An Innovative Task Scheduling Method Utilizing the Knapsack Algorithm in Heterogeneous Computing Systems

Lotfi Bendiaf[1], Ahmed Harbouche[2], Mohammed Amin Tahraoui[2], Fatima Zohra Lebbah[3,4]
[1]LME Laboratory, Hassiba BenBouali University of Chlef, 02000, Algeria
[2]LIA Laboratory, Hassiba BenBouali University of Chlef, 02000, Algeria
[3]Higher School of Electrical and Energetic Engineering, Oran, 31000, Algeria
[4]Computational Intelligence and Soft Computing Team (CISCO), Laboratory of Research in Computer Science, Higher School of Computer Science, Sidi Bel Abbes, 22016, Algeria

Keywords: scheduling, dynamic scheduling, multi-processors system, knapsack problem, heterogeneous computing environments

Received: February 27, 2024

As with any technical field, IT systems become increasingly complex, which can jeopardize their performance in terms of both time and quality of response. Consequently, performance is recognized as a primary criterion to meet IT users' requirements and expectations. Task scheduling techniques aim to fully exploit the power of multicore processors to enhance computer performance and optimize the management of computer networks. However, task scheduling, particularly in Heterogeneous Computing Systems (HCS), presents a challenging NP-hard problem.In this article, we propose a novel approach to address the task scheduling problem in HCS. We introduce a Knapsack-based Co-Scheduling Algorithm for Task Allocation KaCoSTA, which integrates the knapsack problem with dynamic programming. We conducted experiments on KaCoSTA and compared it with state-of-the-art methods, applying it to various task sets and processor configurations. The experimental results demonstrate that KaCoSTA significantly reduces the makespan and improves Resource Utilization (RU). Specifically, KaCoSTA optimizes task allocation and resource management while minimizing makespan, making it an efficient solution for task scheduling in heterogeneous computing environments.

Povzetek: KaCoSTA je pristop za sočasno razporejanje, ki temelji na problemu nahrbtnika in združuje dinamično programiranje ter algoritem nahrbtnika za optimizacijo dodeljevanja nalog in upravljanja virov v heterogenih računalniških sistemih, s čimer učinkovito skrajša čas obdelave in izboljša splošno učinkovitost sistema.

1 Introduction

In recent decades, the field of computer science has seen rapid advancements in both hardware and software. New technologies, such as multi-core processors, multi-processor systems, and distributed computing, have emerged to enhance the efficiency of IT systems in terms of time and quality of service. However, despite hardware development, conceiving modern materials for HCS remains insufficient without efficient software techniques. Task scheduling in an HCS environment is particularly challenging due to the NP-hard nature of the problem [1]. It requires a balancing solution between minimizing schedule length and maximizing processor utilization. To tackle task scheduling problems, according to the characteristics of the corresponding systems and their environments, algorithms with specific criteria and limits have been proposed. Several scheduling techniques have been proposed to assign tasks to a processor that provides the best performance in terms of Quality of services. Although, as shown in [2], in order to meet the requirements of task precedence and minimize schedule length, it is impossible to map all the tasks to the most efficient processor. Among these algorithms, Min-Min [2, 3, 4] and Max-Min algorithms [5, 7] succeed in minimizing makespan, but they often fall short in terms of RU and scalability. On the other side, the algorithms HEFT (Heterogeneous Earliest Finish Time) and CPOP (Critical Path On a Processor) [8, 9] prioritize tasks based on static rankings, which are not well adapted to dynamic HCS environments. Moreover, the authors in [10] presents an improved elite genetic algorithm for disassembling complex equipment with asynchronous tasks, taking into account priorities, mutual interference and human resources. Li et al. [11] proposed a load balancing and ants' colony optimization algorithm for cloud task scheduling, demonstrating improved load balancing and reduced makespan. Parsa and Entezari-Maleki [12] introduced RASA which uses Min-Min strategy to execute small tasks before the large ones, and applies Max-Min strategy to avoid delays in the execution of large tasks and to support concurrency in the execution of large and small tasks in grid environments. Hybrid scheduling algorithms for col-

laborative mobile edge computing in 5G networks, such as Raeisi-Varzaneh et al. in [13], who published a survey on fairness and load balancing indicators in scheduling. Bittencourt et al. proposed in [14] a genetic algorithm for task scheduling in hybrid cloud environments, optimizing resource allocation and reducing execution time. Beck and Werner presented a QoS-aware [3] task scheduling algorithm for mobile edge computing, focusing on meeting the quality of service requirements while optimizing the resources' utilization. In [15], Rodriguez and Buyya introduced a multi-objective scheduling algorithm for dynamic workloads' multi-processing environments, balancing workload distribution and minimizing execution time [11].

Despite the variety of existing techniques, it is hard to adequately address the complexity of task dependencies and dynamic resource availability in HCS. This highlights the need for more adaptive and efficient scheduling algorithms. In this article, we propose a Knapsack-based Co-Scheduling Algorithm for Task Allocation (KaCoSTA) to avoid the deficiencies cited above, by combining the knapsack problem with dynamic programming to optimize task allocation and resource management. KaCoSTA is conceived to minimize makespan while maximizing RU, and adapting to changes in task arrival times and system states. The experimental results compared to SOTA algorithms demonstrate the success of our approach applied in heterogeneous computing environments.

The remainder of this article is structured as follows: Section 2 presents the state of the art, discussing various approaches and techniques in the literature. Section 3 provides a detailed explanation of our approach and its underlying principles. Section 4 includes experimental results and comparisons with existing methods. Finally, Section 5 concludes the paper with a summary and perspectives for future work.

2 State of the art

Because of the strong relationship between performance in IT systems and scheduling, several scientific works have been proposed in this field. Indeed, algorithms [16, 17] performed in multiple steps to solve the problem of matching application needs with resource availability without neglecting Quality of Service (QoS). In HCS, one of the goals of tasks' scheduling is to achieve high system throughput while considering available computing resources and QoS. Thus, to meet minimal makespan requirements, the authors in [2] have proposed a technique to assign tasks to processors according to the minimum execution cost of each task computed on a different processor. However, in the majority of cases, the results obtained show an increase in the makespan measure, which is poor in terms of QoS. Therefore, several researchers, such as Ezzatti et al. [4, 2, 18], have proposed an improved implementation of Min-Min heuristic by taking into account QoS constraints. The researchers in [19] consider that a Genetic Algorithm (GA) should be performed naturally in parallel systems with multiple processing nodes. Thus, they proposed an appropriate allocation by applying genetic operators crossover and mutation. On the other hand, the authors in [20] have adapted a distributed algorithm for cloud systems and proposed a workflow scheduling algorithm that considers dynamic priority for the tasks. This approach undergoes a process of Min-Max normalization [5]. In addition, Jasim et al. [6] present an intelligent algorithm for scheduling tasks in cloud data centres, based on the Cuckoo intelligent methodology. The authors analyse in detail the different optimization methods such as genetic algorithms, greedy algorithms, Ant-lions optimiser and ant colony optimization. The proposed use of an algorithm based on the Cuckoo method is expected to improve the scheduling time and the optimization of resources in dynamic environments, which would contribute to the efficiency of cloud services. While we focus on the optimization aspect of the scheduling problem, Stützle et al. [7] introduced Max–Min Ant System (MMAS), which is an Ant Colony based optimization algorithm that considers ants as simple agents that progressively construct candidate solutions to treat an NP-hard static combinatorial optimization problem.

In this article, we propose a dynamic priority-based task scheduling algorithm for heterogeneous computing environments. Drawing an analogy to the knapsack problem algorithm [21], each node in the system is assigned an estimated Makespan Threshold (MT) value (see Equation 6). Specifically, MT represents the knapsack capacity, acting as the upper bound that should not be exceeded. This ensures that the sum of the assigned tasks' execution times stays within the MT. Our approach integrates resource utilization optimization at the processor level with Quality of Service (QoS) considerations.

2.1 Scheduling problem definition

Firstly, we introduce the scheduling problem in HCS context, followed by its corresponding model based on scheduling criteria, including processor computing speeds, total runtime, and system RU.

The scheduling system is defined by the quintuple $S = (T, E, P, K, MT)$, where $T = t_1, \ldots, t_n$ is the set of available non-preemptive tasks, $K = k_1, \ldots, k_m$ is the set of heterogeneous processors, E is the tasks' execution cost matrix where $E[i][j]$ represents the execution cost of task t_i on processor k_j, $P = p_1, \ldots, p_n$ is the set of task priorities (or task weights), and MT is defined as the makespan threshold.

Secondly, the system is represented by the directed acyclic graph (DAG), $G = (T, E_d)$, where $T = t_1, \ldots, t_n$ denotes the set of application tasks, and E_d denotes the set of edges that represent inter-task data dependencies. Task t_x cannot start execution until task t_y completes, if t_x is a child task of t_y.

