World Academy of Science, Engineering and Technology
International Journal of Cognitive and Language Sciences
Vol:18, No:11, 2024

Resource-Constrained Heterogeneous Workflow Scheduling Algorithm for Heterogeneous Computing Clusters

Lei Wang, Jiahao Zhou

Abstract—The development of heterogeneous computing clusters provides robust computational support for large-scale workflows, commonly seen in domains such as scientific computing and artificial intelligence. However, the tasks within these large-scale workflows are increasingly heterogeneous, exhibiting varying demands on computing resources. This shift necessitates the integration of resource-constrained considerations into the workflow scheduling problem on heterogeneous computing platforms. In this study, we propose a scheduling algorithm designed to minimize the makespan under heterogeneous constraints, employing a greedy strategy to effectively address the scheduling challenges posed by heterogeneous workflows. We evaluate the performance of the proposed algorithm using randomly generated heterogeneous workflows and a corresponding heterogeneous computing platform. The experimental results demonstrate a 15.2% improvement in performance compared to existing state-of-the-art methods.

Keywords—Heterogeneous Computing, Workflow Scheduling, Constrained Resources, Minimal Makespan.

I. Introduction

ACCELERATORS are playing an increasingly critical role in high-performance computing, driven by the rising power consumption of central processing units (CPUs) and the relatively higher energy efficiency offered by accelerators. Modern supercomputing clusters have long incorporated heterogeneity within their computing nodes, with an increasing number of applications offloading computational tasks to accelerators to improve performance [8], [9]. To support diverse application requirements, contemporary computing cluster nodes adopt a CPU-plus-accelerator (XPU) architecture, consisting of multiple CPUs and several accelerators within each node [10]. Fig. 1 illustrates a typical heterogeneous computing node, where four accelerators represent distinct device types (e.g., GPUs, TPUs), each with varying computational capabilities. Due to the differing communication protocols used by accelerators from different manufacturers, direct communication between devices is not feasible. Instead, data must be transferred to host memory via PCIe [5], which then relays the data to the respective devices. Additionally, the bandwidth of data transmission between the host and each device varies.

Moreover, applications themselves are increasingly heterogeneous in nature [6]. Complex workflows integrating data analysis, simulation, and artificial intelligence (AI) methodologies are reshaping the use of supercomputers. For instance, data processing tasks are best suited for highly parallel GPUs, while TPUs, optimized for tensor computations, are more appropriate for AI workloads. Fig. 2 provides an example of a heterogeneous workflow composed of multiple dependent tasks, each characterized by distinct workloads and corresponding accelerators optimized for their execution.

The scheduling of heterogeneous workflows on heterogeneous computing platforms poses significant challenges, primarily due to the complex task dependencies and the vast combinatorial space involving task-to-accelerator mappings. The Heterogeneous Constrained Minimum Makespan Scheduling algorithm ($HCMMS$) proposed in this paper effectively addresses these challenges, offering a solution that minimizes the overall workflow completion time. The main contributions of this work are summarized as follows:

- We propose $HCMMS$, a scheduling algorithm based on a greedy policy, to tackle the problem of scheduling heterogeneous workflows on heterogeneous computing platforms.
- A data prefetching mechanism is introduced to enhance communication between tasks, increasing the overlap between communication and computation, which leads to a significant reduction in workflow makespan.
- The static heuristic approach, $HCMMS$, demonstrates scalability and can accommodate realistic, large-scale workflow application scenarios. Experimental results show that $HCMMS$ outperforms existing state-of-the-art approaches in terms of makespan reduction.

The subsequent sections of this paper are structured as follows: Chapter II offers a comprehensive overview of related literature. Chapter III delineates the problem model, while Chapter IV presents our proposed solution. Chapter V discusses the experimental results obtained. Finally, Chapter VI provides a summary of the entire paper.

Lei Wang is with the Research Institute of Electronic Science and Technology, University of Electronic Science and Technology of China, Chengdu, Sichuan, 611731 CN (corresponding author, e-mail: wang_lei@uestc.edu.cn).

Jiahao Zhou is with the Research Institute of Electronic Science and Technology, University of Electronic Science and Technology of China, Chengdu, Sichuan, 611731 CN (e-mail: 202221230139@std.uestc.edu.cn).

Research supported by Fundamental Research Funds for the Central Universities and the Sichuan Science and Technology Program.

