INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2025, VOL. 71, NO. 2, PP. 397–402
Manuscript received February 17, 2025; revised May, 2025. doi: 10.24425/ijet.2025.153585

A boundle of on-line algorithms for scheduling computational tasks

Dariusz Dorota, Czeslaw Smutnicki

***Abstract*—We deal with the problem of scheduling the set of computational tasks on parallel identical processors. Each task needs a predefined number of processors to perform. The problem is known in scheduling theory and has been considered up to now by a few authors. Starting from the formal original description of the problem, we provide a mathematical model and then propose, at first, the solution method in the deterministic case. In fact, the paper focuses chiefly on the non-deterministic variant of the problem. We have proposed several online algorithms for this case. These algorithms are evaluated through competitive analysis and experiments. The practical application of the problem can be found in embedded systems with increased dependability obtained through hardware and software redundancy.**

***Index Terms*—Multiprocessor tasks, on-line scheduling**

I. Introduction

Tracing on Web the tendency in development of the electronic devices dedicated for control systems, one can observe in recent years: (a) the significant reduction of the production cost of a chip, (b) unification (standardization) of chips, (c) chips' miniaturization, and (d) replacing crucial functions of a device by programs (passing to programmable mode). The feature (d) induces us to focus on designing more advanced algorithms perceived as programs (software controller). Moreover, features (a) and (d) allow us to introduce some redundancy of hardware and software to increase dependability. Redundancy means that automatic decision is made as the result of "voting" among the predefined subset of identical processors which perform the same functions and run the same programming code for identical data, see Fig. 1 for details. From the formal point of view the mentioned software and hardware redundancy lead to the so-called multiprocessor task scheduling. This problem has appeared in the literature; however, the approach presented in the paper is new.

The possible applications are among others in controllers used in critical civil objects (monitoring, energy transmission, communication, transport), military and cosmic installations, IoT, IoV, and IoR. More practical applications one can find in our paper [10].

We consider a chiefly more realistic problem where the set of tasks is unknown in advance and tasks arrive at random time moments. In this case, it is justifiable to use an on-line scheduler. Thus, we consider several on-line algorithms that provide competitive as well as experimental evaluation of their quality.

D. Dorota is with Cracow University of Technology, Crakow, Poland (e-mail: ddorota@pk.edu.pl).

C. Smutnicki is with Wrocław University of Science and Technology, Wrocław, Poland (e-mail: czeslaw.smutnicki@pwr.edu.pl).

II. The problem

Let us denote the set of tasks $T = \{1, 2, 3, ..., n\}$ to be performed and the set of machines (processors, independent embedded devices) to perform these tasks. The machines are identical and work in parallel. The task $i \in T$ is processed synchronously on $a_i \leq m$ processors. The task can be perceived as identical copies of a_i of the program running simultaneously. The processing time of the task is denoted by $p_i > 0$, $i \in T$. The solution of the problem is defined by the pair (S, V), where $S = (S_1, S_2, \ldots, S_n)$ defines task starting times and $V = (V_1, V_2, ..., V_n)$, $V_i \subseteq M$, $|V_i| = a_i$ defines processors allocated for the task. In any moment of time, the number of machines allocated for tasks cannot exceed m. For each pair of tasks $i, j \in T$, which uses conflicting sets of processors, it has to be processed either $i \rightarrow j$ or $j \rightarrow i$. Our goal is to find (S, V), which meets time and resource constraints and minimizes certain quality indicators. In this paper, we use the criterion "to estimate the length of the schedule" $C_{\max} = \max_{i \in T}(S_i + p_i)$. This criterion also maximizes the degree of machine utilization. The problem is NP-hard.

To establish the location of the stated problem in the well-known taxonomy $\alpha|\beta|\gamma$, we refer to notation [15] which operates on the problem structure α, constraints β, and the goal function γ. Embedding the problem considered in the common taxonomy, we skip an extension of α that traditionally covers parallel processing, [27]. Instead, we prefer some extension of the meaning β. We use symbol $a_i = k$ in the