The problem to be addressed is: 'How can tasks be as-

signed to processors without exceeding the MT, while considering RU constraints and Quality of Service (QoS)?'

2.2 Related works

For example, consider the heterogeneous environment S shown in Table 1, where the same set of tasks $T = t_1, t_2, t_3, t_4, t_5$ is assigned to three interconnected processors $K = k_1, k_2, k_3$, each with different processing speeds. Consequently, different execution costs are observed, as expected. In this system, it is assumed that all tasks are independent.

Processors	t_1	t_2	t_3	t_4	t_5
k_1	94	55	14	30	108
k_2	74	35	10	22	81
k_3	99	65	23	42	130

Table 1: Tasks execution costs in heterogeneous environment

A new version of Min-Min approach was proposed in[2], by He et al. to improve the original algorithm results. Effectively, by applying both algorithms on the same system, traditional Min-Min and QoS Guided Min-Min (see Figure 1), the makespan shows an enhancement of 8.85% from 113 to 103.

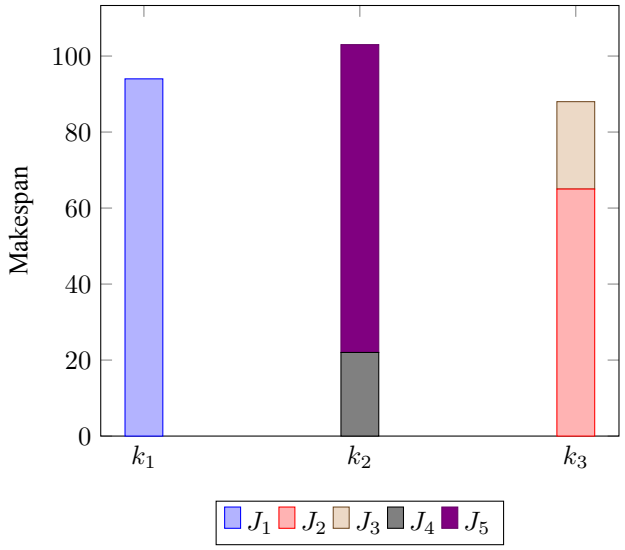

Figure 1: QoS guided Min-Min approach

Another study, called the Static Task Graphs Stratification, was presented in [22], this algorithm mainly focuses on multi-core load balance, by partitioning the tasks. Thus, at first, the independent tasks are assigned to distinct levels (see Figure 2). Then, the static task group scheduling algorithm will be applied to allocate these tasks for secondary cores. During the system's running, the tasks whose the running times are unpredictable, are allocated to secondary cores by applying the dynamic link algorithm.

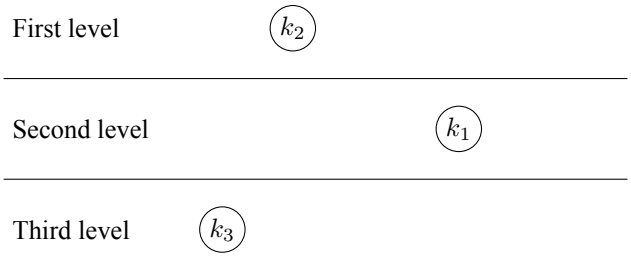

Figure 2: Graph stratification approach

The tasks $T_i, i = 1,..n$ are selected for cores $k_j \in K, j = 1,..m$ decreasingly, according their time-consuming. In other words, the none allocated task T_k that verifies the relationship Formula 1 is chosen to be processed, and continue until all tasks are mapped.

$$time_{total} + t_k > avg_{time} \tag{1}$$

where:

$avg_{time} = \sum_{i=1,..n} Ti/|K|$,

$K = k_1, k_2, ...k_m$ is the set of the available cores in the system,

$time_{total} = \sum_{i=1,..p, i \neq c} Ti$.

Our proposed approach, KaCoSTA, is compared with other task scheduling methods across various characteristics, as summarized in Table 2. This comparison highlights KaCoSTA's advantages in terms of efficiency, makespan reduction, and resource utilization.

3 The proposed approach

Dynamic programming methods are often used when solving the discrete optimization problems, since most of them are usually NP-hard. The knapsack problem the one of the most adopted discrete optimization problem in the literature. In other words, many tackled problems in the discrete optimization field have the same characteristics as those of the knapsack problem [23].

Definition 1. *Dynamic programming*
Dynamic programming (DP) is a method used for solving complex problems by breaking them down into simpler subproblems. It is particularly effective for optimization problems where the solution can be constructed from the solutions of overlapping subproblems. In the context of task scheduling, DP helps in determining the best assignment of tasks to processors by systematically evaluating all possible task allocations and selecting the one that minimizes makespan and maximizes RU [24].

The researchers are often attracted by applying algorithms and models coming from local search area, to raise the scheduling challenges. Thus, local optimization techniques and DP methods [24] have been proposed and succeeded to provide efficient schedulings.

algorithm	Makespan	RU	Complexity	Experimental Characteristics
Min-Min	High	Low	$O(n \cdot m)$	Not suitable for large tasks
Max-Min	Medium	Low	$O(n^2 \cdot m)$	Less efficient in dynamic environments
QoS Min-Min	Medium	Medium	$O(n \cdot m)$	Limited scalability, Less efficient for large datasets
HEFT	Medium	High	$O(n^2 + n \cdot m)$	Task prioritization, Difficult to implement
CPOP	Medium	Medium	$O(n \log n + n \cdot m)$	Task ranking
PETS	Medium	Low	$O(n \log n + n \cdot m)$	Task prioritization, Less efficient for non-preemptive tasks
PHTS	Low	Medium	$O(n \log n + n \cdot m)$	Path-based priority, Less adaptable
KaCoSTA	**Low**	**High**	$O(n \cdot m)$	Dynamic programming, Knapsack-based model Task prioritization, Tasks' run times are known beforehand, Deterministic conditions

Table 2: Characteristics of SOTA scheduling algorithms compared to our algorithm KaCoSTA

The aim of this work is to highlight the efficiency of the our approach that we conceive by analogy to the knapsack problem, to address the scheduling problem in heterogeneous computing environment. More precisely, our approach is based on the one dimension knapsack problem applied on multi-processors systems, whose the definition is given via Model 2.

3.1 Proposed model

Assume we have n items, their costs $c_i > 0$ and weights $w_i > 0, i = 1, 2, \cdots n$, and a knapsack carrying capacity R. In addition, suppose that $\sum w_i > R$, and $0 < w_i \leq R$.

The purpose is to fill the knapsack with a set of items, whose the sum of their capacities is maximal. As shown in Model 2, the problem is described as a discrete mathematical model, in terms of boolean variables $x_i, i = 1 \cdots n$, where:

$$\begin{aligned} &\text{Max F} = \textstyle\sum_{i=1}^{n} c_i \cdot x_i, \\ &s.c \begin{cases} \sum_{i=1}^{n} w_i \cdot x_i \leq R \\ x_i \in \{0, 1\} \end{cases} \end{aligned} \tag{2}$$

By analogy to model 2, consider a processor K_m, and its positive estimated $Makespan_{Threshold}$ (MT), a set $T = \{t_1, \ldots t_n\}$ of tasks, their respective processing positive costs (processing times) $e_1, \cdots e_n$ and their respective positive priorities $p_1, \cdots p_n$.

The purpose of our approach is to select the best subset T_k of tasks (see Definition 2), which compromises between the priorities $p_i/t_i \in T_k$ and the processing times $e_i/t_i \in T_k$. In addition, the sum of the processing times $e_i/t_i \in T_k$ should not exceed $Makespan_{Threshold}$.

Definition 2. *Scheduling Mathematical Model*
Let T be a set of tasks to schedule and T_k its tasks' subset which is assigned to the processor k_j. T_k is the best subset $T_k \subset T$ of tasks, that verify the linear program given below:

$$\begin{aligned} &Max\ Z = \textstyle\sum_{i=1}^{n} P_i \cdot x_i \\ &s.c \begin{cases} \sum_{i=1}^{n} E[i][j] \cdot x_i \leq Makespan_{Threshold} \\ x_i \in \{0, 1\} \end{cases} \end{aligned} \tag{3}$$

where:

P_i: the priority of the task t_i (See Definition 3),

$E[i][j]$: the execution cost of the task t_i on the processor j,

$Makespan_{Threshold}$ (See Definition 5): the estimated makespan threshold of the system,

x_i: the corresponding variable of the task t_i, where:

$$x_i = \begin{cases} 1 \text{ if } t_i \text{ is assigned to the processor} \\ 0 \text{ otherwise} \end{cases}$$

Definition 3. *Task Priority Calculation [25]*
The Upward Technique is applied for task prioritization while satisfying the precedence constraint [26]. The priority of each task is calculated by Eq. 4

$$P_i = avgEx_i + \max_{j \epsilon succ(t_i)} CC(i, j) + P_j \tag{4}$$

where:

$avgEx_i$: average execution cost of t_i on all processors

$succ(t_i)$: represent all successors of t_i on the scheduling system

$CC(i, j)$: Communication Cost between t_i and t_j

3.2 Methodology and mathematical formulations

Based on Model 3, our approach is a scheduling method that performs in a heterogeneous computing system. By means of DP, this approach combines between makespan (See Definition 4) minimisation and optimization of resources utilization.