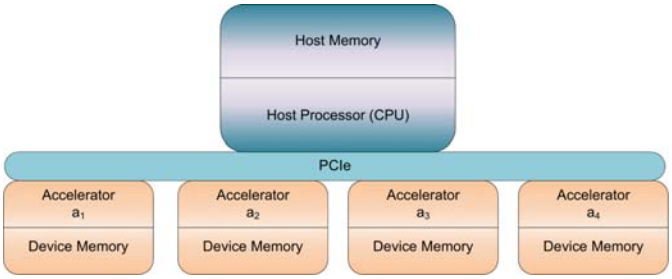


Fig. 1 The architecture of heterogeneous computing platform

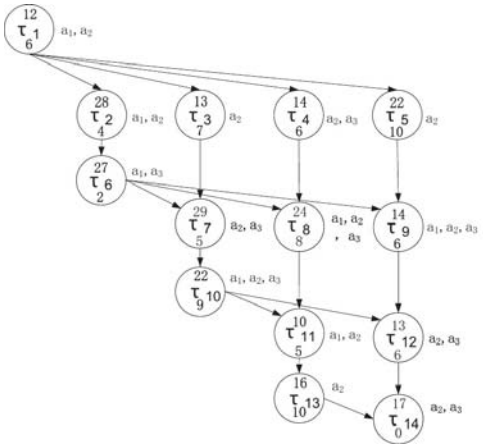


Fig. 2 An example of heterogeneous workflow

II. Related Works

List-based heuristic scheduling methods [1]–[3], [11]–[13] have been extensively studied and applied to task scheduling problems in heterogeneous computing environments. These methods typically follow a common approach: tasks are first prioritized to respect precedence constraints, after which computational resources are allocated sequentially to the prioritized tasks. A representative example is the Heterogeneous Earliest Finish Time ($HEFT$) algorithm [1], proposed by Topcuoglu et al. in 2002. $HEFT$ assigns an upward rank to each task during the initial phase, where this rank reflects the estimated critical path length from the current task to the terminal task. In the subsequent phase, the algorithm selects the computational resource for the highest-priority task that minimizes the overall completion time. The Predict Earliest Finish Time ($PEFT$) algorithm [2], introduced by Arabnejad et al. in 2013, refines task prioritization by calculating an optimal cost value for each task-processor pair, leading to a performance improvement over HEFT. In 2018, He et al. proposed the Task Duplication based Clustering Algorithm ($TDCA$) [3], which enhances scheduling efficiency by optimizing parameter calculation, task replication, and task merging strategies. The method introduced in this paper also falls within the category of list-based heuristic scheduling. However, it incorporates key enhancements in task priority determination and resource selection strategies, with the goal of optimizing workflow makespan.

III. Problem Model

A. Problem Background

In this chapter, we present the problem formulation, with the mathematical notation summarized in Table I. We consider a complex workflow comprising multiple dependent tasks, modeled as a dataflow task graph, to be scheduled on a fully connected heterogeneous computing platform. Each task within the workflow is restricted to execution on specific accelerators, owing to its code characteristics and computational requirements.

B. Mathematical Formulation

Fig. 2 illustrates a dataflow task graph represented as a directed acyclic graph (DAG) $G(T, D)$, where $T = \{\tau_1, \tau_2, ..., \tau_{|T|}\}$ denotes the set of task nodes in the workflow, each characterized by $(size, output, devices)$. The set $D = \{d_{j,k} | \tau_j, \tau_k \in T\}$ denotes the data dependencies between tasks, where the execution of task τ_k depends on the data produced by task τ_j. In Fig. 2, the number above each task node τ_i represents the task's workload ($size$), while the number below corresponds to the amount of output data ($output$). The set on the right side of each task node specifies the accelerators capable of executing the task. We assume the task graph contains a single source node τ_{entry} and a single sink node τ_{exit}. For task graphs with multiple source or sink nodes, this assumption can be met by introducing a pseudo-source (or pseudo-sink) node with zero $size$ and zero $output$, connected as a predecessor (or successor) to all actual source (or sink) nodes.