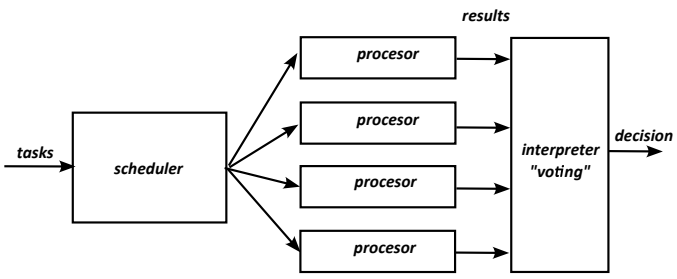

Fig. 1. Structure of the system

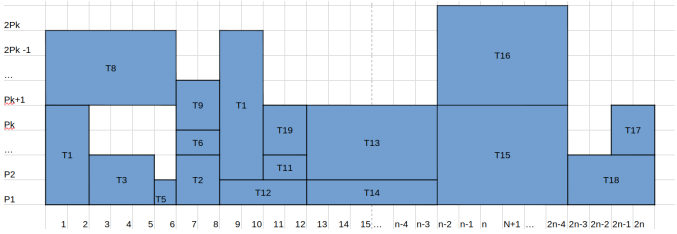

Fig. 2. Example of Gantt chart for some solution of multiprocessor scheduling

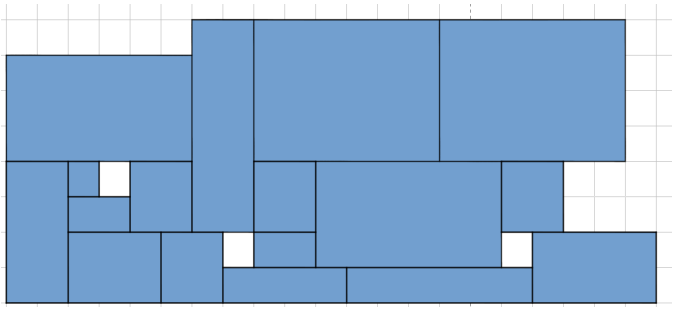

Fig. 3. Non-guillotined SCP as the special case of multiprocessor scheduling

case where all tasks need the same number k of processors. The symbol $a_i \leq k$ is used in the case where tasks need various (but bounded) a_i. Unbounded (free) a_i is marked by a_i alone. The proposed taxonomy is an extension of this from [2], [13], [18], [20], [27]. Accordingly to the proposed taxonomy, we consider the problem denoted as $P|\beta|C_{\max}$, where $\beta \in \{\circ, a_i = k, a_i \leq k, a_i, pmtn, prec\}$.

III. Deterministic scheduling

We start with a certain formulation of the problem in terms of the deterministic scheduling notions. This model provides the optimal reference value of the criterion for the competitive analysis carried out in the sequel. Let us assume that T is known and that a_i, p_i, $i \in T$ are also known. Next, we assume that time is discrete, that is, $t = 0, 1, 2, \ldots, H$, where H is the horizon of the schedule. This fact holds if only p_i, $i \in T$ are integers. Let us define the set of tasks $A_t(S, V)$ processed in the time interval $[t-1, t]$. Clearly, $A_t(S, V)$ depends on S, V and t. The formal definition is

$$A_t(S, V) = \{i \in T : \ t - p_i \leq S_i \leq t\}. \tag{1}$$

The optimization problem is to minimize the goal function (2) under under constraints (3)–(5), where where S and V are decision variables.

$$\min_V \min_S \max_{i \in T} (S_i + p_i) \tag{2}$$

$$(S_i + p_i \leq S_j) \vee (S_j + p_j \leq S_i),$$

$$V_i \cap V_j \neq \emptyset, \ i, j \in T, \ i \neq j, \tag{3}$$

$$\sum_{i \in A_t(S,V)} a_i \leq m, \ t = 1, 2, \ldots, H \tag{4}$$

$$S_i \geq 0, \ i \in T. \tag{5}$$

The goal function (2) is the makespan, which is minimized. Constraint (3) ensures sequencing of tasks having conflicting set of resources. Constraint (4) restricts the capacity of the system. Note that $\max_{i \in T}(S_i + p_i)$ in (2) can be converted to linear expression as follows, where C means the makespan

$$\min_V \min_S \min_C C \tag{6}$$

$$C \geq S_i + p_i, \ i \in T. \tag{7}$$

Similarly, the disjunctive condition (3) can be converted to a linear expression with the help of an auxiliary binary variable; see also [27] for details. To this aim we introduce binary variable

$$x_{ij} = \begin{cases} 1 & i \to j \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

From (3) we obtain for some large number K

$$S_i + p_i - S_j \leq K(1 - x_{ij})$$

$$S_j + p_j - S_i \leq K x_{ij},$$

$$V_i \cap V_j \neq \emptyset, \ i, j \in T, \ i \neq j, \tag{9}$$

This has an influence on the form of the goal function (2)

$$\min_V \min_x \min_S \min_C C \tag{10}$$

Unfortunately, the constraint (4) cannot be converted to linear. The recommended approach to the solution algorithm goes through the Lagrange function and dual prices; see [27] for details.