Definition 4. *Makespan, C_{max}*
The total schedule length also defined as completion time, calculated in Eq. 5, is the maximum Finish Time (FT) of the exit task. [25]

$$makespan, C_{max} = Max_i(FT(exit\_task)) \quad (5)$$

Definition 5. *$Makespan_{Threshold}$ (MT), is an estimated value, which is defined as the average of the processing times of all jobs across processors. MT in this work is obtained by processing the following formula 6.*

$$MT = \frac{1}{|K|^2} \sum_{j=1}^{m} \sum_{i=1}^{n} E[i][j] \quad (6)$$

where:

$T = \{t_1, t_2, \ldots, t_n\}$ *is the set of tasks,*

$K = \{k_1, k_2, \ldots, k_m\}$ *is the set of processors,*

$E[i][j]$ *: Execution matrix, is the execution cost of task t_i on processor k_j.*

n *: Number of tasks on the queue.*

m *: Number of processors on the system.*

Basically, dynamic programming (DP) is used for solving complex problems by breaking them down into simpler sub-problems. Particularly, DP is effective for tackling problems from combinatorial optimization field, whose the solution can be constructed from the solutions of overlapping subproblems. In the context of task scheduling, DP helps in determining the best assignment of tasks to processors by evaluating systematically all possible task allocations and selecting which minimizes makespan and maximizes RU.

Our purpose, through DP techniques, is to maximize the total priority weight without exceeding the MT for the execution times of the tasks chosen. We interpreted our problem in a formal mathematical way as defined in Model 3.

The utilization of DP Recurrence, allows the leading of the best capacity $DP(k)$, which is the DP table entry for capacity k. For each task t_i and capacity k, $DP(k)$ is evaluated through Formula 7

$$DP[k] = \max(DP[k], DP[k - E[i][j]] + P[i][j]) \quad (7)$$

The boolean matrix $A(n \times m)$ gives the assignments' information of a task to a processor. In other words, a term of A is defined as follows:

$$A[i][j] = \begin{cases} 1 \text{ if a task } t_i \text{ is assigned to a processor } p_j \\ 0 \text{ otherwise} \end{cases}$$

Therefore, at each recursion, if $DP[k] > DP[k - E[i][j]]$ then $A[i][j] = 1$ (the matrix A is updated).

As given in Formula 8, the makespan MP is calculated, by using the last evaluation of A as the maximum completion time across all processors.

$$MP = \max_{j=1}^{m} \sum_{i=1}^{n} E[i][j] \cdot A[i][j] \quad (8)$$

After computing A then MP through the system, the resource utilization in the HCS is calculated by formula 9

$$\text{RU} = 100 - \frac{MP - Min(ProcessorUsage)}{MP} \quad (9)$$

where:

$Min(ProcessorUsage)$ is the least utilized processor in the scheduling system.

3.3 Approach process

The method process steps are outlined as follows:

1. **(Tasks Partitioning Dependency)**
   The first step consists of addressing tasks precedence constraint by partitioning the tasks, the independent tasks are grouped to distinct partition levels from DAG G(T, E_d) (Figure 3), and the result is defined in Figure 4.
   The following steps are applied to each partition

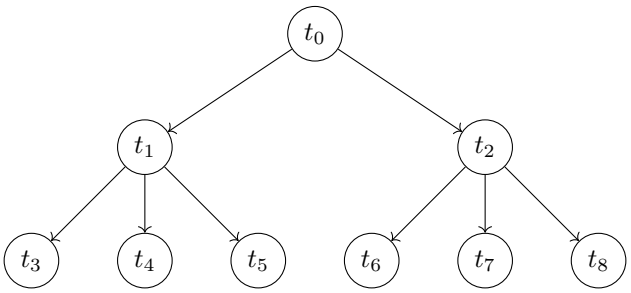

Figure 3: Example of DAG (Task Graph)

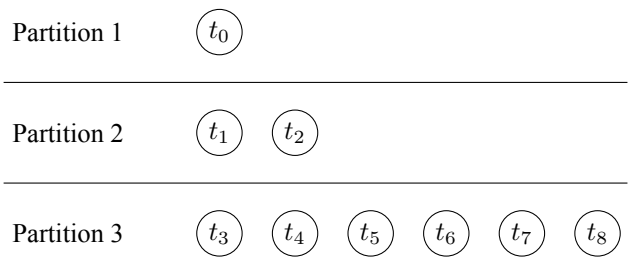

Figure 4: Tasks Partitioning Dependency (Step 1)

2. **(Estimation of the $Makespan_{Threshold}$)** (See Definition 5)

3. **(Modulation and decomposition of the MT)** $Makespan_{Threshold}$ modulation takes part of the DP process (See Eq 7), thus MT value is decomposed. For instance, in the case of $MT = 500$, the modulation process of $Makespan_{Threshold}$ provides the following modulation vector V_l defined as:
$V_l = 1, (1,2), (1,2,3), ...(1, ...500)$, where the MT is broken down into units, each unit called Modulated Makespan Threshold (MMT_l), and l represents the unit number.

4. **(Adding tasks process (Pre-scheduling phase))** Adding the tasks one by one according to their arrival times in the system, then browse the MTM_l modulation vector and process step 3 to each unit.

5. **(Fill the dynamic table)** Based on DP, completing the table (Tab) by dynamically adding the tasks by processing the following formula:

$$Tab[i, E[i-1][j]] = Max[Tab(i-1, E[i-1][j]), Tab(i-1, E[i][j] - E[i][j]) + P[i]] \quad (10)$$

6. **(Subset and Assignment Vector Construction)** If $Tab[i, E[i][j]] = Tab[i-1, E[i][j]]$
$\Rightarrow$ Task not assigned

$$A[i][j] = \begin{cases} 1 \text{ if } t_i \text{ is assigned to the processor } k_j \\ 0 \text{ otherwise} \end{cases}$$

where:

A : Assignment Matrix

7. **(Processing all the tasks in the system)** The process is repeated with none selected tasks for the remain processors, Unassigned tasks will be handled by the same process for the next node until all tasks will be treated, then assigned tasks are executed according to their priority.

3.4 Proposed algorithms

3.4.1 Knapsack based recursive algorithm for scheduling tasks allocation (KReSTA)

The first algorithm represents the first version of the approach.

Recursion is one of the popular problem-solving approaches in data structure and algorithms. Even some problem-solving approaches are totally based on recursion for example: decrease and conquer, divide and conquer, DFS traversal of tree and graphs, backtracking, top-down approach of DP and many others. Thus, time complexity analysis of recursion is critical to understand these approaches and improving our code's efficiency.

In the Figure 5 bellow, we present the recursive version.

Algorithm 1 KReSTA (Knapsack-based Recursive Scheduling algorithm for Tasks Allocation)

Require: T: Set of tasks $\{t_1, t_2, \ldots, t_n\}$,$E$: Execution costs matrix, where $E[i][k_{max}]$ is the execution cost of task t_i on processor k_{max},P: Priority weight of task t_i

Ensure: A: Assignment Vector, where $A[i][k_{max}] \leftarrow 1$ if task t_i is assigned to processor k_{max} and $A[i][k_{max}] \leftarrow 0$ otherwise

Order the processors from the most to the least efficient, then the most efficient processor k_{max} available in the queue is chosen.
Initialize Assignment Matrix $A[i][k_{max}] \leftarrow [0, 0, \ldots, 0]$

function DP(k)
  if $k = 0$ **then return** 0
  end if
  $max\_value \leftarrow 0$
  for $t_i \in T$ **do**
    if $E[i][k_{max}] \leq k$ **then**
      $value \leftarrow$ DP$(k - E[i][k_{max}]) + P[i]$
      $max\_value \leftarrow \max(max\_value, value)$
    end if
  end for
return max_value
end function
for $t_i \in T$ **do**
  if $E[i][k_{max}] \leq MT$ **then**
    $k \leftarrow MT$
    for $k < E[i][k_{max}]$ **do**
      if DP$(k) >$ DP$(k - E[i][k_{max}]) + P[i]$ **then**
        $A[i][k_{max}] \leftarrow 1$ ▷ Assign task t_i to processor k_{max}
      end if
      $k \leftarrow k - 1$
    end for
  end if
end for
Update the processor queue then repeat the process with remaining processors and tasks
  return A ▷ Assigned tasks' set to the processor k_{max}
The results tasks are queued then executed according to their priority

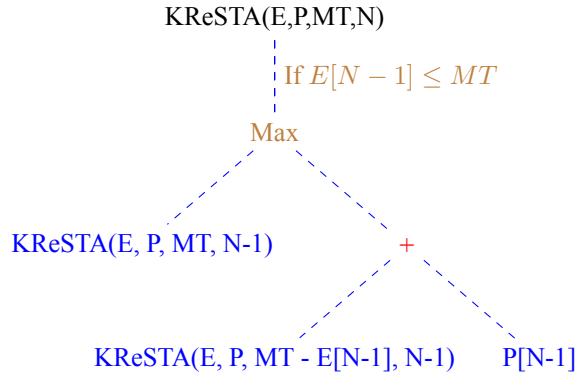

Figure 5: Knapsack Recursive Scheduling Tasks Allocation (KReSTA)

3.4.2 Knapsack based iterative algorithm for scheduling tasks allocation (KISTA)

The algorithm 2 mentioned bellow is featured by the introduction of iterative approach method to enhance time complexity.