Fig. 1 illustrates a heterogeneous computing platform denoted as $P = \{a_1, a_2, ..., a_{|P|}\}$, where each accelerator a_m is characterized by its processing speed (s) and bandwidth (b). Data communication between different types of accelerators is routed through the host. The legal start time for task τ_j on accelerator a_m is denoted as

$$start_{\tau_j}^{a_m} = \max\left[\max_{\tau_i \in \zeta(\tau_j)}(receive_{\tau_i}), avail[a_m]\right] \quad (1)$$

where

$$receive_{\tau_i} = \begin{cases} end_{\tau_i}, & \text{if } m = n \\ end_{\tau_i} + \frac{output_{\tau_i}}{b_m} + \frac{output_{\tau_i}}{b_n}, & \text{otherwise} \end{cases} \quad (2)$$

TABLE I
Notations

Symbol	Definition
$size$	Workload of task
$output$	Output data size of task
$devices$	The set of devices on which this task can run
s	Computing speed of device
b	Bidirectional communication bandwidth with the host
$\zeta(\tau_j)$	All predecessors of task node τ_j
$\xi(\tau_j)$	All successors of task node τ_j
$avail[a_m]$	Available time of the device a_m
$start_{\tau_j}$	Start time of task τ_j
end_{τ_j}	End time of task τ_j
$exec_{\tau_j}^{a_m}$	Execution time of task τ_j on a_m

while the execution time of task τ_j on a_m is denoted as

$$exec_{\tau_j}^{a_m} = \frac{size_{\tau_j}}{s_m} \tag{3}$$

As shown in (2), the choice of accelerator for a given task impacts not only its computation time (ct) but also the data transfer time (dtt) from its parent tasks. This becomes a critical factor when selecting an appropriate accelerator.

It is important to note that a prefetching mechanism is employed to reduce data transfer time. Specifically, when a parent task τ_i transfers its output data to the host, the child task τ_j can begin reading these data immediately, without waiting for all parent tasks to complete their transfers. To further optimize data transfer, a first-in-first-out (FIFO) reading mechanism is used, allowing τ_j to access the data as soon as it is available on the host. This strategy minimizes the data transfer time and thus contributes to reducing the overall completion time of the entire workflow, as represented by

$$makespan = end_{\tau_{exit}} - start_{\tau_{entry}}. \tag{4}$$

C. Optimization Object

A valid scheduling solution must satisfy the following constraints:

- Resource constraint, each accelerator can only execute one task at the same moment, i.e., any two tasks τ_i and τ_j executed on the same accelerator should satisfy $[start_{\tau_i}, end_{\tau_i}] \cap [start_{\tau_j}, end_{\tau_j}] = \emptyset$.
- Inter-task dependency constraints, i.e., satisfying $start_{\tau_k} \geq end_{\tau_j}$ for any $d_{j,k} \in D$.

Based on the aforementioned system model, we formally define the problem as follows: given a workflow G and a fully connected heterogeneous computing platform P, the objective is to find a feasible schedule that minimizes the makespan.

IV. The Proposed Method

A. Task Sorting

Our proposed method adheres to the constraints of legal scheduling by computing a matrix of dimensions $|T| \times |P|$, where each element $LST[\tau_j, a_m]$, for $1 \leq j \leq |T|$ and $1 \leq m \leq |P|$, represents the legal start time of task τ_j on accelerator a_m. A greedy strategy is then employed to assign the most suitable accelerator to each task based on the values in the LST matrix, yielding a near-optimal legal schedule. Prior to calculating the LST matrix, the following preliminary steps are necessary: (1) compute the remaining estimated workload $REW[\tau_j, a_m]$ for each task-accelerator pair, which serves as the basis for task prioritization and resource allocation; and (2) calculate the priority $Rank[\tau_j]$ for each task, followed by ordering the tasks in non-increasing order based on their $Rank$ values.

The computation of $REW[\tau_j, a_m]$ and $Rank[\tau_j]$ is detailed in Algorithm 1. It is important to note that the data transfer time between each task τ_j and its corresponding successor τ_k is denoted as

$$dtt_{j,k}^{m,n} = \begin{cases} 0, & \text{if } m = n \\ \frac{output_{\tau_j}}{b_m} + \frac{output_{\tau_j}}{b_n}, & \text{otherwise} \end{cases}. \tag{5}$$