Problem (2)–(5) can alternatively be perceived as a special case of "project scheduling" under constraints imposed on discrete renewable resources with a constant amount accessible in time (RCPSP) [17]. Although the solution methods recommended for RCPSP can be applied there, [17], nevertheless we do not find such an approach in the literature. Problem (2)–(5) can also be perceived as a special case of the "cutting strip problem " (SCP), see Fig. 3 for example, see also [18] for details of the formulation, description, and solution methods of SCP. We also found other methods of modeling problems (2)–(5) by using a special class of graphs; see [22] for details. The analysis of deterministic case will not be developed in the sequel, since it plays only an auxiliary role.

IV. Uncertain scheduling

In this section we refer to the effective problem with uncertain data, namely task set T is unknown in advance and data p_i, a_i, $i \in T$ are also unknown. The considerations presented here are continuation of the research reported in [11], however, here they are built on a different ground than the one presented in [8], [9]. Note that there may exist various types of uncertainty; see [11] for details, namely, among others: unknown the set T; unknown task inflow; unknown task processing time; unknown task transfer time; unknown moments of processor breakdowns; unknown policy of processor maintenance; unknown preceding constraints among tasks; unknown resource needs required by the task. Uncertainty can

be modeled and analyzed through: (a) queening approach, (b) stochastic scheduling, (c) fuzzy scheduling, and (d) on-line scheduling. Approach (a) assumes that we have an infinite inflow of tasks. Cases (b) and (c) assume that T is known but unknown parameters remain p_i, $i \in T$, which are random or fuzzy, respectively. Thus, in this paper, we consider case (d) assuming that task set is not known in advance, tasks arrive at random moments of time and in random sequence. Note that the inflow has no stochastic features.

V. Online scheduling

On-line scheduling means that the decision referring to which task is processed next is made immediately after the tasks arrive. The fundamental difference between online and offline working mode is the access to data: (a) in the offline scheduling mode, we know all data in advance, see deterministic problem in Section III; (b) in the online mode, data are accessible (fully or partially) step-by-step with the arriving tasks. Variant (b) also includes cases with incomplete data or with no deterministic events, [21], [6].

The online schedulers are considered to be suitable for real applications. There are commonly classified accordingly: (A) *online list*; (B) *online time*; see, for example, [1], [4], [7], [12], [25]. The scheduling strategy *on-line list* assumes that the entry of the tasks randomly without any statistical distribution, alone, and all the data of the task (namely the processing time, the resource requirements) becomes known when the task has appeared. Immediately, after the task arrives, the scheduler has to plan task processing (namely assigns resources as well as determines the processing order) and the made decision has to remain unchanged in the future. This implicitly means that tasks are considered in the sequence of their arrivals (scheduler works in FIFO mode). The formerly mentioned *on-line time* model allows tasks to come randomly but may arrive contemporaneously. Some data associated with an incoming task may be known (for example, a_i), other data (for example p_i) may remain unknown or not predictable even during task service. When p_i is unknown, we usually employ a certain preemptive service policy (such as, for example, Round Robin, RR). In this case, the scheduler has to choose among all tasks waited in the queue (these new and these postponed) by using certain priority rule. In accordance to this classification, we consider in the paper only algorithms from the category online list.