First, we will estimate the difference between the two proposed algorithm 1 and 2 to highlight the optimization of the approach which intuitively brings an added value and thus a relevant aspect to the scheduling system overall. Scheduling algorithms performances are estimated with $\Delta time = FinishTime - BeginTime$ measure defined in Section 4, consequently, according to tasks-set processed, trivial difference can be noticed on Figure 6 bellow, Indeed, KReSTA algorithm shows exponential comportment while increasing number of tasks, whereas KISTA increases proportionally, which explains time complexity differences, we can observe for KReSTA with tasks-set : N=960, the algorithm needs 400.13 milliseconds to process, while with tasks-set : N=1920 the algorithm displayed the result in 848.57 milliseconds, hence, an increase of 112.07%, in the other side, KISTA with the same tasks-set : N=240, N=480 shows respectively 281.98 and 566.73 milliseconds which results in 100.98% increase, as a consequence, KISTA increases proportionally with N while KReSTA does not, for instance for N = 1920, KISTA is 41.63% better than KReSTA, as a result traduces a better performance compared to KReSTA in the first experiments as shown in Figure 6.

3.4.3 Co-scheduling algorithm

The final step consists of taking into consideration the different nodes in the system, which implies managing all tasks to represent the heterogeneous computing environment's behaviour.

In this section, the KaCoSTA algorithm (See Algorithm 3) consists to apply the approach to the whole system in heterogeneous environment with different processors processing capacities. First, the Makespan Threshold (MT) is calculated. Next, the processors are sorted in ascending order

Algorithm 2 KISTA (Knapsack-based Iterative Scheduling algorithm for Tasks Allocation)

Require: T: Set of tasks $\{t_1, t_2, \ldots, t_n\}$,$N = |T|$, E: Execution costs matrix, where $E[i][k_{max}]$ is the execution cost of task t_i on processor k_{max},P: Priority weight of task t_i, $tab[i][l] \leftarrow -1$ ▷ Matrix of optimization, initialised with -1 (Undefined), with l : modulation cursor

Ensure: A: Assignment Vector, where $A[i][k_{max}] \leftarrow 1$ if task t_i is assigned to processor k_{max} and $A[i][k_{max}] \leftarrow 0$ otherwise

Order the processors from the most to the least efficient, then the most efficient processor k_{max} available in the queue is chosen.
Initialize Assignment Matrix $A[i][k_{max}] \leftarrow 0$
for $t_i \in T$ **do**
  $mt \leftarrow 0$
  for mt to $MT + 1$ **do**
    if $i = 0$ Or $mt = 0$ **then**
      $t[i][mt] \leftarrow 0$
    else if $E[i-1] \leq mt$ **then**
      tab$[N][MT] \leftarrow$ Max(P$[i-1]$ + tab$[i-1][MT - E[i-1]]$, tab$[i-1][MT]$)
    else
      tab$[i][mt] \leftarrow$ tab$[i-1][mt]$
    end if
  end for
end for
$Res \leftarrow tab[N][MT]$
$mt \leftarrow MT$
for $i = N$ to 0 **do**
  if $Res \leq 0$ **then**
    Break
  else if $Res \leftarrow tab[i-1][mt]$ **then**
    Continue
  else
    $A[i-1][k_{max}] \leftarrow 1$ ▷ Assign task t_i to processor k_{max}
    $Res \leftarrow Res - P[i-1]$
    $mt \leftarrow mt - E[i-1]$
  end if
end for
Update the processor queue then repeat the process with remaining processors and tasks
  return A ▷ Assigned tasks' set on processor k_{max}
The results tasks are queued then executed according to their priority

Algorithm 3 KaCoSTA (Knapsack-based Co-Scheduling Algorithm for Tasks Allocation)

Require: T: Set of tasks $\{t_1, t_2, \ldots, t_n\}$, K: Set of processors $\{k_1, k_2, \ldots, k_m\}$, E: Execution costs matrix, where $E[i][j]$ is the execution cost of task t_i on processor p_j, P: Priority weights matrix, where $P[i]$ is the priority weight of task t_i, $Memo$: Memoization matrix $Memo[i][j]$ to avoid repeated computing

Ensure: A: Assignment matrix, where $A[i][j] = 1$ if task t_i is assigned to processor p_j and $A[i][j] = 0$ otherwise, Makespan: (See Definition 4)

Initialize Makespan Threshold (MT)
Sort processors in ascending order of their processing capacities.
for $k_j \in K$ **do**
    Initialize Assignment Matrix $A[i][j] \leftarrow 0$
    Initialize Memoization Table $Memo[j] \leftarrow$ Null ▷ Length $MT + 1$
end for

function DP(j, k)
    if $k = 0$ **then return** 0
    end if
    if $Memo[j][k] \neq$ Null **then return** $Memo[j][k]$
    end if
    $max\_value \leftarrow 0$
    for each task t_i in T **do**
        if $E[i][j] \leq k$ **then**
            $value \leftarrow$ DP$(j, k - E[i][j]) + P[i]$
            $max\_value \leftarrow \max(max\_value, value)$
        end if
    end for
    $Memo[j][k] \leftarrow max\_value$ **return** max_value
end function
for $t_i \in T$ **do**
    if $E[i][j] \leq MT$ **then**
        for $k = MT$ **down to** $E[i][j]$ **do**
            if DP$(j, k) >$ DP$(j, k - E[i][j]) + P[i]$ **then**
                $A[i][j] \leftarrow 1$ ▷ Assign task t_i to processor k_j
            end if
        end for
    end if
end for
    return A, MP.

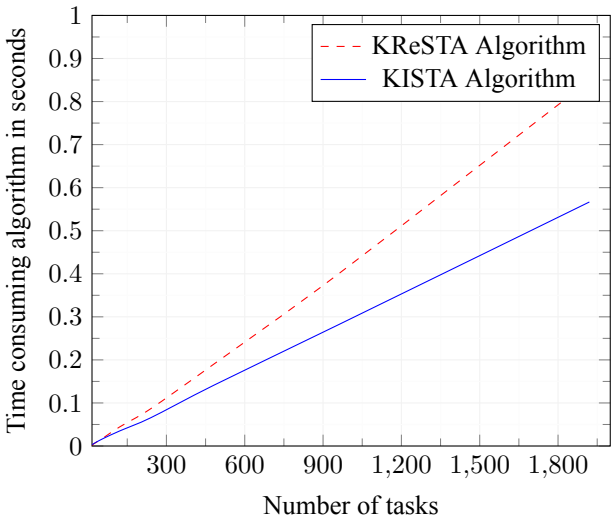

Figure 6: KISTA and the improved algorithm KReSTA comparison on total elapsed time with the same given number of tasks

based on their performance capacities. The KaCoSTA algorithm is then applied. Any tasks that remain unassigned, indicated by $A[i][j] = 0$ for all $i = 1, \ldots, N$, are queued for processing in the next iteration. This process continues until all tasks are successfully assigned to a processor during the pre-scheduling phase (See Figure 7).

3.5 Assumptions

In our approach we suppose that:

- All tasks are non-preemptive, meaning once a task starts execution on a processor, it runs to completion.
- Task execution costs are known and deterministic.
- Processors have different capacities and execution speeds, modeled by the execution costs matrix E.
- MakespanThreshold (MT) is estimated based on average processing costs and may be adjusted dynamically.

3.6 Complexity analysis of KaCoSTA algorithm

The complexity of the KaCoSTA depends on tasks number n and processors number m, The algorithm complexity can be broken down into several components based on its steps.

Proposition 1 (Initialization of Makespan Threshold (MT))**.** *The time complexity for calculating the Makespan Threshold (MT) is* $O(n \cdot m)$.