Algorithm 1 $REW\_Rank(G, P)$

1: Initialize a queue Q with exit node
2: **while** Q is not empty **do**
3: Dequeue a task node τ_j from Q
4: **for** Each legal device a_m in $devices$ of τ_j **do**
5: **if** $\tau_j = \tau_{exit}$ **then**
6: $REW[\tau_j, a_m] = ct_{j,m}$
7: **else**
8: $REW[\tau_j, a_m] = \max_{\tau_k \in \xi(\tau_j)}[\min_{a_n \in devices(\tau_k)} (REW[\tau_k, a_n] + ct_{j,m} + dtt_{j,k}^{m,n})]$
9: **end if**
10: **end for**
11: $Rank[\tau_j] = \sum_{a_m \in devices} \frac{REW[\tau_j, a_m]}{|devices|}$
12: $maxSuccRank = \max_{\tau_k \in \xi(\tau_j)} Rank[\tau_k]$
13: **if** $maxSuccRank \geq Rank[\tau_j]$ **then**
14: **for** Each legal device a_m in $devices$ of τ_j **do**
15: $REW[\tau_j, a_m] = REW[\tau_j, a_m] \times \frac{maxSuccRank}{Rank[\tau_j]} \times \alpha$
16: **end for**
17: $Rank[\tau_j] = maxSuccRank \times \alpha$
18: **end if**
19: **for** τ_i in $\zeta(\tau_j)$ **do**
20: Enqueue τ_i in Q if τ_i's all immediate successors have been processed
21: **end for**
22: **end while**
23: **return** $REW, Rank$

In step 15, the parameter α is a hyperparameter slightly greater than 1; for our evaluations, we utilized a value of 1.01.

B. Accelerator Allocation

We compute LST sequentially for the tasks in sorted order and assign each task to the accelerator that minimizes the sum of LST and REW. Algorithm 2 provides a detailed description of the iterative computations for both LST and the allocation process. Here, $AST[\tau_j]$ represents the actual start time of task τ_j. As a result, we obtain the specific accelerator assigned to each task along with their respective start execution times, thus yielding a legal schedule.

C. Complexity Analysis

The algorithmic complexity of $HCMMS$ consists of two primary components: Algorithm 1 and Algorithm 2. In Algorithm 1, the computation of REW processes each edge in the task graph exactly once and iterates over no more than $|P|$ accelerators to identify the minimum. Consequently, the overall complexity of the REW computation does not exceed $O(|D| \times |P|)$. The complexity associated with computing the priority $Rank$ for each task is also bounded by $O(|P|)$, resulting in an overall complexity for $Rank$ computation that does not exceed $O(|T| \times |P|)$. Assuming that $|D|$ is greater than $|T|$, the computational complexity of Algorithm 1 is therefore capped at $O(|D| \times |P|)$.

In Algorithm 2, the initial complexity of sorting all tasks is $O(|T| \log |T|)$. The subsequent computational complexity

Algorithm 2 Accelerator Allocation

1: $REW, Rank = REW\_Rank(G, P)$
2: Sorting tasks based on non-increasing Rank values
3: **for** τ_j in sorted tasks **do**
4: **if** $\tau_j = \tau_{entry}$ **then**
5: $LST[\tau_j] = 0$
6: **else**
7: **for** Each legal device a_m in $devices$ of τ_j **do**
8: $LST[\tau_j, a_m] = \max[avail[a_m], \max_{\tau_i \in \zeta(\tau_j)}(receive_{\tau_i})]$
9: **end for**
10: **end if**
11: $alloc[\tau_j] = \arg\min_{a_m}(LST[\tau_j] + REW[\tau_j])$
12: $AST[\tau_j] = LST[\tau_j, alloc[\tau_j]]$
13: $avail\,[alloc[\tau_j]] = AST[\tau_j] + \frac{size_{\tau_j}}{s_{alloc[\tau_j]}}$
14: **end for**

is primarily determined by the computation of LST for each task-accelerator pair, as the calculations for $alloc$, AST, and $avail$ are performed in constant time. As indicated in step 8, the overhead for computing LST for each task-accelerator pair is $O(|\zeta(\tau_j)|)$, which leads to an overall overhead for LST across all task-accelerator pairs for a single accelerator of $O(|D|)$. Therefore, the total computational complexity for LST does not exceed $O(|D| \times |P|)$, given that the number of available devices per task is limited to $|P|$.

Thus, the computational complexity of Algorithm 2 can be expressed as $O(|T| \log |T| + |D| \times |P|)$, which simplifies to $O(|D| \times |P|)$. Overall, the algorithmic complexity of $HCMMS$ is therefore $O(|D| \times |P|)$.

