When EDF_{np} selects a job

$\tau'_{i,j} \in \tau'_i$ from τ_{active} to execute it, it actually selects its corresponding job $\tau_{i,j}$ in the job-shop τ_i which is assigned to the heterogeneous platform.

The start $s_{i,k}$ and release $r_{i,j}$ times of every job will be the same in both platforms. Although the jobs assigned to the heterogeneous platform finish its processing in less time they follow the execution of the homogeneous platform (see Figure 3).

Since $\tau'_i \in \tau_{active}$ its schedulability is guaranteed by *BAR06*, as previously mentioned, the execution requirements have been given in terms of the slowest processor speed, then at run-time, when $\tau'_{i,j}$ is assigned to the homogeneous platform, with an execution requirement of C_i, at instant $s_{i,j}$ then the job associated ,$\tau_{i,j}$, executed on the π_l processor does not exceed the limit imposed by the homogeneous platform, this is

$$s_{i,j} + C_i \cdot \frac{s_l}{s_{max}} \leq s_{i,j} + C_i$$

as $0 < s_l \leq s_{max} \leq 1$ then $\frac{s_l}{s_{max}} \leq 1$, where s_l is the speed rate of π_l.

As the speeds in the heterogeneous platform are faster than in the identical (homogeneous) platform (see lemma 1), the execution requirement of every job $\tau_{i,j}$, $c_{i,j}$, will meet $c_{i,j} \leq C_i$, where C_i is the execution requirement of the associated pseudo-periodic task τ'_i. □

8 Conclusions

This extension to *BAR06* lets determine online if a new entering nonpreemptive job-shop is able to meet its deadline over a heterogeneous platform. Since this schedulability analysis extension has a linear time complexity, according to the number of tasks, it is convenient to implement the presented result on embedded systems with limited computational resources where human activities occur at any moment generating new aperiodic entering tasks.

It is important to emphasize that the time of estimating the stay of a task in embedded systems represent a challenge due to the availability and capabilities of resources. It should be also noted that important aspects like memory management and communication delays in task migration are not yet considered. To fulfill current system requirements most of these features should be included to provide better schedulability analysis.

Nevertheless this is a deterministic approximation that focuses on systems with limited resources on platforms composed of diverse components and it could be used as guidance for developing another algorithms, metrics, and strategies.

The presented strategy has the potential to be implemented in Decision Support Systems (DSS). One concrete example could be DISPATCHER® [5] which is a family of DSS that are meant to support the logistics and production control decisions to be made in the context of the continuous process industries and related fields. Another example could be an air traffic management system where the scheduling conditions change frequently and the necessities of resources usage (for instance, the aircraft takeoff and landing areas) must be validated constantly.

Conflict of interest

The authors declare no conflict of interest.

References

[1] D. Applegate and W. Cook, *A Computational Study of the Job-Shop Scheduling Problem,* INFORMS Journal on Computing, vol. 3, pp. 149–156, May 1991.

[2] H. Baek, J. Kwak and J. Lee, *Non-Preemptive Real-Time Multiprocessor Scheduling Beyond Work-Conserving*, in 2020 IEEE Real-Time Systems Symposium (RTSS), Houston, TX, USA, 2020 pp. 102-114.

[3] S. K. Baruah, *The Non-preemptive Scheduling of Periodic Tasks upon Multiprocessors*, Real-Time Systems Journal, vol. 32, pp. 9–20, 2006.

[4] J. Carpenter, S. Funk, P. Holman, J. Anderson, and S. Baruah, *A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms,* in *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*, Chapman and Hall/CRC, pp. 1–19, 2004.

[5] F. G. Filip, *A Decision-Making Perspective for Designing and Building Information Systems.* International Journal of Computers Communications & Control, [S.l.], v. 7, n. 2, p. 264-272, sep. 2014. ISSN 1841-9844.

[6] M. R. Garey, D. S. Johnson, and S. Michael, *Computers and Intractability: A Guide to the Theory of NP-completeness,* in *Books in mathematical series*, W. H. Freeman publisher, 1979.

[7] N. Guan, W. Yi, Z. Gu, Q. Deng, and G. Yu, *New Schedulability Test Conditions for Non-preemptive Scheduling on Multiprocessor Platforms,* in *2008 Real-Time Systems Symposium*, pp. 137–146, Nov 2008.

[8] K. Jeffay, D. F. Stanat, and C. U. Martel, *On non-preemptive scheduling of period and sporadic tasks,* in *Proceedings Twelfth Real-Time Systems Symposium*, pp. 129-139, Dec 1991.

[9] S. Nogd, G. Nelissen, M. Nasri and B. B. Brandenburg, *Response-Time Analysis for Non-Preemptive Global Scheduling with FIFO Spin Locks*, 2020 IEEE Real-Time Systems Symposium (RTSS), 2020, pp. 115-127.

[10] M. A. Palomera Perez, H. Benitez-Perez and J. Ortega-Arjona, *Coordinated Tasks: A Framework for Distributed Tasks in Mobile Area Networks,* vol. 5, May 2011.

[11] S. Kato and N. Yamasaki, *Scheduling Aperiodic Tasks Using Total Bandwidth Server on Multiprocessors,* in *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, vol. 1, pp. 82–89, Dec 2008.

[12] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, *Review of job shop scheduling research and its new perspectives under Industry 4.0,* Journal of Intelligent Manufacturing, vol. 30, No. 4, pp. 1809–1830, Apr 2019.

Cite this paper as:

Hermosillo-Gomez, J. A.; Benitez-Perez, H. (2021). Job-shop scheduling over a heterogeneous platform, *International Journal of Computers Communications & Control*, 16(3), 4221, 2021.