Proof. Calculating the average execution costs involves iterating over the execution cost matrix E, which has dimensions $n \times m$.

$$\text{MT} = \frac{1}{|K|^2} \sum_{i=1}^{n} \sum_{j=1}^{m} E[i][j]$$

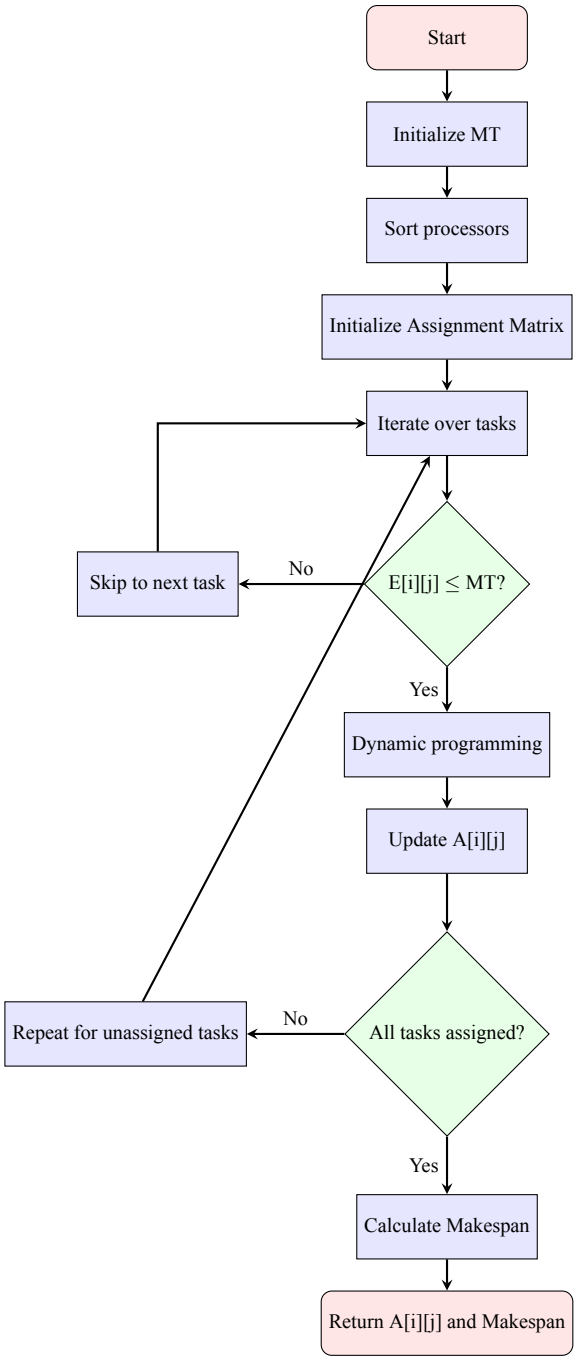

Figure 7: KaCoSTA Flowchart

Since we need to sum up all the elements in the matrix, the time complexity is:

$$O(n \cdot m)$$

Thus, the time complexity for calculating MT is $O(n \cdot m)$. □

Proposition 2 (Sorting Processors)**.** *The time complexity for sorting the processors is* $O(m \log m)$.

Proof. Sorting the processors by their processing performances involves using a sorting algorithm such as quicksort or mergesort, both of which have a time complexity of $O(m \log m)$.

$$\text{Time Complexity} = O(m \log m)$$

Thus, the time complexity for sorting the processors is $O(m \log m)$. □

Proposition 3 (Dynamic Programming with Memoization)**.** *The time complexity for solving the knapsack problem using dynamic programming with memoization is* $O(m \cdot MT \cdot n)$.

Proof. The dynamic programming approach involves filling a table DP for each processor k_j up to the Makespan Threshold (MT). For each task t_i, we update the table from MT down to the execution cost $E[i][j]$. The nested loops iterate over all processors, all tasks, and all capacities up to MT.

$$\begin{aligned} TimeComplexity &= O(m) \times O(MT) \times O(n) \\ &= O(m \cdot MT \cdot n) \end{aligned}$$

Thus, the time complexity for solving the knapsack problem using dynamic programming with memoization is $O(m \cdot MT \cdot n)$. □

Proposition 4 (Assignment and Update)**.** *The time complexity for the assignment and update step is* $O(n \cdot m)$.

Proof. For each task t_i, we need to update its assignment to a processor p_j and update the available capacities. This involves iterating over all tasks and all processors.

$$\text{Time Complexity} = O(n) \times O(m) = O(n \cdot m)$$

Thus, the time complexity for the assignment and update step is $O(n \cdot m)$. □

Proposition 5 (Overall Complexity)**.** *The total time complexity for the KaCoSTA algorithm is* $O(m \cdot MT \cdot n)$.

Proof. Combining the complexities of each step:

1. Initialization of Makespan Threshold (MT): $O(n \cdot m)$
2. Sorting Processors: $O(m \log m)$
3. Dynamic Programming for Knapsack Problem with Memoization: $O(m \cdot MT \cdot n)$
4. Assignment and Update: $O(n \cdot m)$

Summing these complexities:

$$O(n \cdot m) + O(m \log m) + O(m \cdot MT \cdot n) + O(n \cdot m)$$

Since $O(n \cdot m)$ and $O(m \log m)$ are dominated by $O(m \cdot MT \cdot n)$ and the MT is a value calculated beforehand and considered a constant, the overall complexity is simplified to:

$$O(m \cdot MT \cdot n)$$

Thus, the total time complexity for the KaCoSTA algorithm is $O(m \cdot n)$. □

4 Experimental studies

The experimental part of the paper primarily focuses on the performance disparities between our proposed techniques and existing approaches in the field of scheduling. Additionally, these tests highlight the importance of improving algorithm processing capabilities to produce results in less time, which is particularly crucial in scheduling for HCS, where every measure impacts the overall process.

This section compares our approach to traditional methods like Min-Min and QoS Guided Min-Min. Finally, we present the results of these improvements relative to well-known and recent approaches. Each test is conducted in a real-time context without making assumptions, ensuring that the results closely reflect real-world system environments.

Our experiments are conducted on a computer which has the following hardware setup: Apple macOS Ventura 13.2.1, 64-bit operating system, Apple Silicon M1 processor with 8-core GPU and 8GB/RAM. The algorithms are implemented in the Python language.

Regarding the datasets adopted in our experiments, We utilized a mix of synthetic and benchmark datasets [1] to evaluate the performance of our approach. We chose this kind of datasets to cover a variety of tasks' sizes and complexities, that reflect real-world scenarios in heterogeneous computing systems.

As given in Table 3, synthetic datasets are generated to include tasks with varied execution times and dependencies that mimic different workloads, and processor capabilities in an HCS environment.

Dataset	Number of tasks	Type of execution time	Number of heterogeneous processors (NHP)
A	<50	random, uniform	2, 3
B	500	random	10
C	1000	follow a normal distribution	15

Table 3: Characteristics of experimented datasets

More precisely, the chosen synthetic datasets allow to understand the impact of task distribution and processor heterogeneity on the efficiency of the implemented algorithms. Moreover, we taken in consideration benchmark datasets that offer real-world scenarios to validate the practical applicability of the proposed algorithm.

Through this part of experiment section, our objective is to compare our approach to algorithms and techniques that share the same specifications, like priority-based criteria, makespan minimisation and HCS involving.

[1] Experimented benchmark datasets are taken from http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html

4.1 Synthetic datasets experience

In this initial comparison, we aim to highlight the differences between various approaches by comparing KaCoSTA's results with those of Min-Min, Max-Min, and other methods in the literature. To achieve the aforementioned objectives, we consider the following example with independent tasks on three heterogeneous processors, as described in Table 4 and summarized in the following table:

Processors	t_1	t_2	t_3	t_4	t_5	t_6	t_7
k_1	77	56	23	29	9	40	31
k_2	98	90	54	50	22	65	76
k_3	82	70	40	43	17	51	45

Table 4: Tasks-set execution costs on three processors k_1, k_2 and k_3

First, the estimated $Makespan_{Threshold}$ is calculated as described in section 3 equation 6: $Makespan_{Threshold} \Rightarrow \frac{1}{3^2}(1068) = \frac{1068}{9} \approx 119$ Then according to the makespan threshold result, a schedule is given by means of DP using knapsack algorithm presented by our approach, the schedule result's makespan (Figure 8) does not exceed the $makespan_{Threshold}$. Finally the results are compared with Min-Min [27] and QoS guided Min-Min [2] in the following figure 8