The scheduling algorithms are classified according to the strategy for making the decision in the *deterministic* or *random* way, [1]. In the literature, we found other classifications of online algorithms [1], [26]: clairvoyant, nonclairvoyant, with adversaries. Skipping the formal definitions of these cases we will only refer the reader to papers [1], [26]. Taking into account this classification, our investigations refer to a specific class of algorithms, namely, list scheduling with deterministic decision-making. The rank of on-line algorithms has been established through competitive analysis called sometimes *analysis of the worst case* or *analysis of the pessimistic case*). This approach establishes the relation between the value of the goal function provided by the online algorithm A compared to the best value of the goal function OPT achieved by the off-line algorithm calculated with complete posterior knowledge of the data, for example, I. Formal expressions on competitive ratio are varied depending on the algorithm type. In the sequel, we use the coefficient ratio for the deterministic on-line algorithm defined as follows.

$$f(A, I) \leq r \cdot f(OPT, I) + \alpha. \tag{11}$$

The general aim is to find the minimal r that is true for the entire population of instances I. The beneficial feature of r is the independence from I, so it is objective. On the other hand, the poor feature of r follows from the observation that extreme instance I theoretically exists but rarely occurs. Therefore, practitioners are interested in prevailing cases in experimental analysis on a sample of instances that appeared in industrial practice. Employing "no free lunch theorem" [28] and philosophy of choosing "suitable sample of instances", the numerical experiments provide only supplementary, subjective evaluation of algorithm quality. Therefore, in this paper, we provide several algorithms that focus mainly on theoretical evaluation of their properties and verifying these features in some experimental research.

VI. Non-preemptive tasks

In this section, we propose a few scheduling algorithms for the case $1 \leq a_i \leq k \leq m$ without task preemption. These algorithms have been designed through enhancing some appropriate off-line algorithms formulated for $a_i = 1$, see e.g. [13]. First, we refer to paper [16] which provided the online list algorithm of scheduling for case $a_i = 1$, hereinafter called the algorithm LIST). Tasks are analyzed in the order of their arrival date. The scheduler works according to the following rule "if the processor is free, then plan to schedule successive waiting tasks from the list representing the queue". This algorithm has been shown to be the best among all known algorithms for the case considered [5], [16].

The natural extension of LIST on multiprocessor scheduling is called $m - LIST$ and defined as follows "if a_i processors are free, use a_i processors to perform the task Note that LIST and m-LIST apply the FIFO service rule. The fundamental role of the scheduler is to allocate resources $V_i \subseteq M$ and to plan the expected processing period $(S_i, S_i + p_i)$. These generally formulated rules of resource allocation, for LIST as well as m-LIST, become ambiguous when several equally evaluated allocations are available. In order to fix this problem, we assume that the allocation with the smallest indices is preferable.

Algorithm 1:
We denote by C_k, $k \in M$ the moment of time such that the processor k is constantly idle after C_k. The initial conditions are $C_k = 0$ for all $k \in M$. The iterative step is as follows. Select from the list of tasks that wait to be processed for the successive task and denote it i. Calculate the earliest time S_i so that a_i processors are available after S_i. Formally we need to find subset $V_i \subseteq M$, $|V_i| = a_i$ so that

$$S_i = \min_{V_i \subseteq M} \max_{k \in V_i} C_k \tag{12}$$

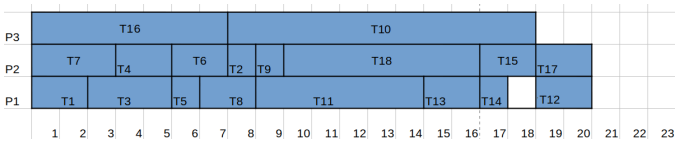

Fig. 4. Instance of the schedule provided by on-line algorithm m-LIST, for $a_i = 1$, $m = 3$

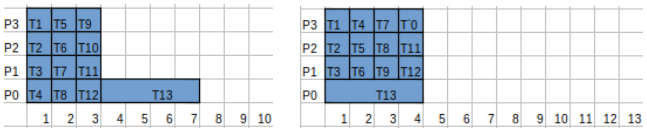

Fig. 5. $m = 4$, $n = 13$, online LIST (left) versus off-line OPT (right).

Then, plan task i in the interval $[S_i, S_i + p_i]$, allocate chosen processors V_i to the task, and modify appropriately $C_k = S_i + p_i$, $k \in V_i$. Repeat this procedure for successive tasks from the queue. ∘

Figure 4 shows the schedule for $n = 18$, $m = 3$ provided by LIST when the sequence of task arrivals is $(1, 7, 16, 3, 4, 5, 6, 8, 2, 10, 11, 9, 18, 13, 14, 15, 12, 17)$. We quote the result of the fundamental well-known literature; see Theorem 1.