V. Evaluation

A. Comparison with $HEFT$ and $PEFT$

To evaluate the effectiveness of our proposed scheduling algorithm, we randomly generated a series of heterogeneous workflows and a computing platform consisting of three heterogeneous accelerators. Initially, we compared the performance of $HCMMS$ with that of $PEFT$ using the example illustrated in Fig. 2. The Gantt chart depicting the scheduling results is presented in Fig. 3. This comparison demonstrates that $HCMMS$ significantly outperforms $PEFT$ in terms of makespan.

Considering the task graph of this workflow, upon the completion of task τ_1, tasks τ_2, τ_3, τ_4, and τ_5 simultaneously meet their dependency conditions, with three accelerators currently available. As shown in Fig. 2, the direct and indirect successors of τ_5 are $[\tau_9, \tau_{12}, \tau_{14}]$, which represent the tasks with the fewest successors among those that can currently be executed. Consequently, our scheduler assigns the three available accelerators to tasks τ_2, τ_3, and τ_4 to minimize the overall completion time of the workflow.

We conducted a comparative analysis of $HCMMS$, $HEFT$, and $PEFT$ across 200 randomly generated heterogeneous workflows. In these experiments, the number of tasks was uniformly distributed between 20 and 1000, while the task sizes followed a normal distribution with a mean ranging from 5 to 50. Similarly, the output values of the tasks were normally distributed with a mean between 2 and 20. The available accelerators for each task were generated randomly. The results of our experiments are summarized in Table II, which clearly demonstrates that $HCMMS$ consistently outperforms both $HEFT$ and $PEFT$ in terms of makespan.

TABLE II
Pair-Wise Makespan Comparison of the Scheduling Algorithms

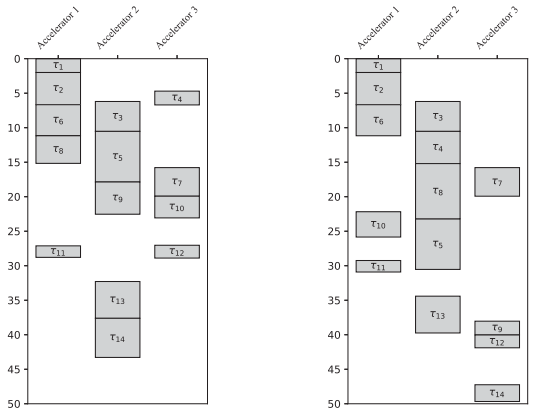
		$HEFT$	$PEFT$
$HCMMS$	better	145	102
	equal	12	27
	worse	43	71

B. Scalability of the Proposed Method

We evaluated the scalability of our proposed approach within a randomly generated cluster comprising 20 heterogeneous accelerators. Fig. 4 illustrates the completion times for workflows containing between 100 and 1000 tasks on this heterogeneous computing cluster. The results indicate that the completion times for the workflows increase steadily with the number of tasks, without exhibiting a sharp rise in completion times for larger workflows. This demonstrates that the proposed method exhibits robust scalability across workflows of varying sizes.

C. Ablation Experiment

As discussed in Chapter I, we incorporated a data prefetching mechanism to reduce the waiting time for data transfer among tasks. In this subsection, we design ablation experiments to evaluate the effectiveness of this measure. In the control group, we removed the data prefetching mechanism, allowing each task to commence data reading only



(a) $HCMMS$ (b) $PEFT$

Fig. 3 The Gantt charts illustrating the schedules for the given DAG in Fig. 2: (a) the schedule produced by the $HCMMS$, with a $makespan$ of 43; (b) the schedule produced by the $PEFT$, with a $makespan$ of 49

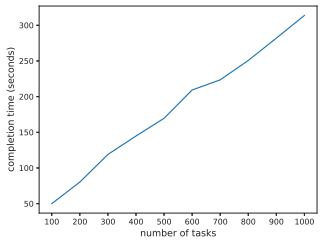
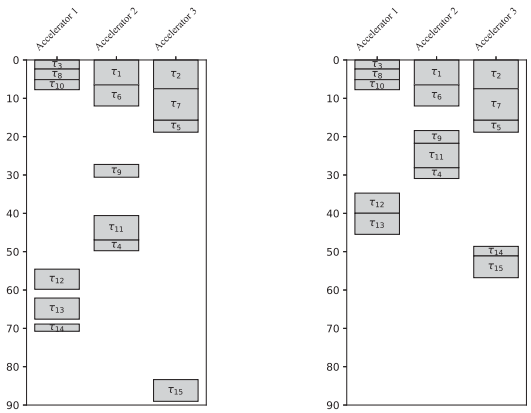