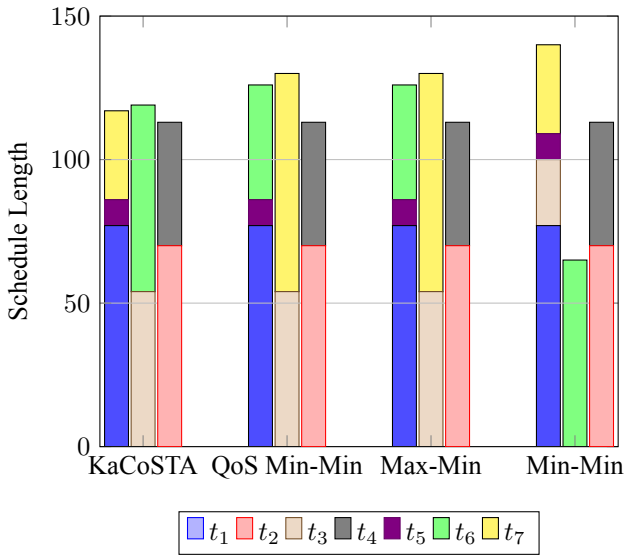

Figure 8: Min-Min, Max-Min, QoS guided Min-Min and KaCoSTA makespan comparison

For this initial comparison, the results demonstrate comparatively better performance of the proposed approach. Specifically, for the given set of tasks, KaCoSTA provides a makespan of 119, which does not exceed the estimated $Makespan_{Threshold}$. This result is achieved by assigning

t_3 and t_6 to processor 2, t_2 and t_4 to processor 3, and the remaining tasks t_1, t_5, and t_7 to processor k_3. In comparison, Min-Min results in a makespan of 140, while both QoS-Guided Min-Min and Max-Min yield a makespan of 130, leading to a 15% improvement in this test. It is noted that the complexities of KaCoSTA and Max-Min are $\mathcal{O}(n \cdot m)$ and $\mathcal{O}(n^2 \cdot m)$, respectively, where n is the number of tasks, m is the number of processors, and MT is the calculated makespan threshold defined in Eq. 6.

For fairness, we will use the same task set and values as in [28], as described in Table 5.

Processors	t_0	t_1	t_2	t_3	t_4	t_5
k_1	4	15	4	13	10	7
k_2	6	22.5	6	19.5	15	10.5

t_6	t_7	t_8	t_9	t_{10}
8	4	12	6	9
12	6	18	9	13.5

Table 5: Tasks-set execution costs on the processors K_1, K_2 [28]

In this part of the test, we assess the comparison between PHTS, PETS, Sorted Nodes in Levelled DAG Division (SNLDD), CPOP, HEFT, and our proposed approach, KaCoSTA.

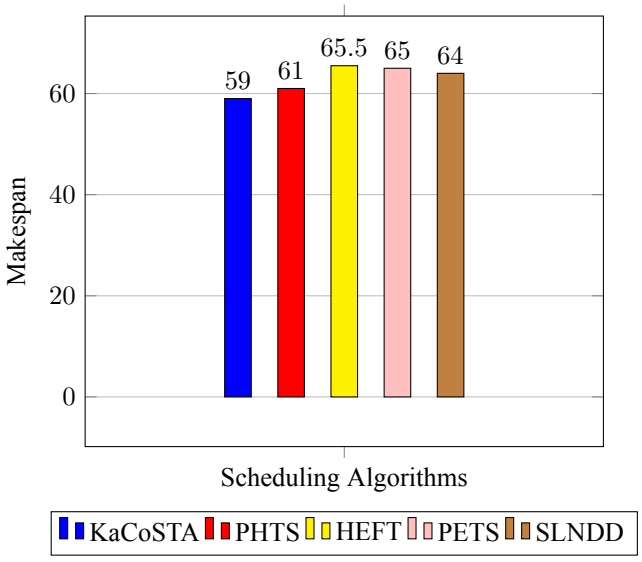

Figure 9: Makespan metric comparisons of all the four algorithms with KaCoSTA

The results of the experiment, as shown in Figure 9, demonstrate a clear advantage. Specifically, in terms of scheduling length, our approach outperforms the other mentioned methods.

Our heuristic, KaCoSTA, achieved these results with the following output: Makespan = $\max(59, 59) \Rightarrow$ makespan = 59 (see results in Figure 10).

Effectively, KaCoSTA achieves a makespan of 59, while the next best approach provides a makespan of no

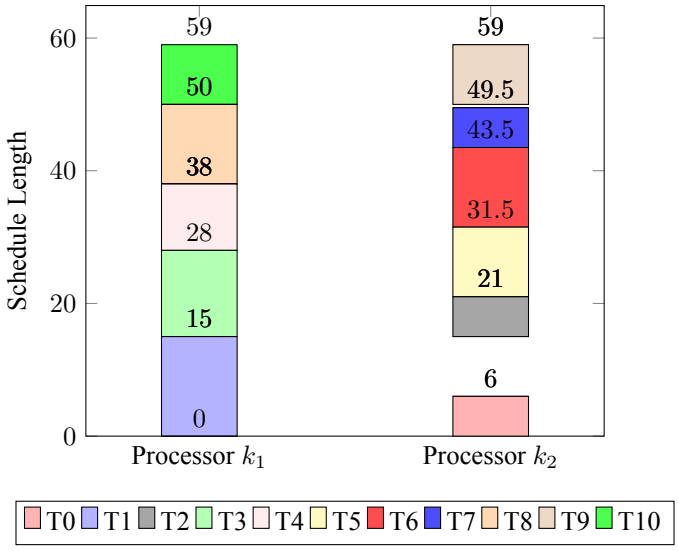

Figure 10: Schedule result generated by KaCoSTA

less than 61. This indicates that our proposed approach shows a 3.39% improvement over PHTS. Additionally, the utilization rates for processors k_1 and k_2 are both 59.

Let us consider the following representation of task set execution costs on three distinct processors, as presented in [29] and summarized in Table 6.

Task	k_1	k_2	k_3
t_1	14	16	9
t_2	13	19	18
t_3	11	13	19
t_4	13	8	17
t_5	12	13	10
t_6	13	16	9
t_7	7	15	11
t_8	5	11	14
t_9	18	12	20
t_{10}	21	7	16

Table 6: Comparison of tasks-set' execution cost values in each node k_1, k_2, k_3

Tasks dependencies represented by DAG in the Figure 11:

The obtained results presented in the following Figure 12, shown how the three approaches process, and how are tasks assigned to the given processors. The heterogeneous system example design is described by Ilavarasan et al. [29]

As illustrated in the figure, there is a clear difference in scheduling length (makespan) between our approach and the presented algorithms. Specifically, KaCoSTA records makespans of 35, 49, and 39 on Processor 1, Processor 2, and Processor 3, respectively, while PETS and CPOP result in makespans ranging from 76 to 85 across all processors. Notably, our algorithm performs 35.52% better than PETS.

This disparity can be attributed to the fact that PETS and

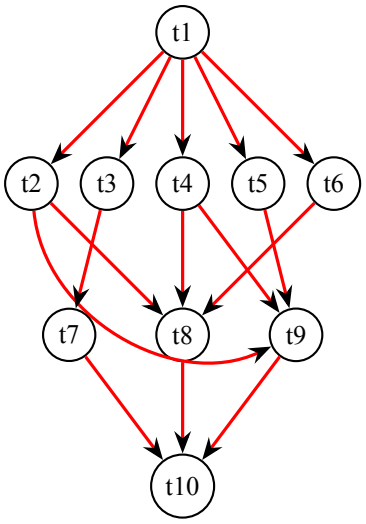

Figure 11: Tasks Dependencies DAG [9]

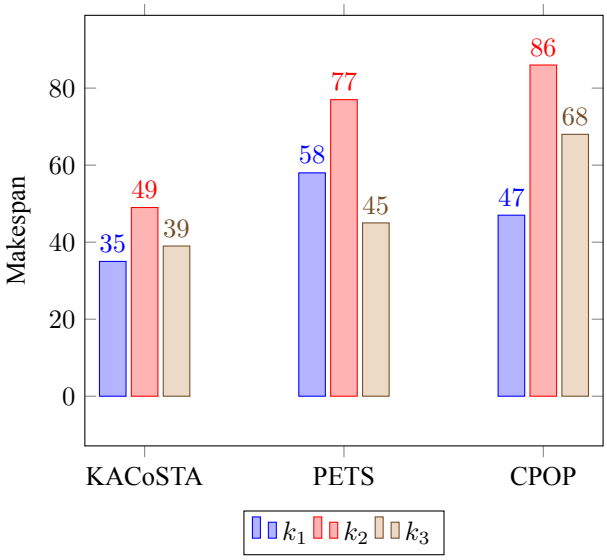

Figure 12: Comparison of elapsed time on each processor k_1, k_2 and k_3 for KaCoSTA, PETS and CPOP

CPOP do not account for independent task groups and employ a task prioritization phase. This phase determines the priority of each task based on rank metrics, and is calculated through parental prioritization, as described in [30].