Theorem 1: LIST has competitive ratio

$$r = 2 - \frac{1}{m}. \tag{13}$$

Figure 5 (left) provides the schedule for some (worst) instance I, the online algorithm LIST, and the sequence $(1, 2, \ldots, 13)$ of incoming jobs. It provides a makespan almost twice as long as OPT. The optimal off-line schedule OPT is given in this figure on the right. It is clear that for this case of LIST we have $r \to 2$ if $m \to \infty$.

We introduce the properties of m-LIST through an approximation of the problem data. To this end, we assume that $a_i = c$, $i \in T$. Approximation means that we replace c processors by single "aggregated artificial processor". Consequently, we can apply the Algorithm 1 to aggregated processors and artificial single-processor tasks. Applying Theorem 1 we conclude that

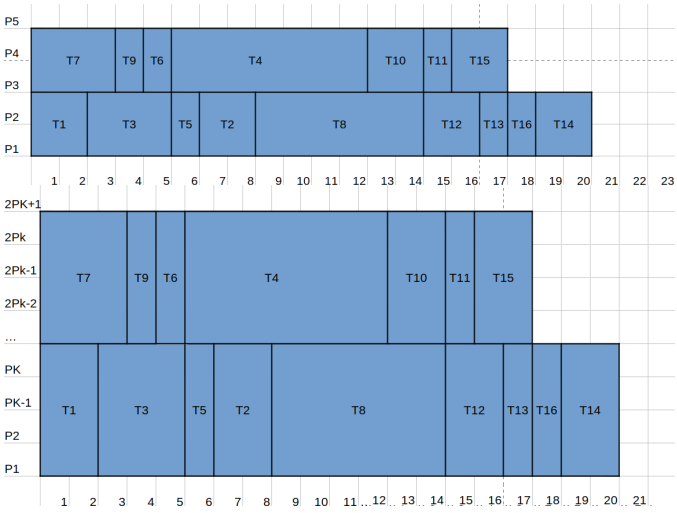

Fig. 6. m-LIST. $m = 5$, $n = 14$, $a_i = 2$, $i \in T$ (upper); $a_i = k$ (lower)

Theorem 2: m-LIST for $a_i = 2$, $i \in T$ has competitive ratio

$$r = \begin{cases} 2 - \frac{2}{m} & \text{if } m \text{ is even} \\ 2 - \frac{2}{m-1} & \text{if } m \text{ is odd} \end{cases} \tag{14}$$

Proof: By grouping processors, we find that there exist no more than $\lfloor m/2 \rfloor$ pairs, each of them leading to "aggregated artificial single processor" representing two physical processors. Let us assume that these pairs have the form for even m as follows $(1, 2)$, $(2, 3)$, ...,$(m-1, m)$ if m and for even as follows m $(1, 2)$, $(2, 3)$, ...,$(m-2, m-1)$. If m is odd, we skip the processor m because it is single (has no pair).

For proof needs, we convert the problem with data n, m, p_j, $\in T$, $a_j = 2$, $j \in T$, to the problem with data n', m', p'_j, $i \in T$ using the transformation: $n' = n$, $m' = \lfloor m/2 \rfloor$, $p'_j = p_j$, $j \in T$. Applying the previous theorem, we have

$$r = 2 - \frac{1}{m'}. \tag{15}$$

Substituting m' into (15) we obtain (14). This ends the proof. ∘

Extending the previous idea, we can formulate, without proof, the analogous theorem.

Theorem 3: m-LIST for $a_i = k$ has competitive ratio

$$r = 2 - \frac{1}{\lfloor \frac{m}{k} \rfloor}. \tag{16}$$

The function $r = 2 - \frac{1}{\lfloor \frac{m}{k} \rfloor}$ decreases depending on k and obtains the maximum value for $k = 1$. Considering the problem where tasks have various a_i, we note that the competitive ratio has to hold for any instance I, any data a_i, p_i, in particular for $a_i = 1$, $i \in T$. Therefore, the m-LIST competitive ratio for various a_i cannot be less than $r = 2 - \frac{1}{m}$.