Fig. 4 Completion times for workflows of diverse sizes



(a) $HCMMS$ without data prefetching mechanism (b) complete $HCMMS$

Fig. 5 The Gantt charts illustrating the schedules as follows: (a) the schedule produced by the $HCMMS$ without data prefetching mechanism, with a $makespan$ of 89; (b) the schedule produced by the complete $HCMMS$, with a $makespan$ of 57

after all its preceding tasks have transferred their data to the host.

We conducted tests using a workflow case comprising 15 tasks and 34 data dependencies, executed on a heterogeneous computing platform featuring three accelerators with varying computational speeds and communication bandwidths. The Gantt chart presented in Fig. 5 illustrates the scheduling outcomes. It is evident that the elimination of the data prefetching mechanism not only delays the actual start times of the tasks but also impacts the processors allocated to each task, resulting in an additional 56% increase in the makespan of the workflow.

VI. Conclusion and Future Work

Heterogeneous computing resources offer effective hardware support for accelerating workflows; however, the successful offloading of heterogeneous tasks to suitable acceleration devices is essential for maximizing resource utilization and minimizing workflow completion times. In this paper, we present $HCMMS$, a scheduling algorithm based on a greedy policy that delivers an efficient scheduling solution for heterogeneous workflows on heterogeneous computing platforms. Nevertheless, scheduling individual workflows on such platforms continues to pose challenges, including low resource utilization and uneven resource loading. In our future work, we will address the problem of scheduling multiple workflows on heterogeneous platforms to ensure optimal utilization of hardware resources.

References

[1] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Trans. Parallel Distrib. Syst., vol. 13, no. 3, pp. 260–274, Mar. 2002.
[2] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 3, pp. 682–694, Mar. 2014.
[3] K. He, X. Meng, Z. Pan, L. Yuan and P. Zhou, "A Novel Task-Duplication Based Clustering Algorithm for Heterogeneous Computing Environments," in IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 1, pp. 2-14, 1 Jan. 2019.
[4] Paraskevas, Kyriakos. "Enabling Direct-Access Global Shared Memory for Distributed Heterogeneous Computing." (2023).
[5] Lawley, Jason. "Understanding Performance of PCI Express Systems." WP350 (v1. 2). **linx 97 2014.
[6] Dongarra, Jack, and Alexey L. Lastovetsky. "High performance heterogeneous computing." John Wiley & Sons, 2009.
[7] Mittal, S. and Vetter, J.S., 2015. "A survey of CPU-GPU heterogeneous computing techniques." ACM Computing Surveys (CSUR), 47(4), pp.1-35.
[8] L. Wu et al., "DOT: Decentralized Offloading of Tasks in OFDMA-Based Heterogeneous Computing Networks," in IEEE Internet of Things Journal, vol. 9, no. 20, pp. 20071-20082, 15 Oct.15, 2022.
[9] A. Reisizadeh, S. Prakash, R. Pedarsani and A. S. Avestimehr, "Coded Computation Over Heterogeneous Clusters," in IEEE Transactions on Information Theory, vol. 65, no. 7, pp. 4227-4242, July 2019.
[10] J. Kim, S. Lee, B. Johnston and J. S. Vetter, "IRIS: A Performance-Portable Framework for Cross-Platform Heterogeneous Computing," in IEEE Transactions on Parallel and Distributed Systems, vol. 35, no. 10, pp. 1796-1809, Oct. 2024.
[11] M. I. Daoud and N. Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," J. Parallel Distrib. Comput., vol. 68, no. 4, pp. 399–409, 2008.
[12] C. -W. Tsai, W. -C. Huang, M. -H. Chiang, M. -C. Chiang and C. -S. Yang, "A Hyper-Heuristic Scheduling Algorithm for Cloud," in IEEE Transactions on Cloud Computing, vol. 2, no. 2, pp. 236-250, 1 April-June 2014.
[13] Mönch, Lars, Hari Balasubramanian, John W. Fowler, and Michele E. Pfund. "Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times." Computers & Operations Research 32, no. 11 (2005): 2731-2750.