4.2 Benchmark datasets experience

Because, we adopt a dynamic programming technique, which optimally balances the load across processors, KaCoSTA consistently achieves lower makespan compared to existing algorithms across all datasets [2]. For example, in Dataset A, KaCoSTA achieves a makespan of 105, significantly lower than Min-Min's 140 and PHTS's 110. Thus KaCoSTA is the most efficient algorithm in task allocation and execution time minimization.

Regarding RU, KaCoSTA shows the highest level of efficiency, with an average of 85% in Dataset A, compared to 65% for Min-Min and 80% for PHTS. This result indicates that KaCoSTA effectively leverages available processing power, reducing idle times and enhancing overall system performance.

In addition, KaCoSTA performs particularly well in scenarios with high task heterogeneity and varying processor capabilities. In Dataset B, with 500 tasks and 10 processors, KaCoSTA's makespan is 600, compared to 700 for Min-Min and 620 for PHTS. This highlights KaCoSTA's adaptability and efficiency in complex environments.

Moreover, lower variance and narrower confidence intervals for KaCoSTA (e.g., 6.5 variance and [103, 107] Confidence Interval (CI) in Dataset A) indicate more consistent performance and reliability compared to other algorithms. This robustness is crucial for applications requiring predictable and stable task scheduling outcomes.

Thus, KaCoSTA demonstrates clear advantages over existing algorithms in terms of makespan reduction, RU, and consistent performance across various datasets.

4.3 Discussion

KaCoSTA performs particularly well in scenarios with high task heterogeneity and varying processor capabilities. For example, in Dataset C, with 1000 tasks and 15 processors, KaCoSTA achieves a makespan of 1250, compared to 1500 for Min-Min and 1350 for PETS. This highlights the algorithm's robustness and efficiency in complex and dynamic environments.

The lower variance and narrower confidence intervals for KaCoSTA, such as a variance of 6.5 and a 95% confidence interval of [103, 107] in Dataset A, indicate more consistent performance and reliability compared to other algorithms. This is crucial for applications requiring predictable and stable task scheduling outcomes.

Algorithm KaCoSTA significantly improves upon existing task scheduling algorithms (see Table 2) in terms of makespan and RU. More precisely, KaCoSTA outperforms Min-Min, Max-Min, and QoS-guided Min-Min to achieve a low makespan, which is due to their unsuitability for long tasks, less efficiency in a dynamic environment, and limited scalability, respectively. In addition, despite the improvement in terms of scalability of the SOTA algorithms considered in this work, our algorithm is still the best in terms of makespan and RU. This is due to KaCoSTA's complexity, which is still the best.

Moreover, the observed makespan's decreasing and the RU's increasing are primarily due to KaCoSTA's dynamic allocation of tasks, which is based on its real-time processing ability. The experimental results of our approach demonstrate significant improvements across all key metrics. We noticed the efficiency of dynamic programming combined with the knapsack techniques, which enables

[2] Experimented benchmark datasets are taken from http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html

Dataset	algorithm	Makespan	Variance	95% CI	RU	Variance	95% CI U
Dataset A	Min-Min	140	15.2	[135, 145]	65%	2.5	[63%, 67%]
	Max-Min	130	12.8	[126, 134]	70%	3.1	[68%, 72%]
	QoS Min-Min	125	10.4	[121, 129]	72%	2.8	[70%, 74%]
	HEFT	115	8.7	[112, 118]	78%	1.9	[77%, 79%]
	PHTS	110	7.5	[107, 113]	80%	1.7	[79%, 81%]
	PETS	120	9.6	[117, 123]	75%	2.4	[73%, 77%]
	KaCoSTA	**105**	**6.5**	**[103, 107]**	**85%**	**1.2**	**[84%, 86%]**
Dataset B	Min-Min	700	45.6	[688, 712]	60%	3.9	[58%, 62%]
	Max-Min	680	40.8	[669, 691]	65%	4.1	[63%, 67%]
	QoS Min-Min	670	38.2	[659, 681]	67%	3.7	[65%, 69%]
	HEFT	640	35.4	[630, 650]	72%	2.9	[71%, 73%]
	PHTS	620	30.6	[611, 629]	75%	2.2	[74%, 76%]
	PETS	650	36.8	[640, 660]	70%	3.4	[68%, 72%]
	KaCoSTA	**600**	**25.2**	**[593, 607]**	**80%**	**1.8**	**[79%, 81%]**
Dataset C	Min-Min	1500	120.8	[1475, 1525]	55%	5.6	[53%, 57%]
	Max-Min	1450	110.4	[1427, 1473]	60%	4.8	[58%, 62%]
	QoS Min-Min	1400	105.2	[1378, 1422]	62%	4.5	[60%, 64%]
	HEFT	1350	98.6	[1330, 1370]	68%	3.8	[67%, 69%]
	PHTS	1300	85.4	[1283, 1317]	70%	3.1	[69%, 71%]
	PETS	1350	90.2	[1333, 1367]	66%	3.6	[65%, 67%]
	KaCoSTA	**1250**	**70.5**	**[1235, 1265]**	**75%**	**2.9**	**[74%, 76%]**

Table 7: Statistical Analysis of Algorithm Performance

KaCoSTA to handle larger task sets with better scalability and quadratic complexity.

Our approach has proven its efficiency in terms of makespan, RU. The reason is that our approach succeeds to address the issues of the tested SOTA methods (see Table 8).

The novelty of our approach lies in the new modeling and solving techniques that allow optimal results in terms of makespan and RU. More accurately, KaCoSTA models a task scheduling problem as a knapsack problem, which allows a structured and efficient task allocation. In addition, a DP technique is utilized to systematically solve the scheduling problem, breaking it down into manageable subproblems. This ensures optimal task allocation and load balancing across processors. MakespanThreshold Estimation is used to guide the scheduling process and to balance the trade-off between minimizing makespan and maximizing RU.

Thus, our algorithm is highly adaptable to various HCS scenarios, effectively handling different task sets and processor configurations. KaCoSTA demonstrates consistent performance with lower variance and narrower confidence intervals, ensuring reliable scheduling outcomes. This robustness is crucial for applications requiring stable and predictable task execution times.

5 Conclusion and future works

In this paper, we first introduced a task scheduling algorithm for Heterogeneous Computing Systems (HCS) called KReSTA, a recursive method. We then presented a second algorithm named KISTA, an improved version of KReSTA that offers better computing performance overall, particularly in terms of time complexity. However, in some cases, KReSTA performs better, especially with relatively small task sets. Specifically, when the number of tasks is $n \lesssim 35$, KReSTA is preferable as it provides better performance. This observation leads us to plan future work on developing an enhanced version that combines the strengths of both algorithms.

Finally, we proposed a procedure that encapsulates our approach for applying the entire method to an HCS, defined in Algorithm 3 and named KaCoSTA. The heuristic introduced in this paper is primarily based on a mathematical model for solving the knapsack and optimization problems. The first phase involves estimating MT, analogous to the knapsack capacity, which must not be exceeded. The second phase involves considering all tasks as objects to be placed in processors' queues without surpassing the calculated MT. The final stage optimizes the objective function concerning the execution cost and priority of each task, followed by scheduling the assigned tasks according to their priority.

The described process of our approach explains why it achieves the minimum computation time. As a result, KaCoSTA provides more efficient task scheduling compared to other algorithms. We compared the performance of KaCoSTA with algorithms like PHTS, HEFT, PETS, CPOP, and SLNDD. The comparison metrics were based on examples from related work, which considered heterogeneous systems with multiple processors and various task sets. KaCoSTA yielded enhanced results in terms of makespan and processing performance compared to PHTS and PETS,

Algorithm	Handicap
Min-Min	Performs poorly in RU due to its tendency to leave some processors underutilized. This results in a higher makespan in heterogeneous environments.
Max-Min	Balances the load better than Min-Min but still lacks efficiency in dynamic environments where task execution times vary significantly.
QoS Min-Min	Incorporates QoS constraints but sacrifices scalability and efficiency.
HEFT	Achieves better makespan and RU but suffers from complexity in implementation, making it less adaptable to varying scenarios.
PHTS	Performs well in terms of makespan and RU but has high complexity and less adaptability, limiting its effectiveness.
PETS	Effective for preemptable tasks but inefficient for non-preemptive tasks, leading to a higher makespan and lower RU.

Table 8: SoTA algorithms handicap

and it outperformed algorithms like QoS Guided Min-Min, HEFT, and SLNDD in other measures.

KaCoSTA demonstrates clear advantages over state-of-the-art algorithms in terms of makespan reduction, RU, and consistent performance across various datasets and scenarios. The utilization of a knapsack problem-based model, coupled with dynamic programming, offers significant improvements in task scheduling for heterogeneous computing systems. This research highlights the potential of KaCoSTA to enhance system performance and resource management in complex and dynamic environments.