Another inspiration for the online algorithm that can be applied for multiprocessor scheduling is provided by the MR algorithm [14]. MR uses specific technology to balance processor workloads. Balancing takes into account the set of tasks waiting for processing, as well as the plan of the already scheduled task. The competitive ratio r of MR tends to

$$1 + \sqrt{\frac{1 + \ln 2}{2}} < 1.921 \tag{17}$$

if $m \to \infty$, [14]. Because MR for is currently the best on-line algorithm, we will shown some its key elements which can be implemented in multiprocessor scheduling. Without losing generality, we assume that tasks arrive in the queue in the order $(1, 2, \ldots, n)$. Consider partial order, where tasks $(1, 2, \ldots, t)$ have already been planned. Iterations of the algorithm are indexed by t. Let us denote L_i^t the load of the processor i in iteration t. We sequence workloads in nonincreasing order,

$L_1^t \geq L_2^t \geq \ldots \geq L_m^t$. These workloads refer to processors $M_1^t \ldots M_m^t$, respectively. Note that M_1^t has the highest load, whereas L_1^t is the makespan. The *average workload* for processors $M_j^t, \ldots, M_m^t$ is defined as follows.

$$D_j^t = \frac{\sum_{k=j}^{m} L_k^t}{m-j+1} \tag{18}$$

We set $D^t = D_1^t$. The authors of MR use some predefined "auxiliary" numbers necessary to describe the algorithm, namely

$$c = 1 + \sqrt{\frac{1+\ln 2}{2}}, \quad i = \left\lceil \frac{5c - 2c^2 - 1}{c} \cdot m \right\rceil - 1,$$
$$k = 2i - m. \tag{19}$$

The schedule is called *flat* in iteration t if the following inequality holds.

$$L_k^{t-1} < \frac{2(c-1)}{2c-3} D_{i+1}^{t-1}. \tag{20}$$

The inequality (20) expresses that the workload of the workload processor $M_k^{t-1}t$ does not exceed the average load of the remaining processors $m-i$. If condition (20) is not satisfied, we say that the schedule is steep .

Algorithm 2:
Let us assume that we scheduled tasks $(1, 2, \ldots, t-1)$ for some t. We would like to plan the processing of task t, where p^t denotes its processing time. Check if the actual schedule is *steep* or *flat*. If the schedule is steep or $p^t + L_i^{t-1} > c \cdot D^t$, then plan to perform task t on M_m^{t-1}. Otherwise, plan the processing of this task on M_i^{t-1}. ∘

The methodology of designing algorithm m-MR (the version for multiprocessor scheduling) is analogous to m-LIST. Therefore, we omit the further details.

VII. Preemptive Tasks

In this section, we design online algorithms for tasks with preemption. We begin with some useful features. First, for $a_i = 1$ there exists an offline polynomial-time algorithm, [23]. The optimal makespan $C_{\max}^*$ is equal to

$$C_{\max}^* = \max\left(\frac{1}{m}\sum_{j=1}^{n} p_j, \max_{1 \leq j \leq n} p_j\right). \tag{21}$$

The detailed schedule follows from the MacNaughton algorithm, [23]. Let us consider now the on-line list scheduling algorithm for this case. We start from CVW, [5], which is also described in Algorithm 3. As previously, we denote the arrived tasks by successive integers $1, 2, \ldots, n$. Let us assume that tasks $1, 2, \ldots, t$ have already been scheduled and $t+1$ will be the next task to be scheduled. Let us denote by L_i^t the processor workload for some fixed t. We change the numeration of the processors $M_1^t \ldots M_m^t$, to obtain $L_1^t \leq L_2^t \leq \ldots \leq L_m^t$. The current optimal makespan is OPT^t. Next, we denote $S^t = \sum_{j=1}^{t} p_j$.

The strategy of CVW is to maintain some lightly loaded processors, to anticipate longer jobs that will come. CVW has been proved to satisfy the following inequality.

$$L_m^t \leq t \cdot OPT^t \tag{22}$$

$$\sum_{i=1}^{k} L_i^t \leq \frac{\alpha^k - 1}{\alpha^m - 1} S^t, \quad 1 \leq k \leq m \tag{23}$$

where $\alpha = \frac{m}{m-1}$. To describe the algorithm we define auxiliary value

$$r \stackrel{\text{def}}{=} \frac{\alpha^m}{\alpha^m - 1} \tag{24}$$

.