By addressing the limitations of existing algorithms and introducing innovative techniques, our heuristic has advanced the field of task planning in heterogeneous IT systems, providing a more efficient and adaptable solution. Our future work will focus on strengthening the KaCoSTA algorithm by exploring the application of Artificial Intelligence (AI), which will improve the estimation of MT and further optimize task allocation behavior.

References

[1] M. Gallet, L. Marchal, F. Vivien (2009), Efficient scheduling of task graph collections on heterogeneous resources. IEEE International Symposium on Parallel & Distributed Processing, IEEE, 2009, pp. 1–11. http://dx.doi.org/10.1109/ipdps.2009.5161045

[2] X.He, X.Sun, G.Von Laszewski (2003), Qos guided min-min heuristic for grid task scheduling. Journal of computer science and technology 18 (4) (2003) pp 442–451. http://dx.doi.org/10.1007/bf02948918

[3] S. K. Panda, P. K. Jana (2016), Uncertainty-based qos min–min algorithm for heterogeneous multi-cloud environment, Arabian Journal for Science and Engineering 41 (8) (2016) pp. 3003–3025. http://dx.doi.org/10.1007/s13369-016-2069-7

[4] P.Ezzatti, M.Pedemonte, A. Martın (2013), An efficient implementation of the min-min heuristic, Computers & operations research 40 (11) (2013) pp. 2670–2676. http://dx.doi.org/10.1016/j.cor.2013.05.014

[5] K. Etminani, M. Naghibzadeh (2007), A min-min max-min selective algorihtm for grid task scheduling, in: 2007 3rd IEEE/IFIP International Conference in Central Asia on Internet, IEEE, 2007, pp. 1–7. http://dx.doi.org/10.1109/canet.2007.4401694

[6] Mohammad, Omer K Jasim and Salih, Bassim M (2024). Improving Task Scheduling In Cloud Datacenters By Implementation Of An Intelligent Scheduling Algorithm. Informatica Journal. 48 (10), 2024 http://dx.doi.org/10.31449/inf.v48i10.5843

[7] T.Stutzle, H.H.Hoos (2000), Max–min ant system, Future generation computer systems 16 (8) (2000) pp. 889–914. http://dx.doi.org/10.1016/s0167-739x(00)00043-1

[8] H.Topcuoglu, S.Hariri,M.-Y.Wu (1999), Task scheduling algorithms for heterogeneous processors, in: Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99), IEEE, 1999, pp. 3– 14. http://dx.doi.org/10.1109/hcw.1999.765092

[9] H. Topcuoglu, S. Hariri, M.-Y. Wu (2002), Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE transactions on parallel and distributed systems 13 (3) (2002) pp. 260–274. http://dx.doi.org/10.1109/71.993206

[10] Xinyu, Zhang and JinJian, Liu and Guanwei, Zhang and Wei, Guo (2024), Optimization of Asynchronous Parallel Tasks Scheduling with Multi-resource Constraints, journal Informatica, (2024) 48 (7) http://dx.doi.org/10.31449/inf.v48i7.5604

[11] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, Z. Gu (2012), Online optimization for scheduling preemptable tasks on iaas cloud systems, Journal of Parallel and Distributed Computing 72 (5) (2012) pp. 666–677. http://dx.doi.org/10.1016/j.jpdc.2012.02.002

[12] S. Parsa, R. Entezari-Maleki, Rasa (2009), A new grid task scheduling algorithm, International Journal of Digital Content Technology

and its Applications 3 (4) (2009) pp. 91–99. http://dx.doi.org/10.4156/jdcta.vol3.issue4.10

[13] M. Raeisi-Varzaneh, O. Dakkak, A. Habbal, B.-S. Kim (2023), Resource scheduling in edge computing: Architecture, taxonomy, open issues and future research directions, IEEE Access 11 (2023) pp. 25329–25350. http://dx.doi.org/10.1109/access.2023.3256522

[14] L. F.Bittencourt, E.R.Madeira (2010), Towards the scheduling of multiple workflows on computational grids, Journal of grid computing 8 (2010) pp. 419–441. http://dx.doi.org/10.1007/s10723-009-9144-1

[15] M. A. Rodriguez, R. Buyya (2015), A responsive knapsack-based algorithm for resource provisioning and scheduling of scientific workflows in clouds, 44th International Conference on Parallel Processing, IEEE, 2015, pp. 839–848. http://dx.doi.org/10.1109/icpp.2015.93

[16] K.Al-Saqabi, S.Sarwar, K.Saleh (1997), Distributed gang scheduling in networks of heterogenous workstations, Computer communications 20 (5) (1997) pp. 338–348. http://dx.doi.org/10.1016/s0140-3664(97)00020-0

[17] G. C. Sih, E. A. Lee (1993), A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures, IEEE transactions on Parallel and Distributed systems 4 (2) (1993) pp. 175–187. http://dx.doi.org/10.1109/71.207593

[18] M.Pedemonte, P.Ezzatti, A. Martın (2016), Accelerating the min-min heuristic, Parallel Processing and Applied Mathematics: 11th International Conference, PPAM 2015, Krakow, Poland, September 6-9, 2015. Revised Selected Papers, Part II, Springer, 2016, pp. 101–110. http://dx.doi.org/10.1007/978-3-319-32149-3_10

[19] D. P. Vidyarthi, A. K. Tripathi (2001), Maximizing reliability of distributed computing system with task allocation using simple genetic algorithm, Journal of Systems Architecture 47 (6) (2001) pp. 549–554. http://dx.doi.org/10.1016/s1383-7621(01)00013-3

[20] I. Gupta, M. S. Kumar, P. K. Jana (2018), Efficient workflow scheduling algorithm for cloud computing system: a dynamic priority-based approach, Arabian Journal for Science and Engineering 43 (12) (2018) pp. 7945–7960. http://dx.doi.org/10.1007/s13369-018-3261-8

[21] S.Martello, P.Toth (1987), Algorithms for knapsack problems, North- Holland Mathematics Studies 132 (1987) pp. 213–257. http://dx.doi.org/10.1016/s0304-0208(08)73237-7

[22] C. Wu, Y. Wang, A. Zhao, T. Qiu (2013), Research on task allocation strategy and scheduling algorithm of multi-core load balance, in: 2013 Seventh International Conference on Complex, Intelligent, and Software Intensive Systems, IEEE, 2013 pp. 634–638. http://dx.doi.org/10.1109/cisis.2013.114

[23] D. Zouache, A. Moussaoui, F. B. Abdelaziz (2018), A cooperative swarm intelligence algorithm for multi-objective discrete optimization with application to the knapsack problem, European Journal of Operational Research 264 (1) (2018) pp. 74–88. http://dx.doi.org/10.1016/j.ejor.2017.06.058

[24] N. Galimyanova (2008), Experimental investigations of combined algorithms of branch and bound method and dynamic program- ming method for knapsack problems, Journal of Computer and Systems Sciences International 47 (3) (2008) pp. 422–428. http://dx.doi.org/10.1134/s106423070803012x

[25] R.M.Sahoo, S.K.Padhy (2022), A novel algorithm for priority-based task scheduling on a multiprocessor heterogeneous system, Microprocessors and Microsystems 95 (2022) 104685. http://dx.doi.org/10.1016/j.micpro.2022.104685

[26] D. M. Abdelkader, F. Omara (2012), Dynamic task scheduling algorithm with load balancing for heterogeneous computing system, Egyptian Informatics Journal 13 (2) (2012) pp. 135–145. http://dx.doi.org/10.1016/j.eij.2012.04.001

[27] T.D.Braun, H.Siegal, N.Beck, L.L.Boloni, M.Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, et al. (1999), A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems, in: Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99), IEEE, 1999, pp. 15–29. http://dx.doi.org/10.1109/hcw.1999.765093

[28] R. Eswari, S. Nickolas, M. Arock (2014), A path priority-based task scheduling algorithm for heterogeneous distributed systems, International Journal of Communication Networks and Distributed Systems 12 (2) (2014) pp. 183–201. http://dx.doi.org/10.1504/ijcnds.2014.059437

[29] E. Ilavarasan, P. Thambidurai, R. Mahilmannan (2005), Performance effective task scheduling algorithm for heterogeneous computing system, in: The 4th international symposium on parallel and distributed computing (ISPDC'05), IEEE, 2005, pp. 28–38. http://dx.doi.org/10.1109/ispdc.2005.39

[30] M. S. Arif, Z. Iqbal, R. Tariq, F. Aadil, M. Awais (2019), Parental prioritization-based task scheduling in heterogeneous systems, Arabian Journal for Science and Engineering 44 (4) (2019) pp. 3943–3952. http://dx.doi.org/10.1007/s13369-018-03698-2