Algorithm 3:
Let us have already planned all the incoming processing tasks up to t. The next scheduled task is $t+1$. Find OPT^{t+1} using (21). If $L_m^t + p_{t+1} \leq r \cdot OPT^{t+1}$, then locate whole $t+1$ task on M_m^t. Otherwise find $\ell = \min\{i : 1 \leq i \leq m, L_i^t + p_{t+1} \geq r \cdot OPT^{t+1}, \}$. Allocate part $r \cdot OPT^{t+1} - L_\ell^t$ of $t+1$ in M_ℓ^t, and allocate the remaining part of $t+1$ in $M_{\ell-1}^t$.

Theorem 4: CVW has competitive ratio

$$r = \frac{1}{1 - (1 - \frac{1}{m})^m} \tag{25}$$

The coefficient r in (25) is equal to the predefined value (24). If $m \to \infty$, then $r \to \frac{e}{e-1} \approx 1.58$. There are no algorithms for $m \geq 2$ with a better competitive coefficient than CVW, [4], [5].

The implementation of CVW for multiple processor tasks $a_i = c > 1$ can be performed by aggregating the processor (similarly to m-LIST). The newly designed algorithm will be called further m-CVW. The quality evaluation follows from the two successive theorems.

Theorem 5: m-CVW for $a_i = 2$, $i \in T$ has

$$r = \begin{cases} \frac{1}{1-\sqrt{(1-\frac{2}{m})^m}} & \text{if } m \text{ is even} \\ \frac{1}{1-\sqrt{(1-\frac{2}{m-1})^{m-1}}} & \text{if } m \text{ is odd} \end{cases} \tag{26}$$

The proof is analogous to that from Theorem 2, therefore, will be skipped.

Theorem 6: m-CVW for $a_i = k$, $i \in T$, has r equal to

$$r = \frac{1}{1 - (1 - \frac{1}{[\frac{m}{k}]})^{[\frac{m}{k}]}} \circ . \tag{27}$$

Function

$$\frac{1}{1 - (1 - \frac{1}{[\frac{m}{k}]})^{[\frac{m}{k}]}} \tag{28}$$

decreases with k. It obtains maximum value $k = 1$. For the instances I with various a_i's the quality evaluation remains, so $r = \frac{1}{1-(1-\frac{1}{m})^m}$.

In addition to CVW, there exist other algorithms [24] for the preemptive case, which, however, have not been discussed further on.

TABLE I
MAKESPANS $C_{\max}$ OBSERVED FOR ONLINE ALGORITHM m-LIST FOR $m = 3$ AND $m = 4$

Instance		a_i			$C_{\max}$	
nr	n	1	2	3	m=3	m=4
1	9	4	2	3	85	70
2	13	5	4	4	170	152
3	15	5	5	5	185	168
4	19	5	7	7	245	224
5	22	6	6	10	305	270
6	21	8	7	6	335	290
7	32	11	8	13	440	355
8	36	10	8	18	640	589
9	43	11	10	22	780	696
10	48	12	12	24	825	750

VIII. COMPUTATIONAL EXPERIMENTS

Theoretical competitive ratio of the online algorithms refers to pessimistic instances I, which rarely occur in practice. However, practitioners are mainly interested not in theoretical evaluations but in the experimentally confirmed quality of the algorithm. In order to fulfill this aim, we performed some computational experiments. Being aware that all algorithms described in this paper require experimental analysis and this causes a large amount of calculations and experimental data, we provide only limited such research. We consider several instances for the online algorithm m-LIST with fixed configuration of a_i and random p_i chosen from the interval $[1, 10]$. Configuration of a_i is provides in Table I in the form of amount of task which require one, two or three processors. The tasks are analyzed in the order $(1, 2, \ldots, n)$. The makespan depends on the number of processors $m = 3, 4$ available in the embedded system.

IX. CONCLUSION

Competitive ratios for a few algorithm dedicated for multiprocessor scheduling in non-preemptive an preemptive cases have been proposed and proved. Worst-case evaluations have pure theoretical results formulated as theorems. However, pessimistic instances appear rather rarely in real scheduling systems. Thus, numerical tests usually provide results better than the worst case taking into account a random sample of instances or instances taken from practice. These findings seem to be closer to the probabilistic approach than to competitive analysis. Combined analyses are complementary.

REFERENCES

[1] S. Albers, "Online Scheduling. Introduction to scheduling", CRC Press, 2009.

[2] J. Błażewicz et al. "Handbook on scheduling: from theory to applications", Springer Science & Business Media, 2007.

[3] J. Błażewicz and Z. Liu, "Scheduling multiprocessor tasks with chain constraints", European Journal of Operational Research, 94(2), 231-241, 1996.

[4] W. Bożejko and E. Gawlińska, "Algorytmy szeregowania online". In: W. Bożejko and J. Pempera (eds.): "Optymalizacja dyskretna w informatyce, automatyce i robotyce" (Polish) Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 2012.

[5] B. Chen, A. Van Vliet and G. J. Woeginger, "An optimal algorithm for preemptive on-line scheduling" Second Annual European Symposium on Algorithms, Utrecht, The Netherlands, September 26–28, 1994, Proceedings 2. Springer Berlin Heidelberg, 1994.

[6] X. Chen, " Selected problems of online scheduling on parallel machines", PhD dissertation, Politechnika Poznańska, Poznań, 2014.

[7] R. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems", ACM Computing Surveys (CSUR), 43(4), 1-44, New York, NY, USA, 2011.

[8] D. Dorota, "Scheduling Tasks in a System with a Higher Level of Dependability". In: International Conference on Dependability and Complex Systems. Springer, Cham, 2019.

[9] D. Dorota, "Scheduling tasks with uncertain times of duration". In: International Conference on Dependability and Complex Systems. Springer, Cham, 2020.

[10] D. Dorota and C. Smutnicki, " On-line scheduling multiprocessor tasks in the non-predictive environment", LNCS, 2024.

[11] D. Dorota, "Szeregowanie zadań wieloprocesorowych w warunkach niepewności", Ph. Thesis (Polish), Politechnika Wrocławska, 2023.

[12] M. Drozdowski, "Scheduling for parallel processing", Springer, London, 2009.

[13] M. Drozdowski, "Scheduling multiprocessor tasks - an overview", European Journal of Operational Research 94(2), 215-230, 1996.

[14] R. Fleischer and M. Wahl, "Online scheduling revisited". European Symposium on Algorithms. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000.

[15] R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey", Annals of Discrete Mathematics 5, 287-326, 1979.

[16] R. Graham, "Bounds for certain multiprocessing anomalies", Bell System Technical Journal, 45, 1563-1581, Wiley Online Library, 1966.

[17] W. Herroelen, B. De Reyck and E. Demeulemeester, "Resource-constrained project scheduling: A survey of recent developments", Computers & Operations Research 25.4, 279-302, 1998.

[18] J. Hurink and J. Paulus, "Online algorithm for parallel job scheduling and strip packing", International Workshop on Approximation and Online Algorithms, Springer, 2007.

[19] S. Im, "Online scheduling algorithms for average flow time and its variants", University of Illinois at Urbana-Champaign, 2012.

[20] B. Johannes, "Scheduling parallel jobs to minimize the makespan", Journal of Scheduling 9, 433-452, 2006.

[21] R. M. Karp, "On-Line Algorithms Versus Off-Line Algorithms: How Much, Algorithms, Software, Architecture", Information Processing 92: Proceedings of the IFIP 12th World Computer Congress, Madrid, Spain, 7-11 September 1992.

[22] M. Makuchowski, "Problemy gniazdowe z operacjami wielomaszynowymi. Własności i algorytmy", PhD thesis. (Polish) Raporty Instytutu Cybernetyki Technicznej PWr. 2004, Ser. PRE; nr 37. 196 s.

[23] R. McNaughton, "Scheduling with deadlines and loss functions", Management Science, INFORMS, 1959.

[24] R. R. Muntz and E. G. Jr Cofmann, "Preemptive scheduling of real-time tasks on multiprocessors systems", Journal of the ACM, 17(2), 324-338, 1970.

[25] M. Pinedo, "Scheduling", Springer, New York, NY, 2015.

[26] K. Pruhs, S. Jiri and E. Torng, "Online scheduling", 2004.

[27] C. Smutnicki, "Algorytmy szeregowania zadań" (Polish), Oficyna Wydawnicza Politechniki Wrocławskiej, 2012.

[28] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization", IEEE Transactions on Evolutionary Computation, 1(1), 67-82, 1997.