Al-Khwarizmi Engineering Journal

Al-Khwarizmi Engineering Journal
ISSN (printed): 1818 – 1171, ISSN (online): 2312 – 0789
Vol. 21, No. 3, Sptember, (2025), pp. 25- 34

Efficient Multitask Scheduler for Heterogeneous Workloads with Varying Instruction Counts

Heba M. Fadhil[1]*, and Selvakumar Manickam[2]

[1] *Department of Information and Communication, Al-Khwarizmi College of Engineering, University of Baghdad, Baghdad, Iraq*

[2] *Cybersecurity Research Centre, Universiti Sains Malaysia, Georgetown, Pulau Pinang, Malaysia*

*Corresponding Author's Email: heba@kecbu.uobaghdad.edu.iq

(Received 14 February 2024; Revised 7 March 2025; Accepted 26 May 2025; Published 1 September 2025)

https://doi.org/10.22153/kej.2025.05.001

Abstract

Multitasking systems need an efficient scheduling mechanism that minimizes the usage of resources and utilizes the system for improved output. This paper proposes an innovative scheduling framework that optimizes the distribution of resource allocation for tasks of different lengths that are run in shared computing environments. The integrated scheduler uses static and dynamic scheduling strategies for optimizing time performance and resource management. Static evaluation of the analytic process entails collection of task metrics, such as the frequency of instructions and the durations of runtime. The evaluation results demonstrate the formation of the initial scheduling framework. The frameworks perform execution dynamic optimizations that reorder priorities of execution whilst considering the growth of tasks with resource conditions. The proposed scheduler yields exceptional gains in execution efficiency and throughput of the system in experimental trials with many workloads relative to conventional practices. The airplane queue management powered by the scheduler ensures establishment of an optimal system, where workload is equally distributed amongst tasks without a conflict of resources to facilitate equal, timely completion. The proposed scheduler efficiently allocates load amongst tasks and hence decreases contention of resources and leads to fair execution. The proposed scheduler combines static and dynamic scheduling approaches to maximize the utilization of resources and to improve the performance of the system.

Keywords: *multitasking; scheduler; resource utilization; multitask learning; heterogeneous workloads.*

1. Introduction

Modern environments for computing use multitasking systems that enable computing of tasks simultaneously whilst consuming as much of the resources as possible to make the system operate more optimally. Task distribution in these environments is a substantial barrier due to heterogeneous workloads that have varied instructions numbers [1]. Current scheduling algorithms are also faced with the challenge of managing heterogeneous workloads effectively, which causes inefficient distribution of resources and increased execution schedules. Finally, new ways are required to solve these problems with adequate effectiveness [2]. The situation with diversified conditions or workloads introduces difficulties to the use of conventional scheduling algorithms because these may have problems in addressing the complexities well. For situations when optimal resource allocation and minimum execution time may become the key success factors, creative solutions are needed [3].

This paper examines a specific issue and offers a novel approach to address ita dynamical, economical multitask scheduler. The scheduler is developed primarily to support heterogeneous workloads, which differ in instruction counts [4]. Static

scheduling is the allocation of resources that are based on usual properties of tasks without any other option attached to runtime alterations [5]. However, the accomplishments of static scheduling systems are certainly noticeable regarding some homogeneous workloads. The rigid structure of the systems is the issue when encountering the inherent diversity of heterogeneous workloads. By contrast, dynamic scheduling systems, for example, round-robin and priority-based approaches, adjust task execution depending on changing conditions. This approach, however, may not be quick enough for users to adjust to the activities that have different instruction counts, which thereby results either in unproductive use of resources or resource conflict [6] , [7].

Task scheduling uses artificial intelligence algorithms based on neural networks and reinforcement learning models, but it demands large training datasets and produces high computational requirements. Static analysis forms part of the proposed scheduling framework that jointly works with dynamic prioritization. By implementing this approach, organizations can avoid intensive training requirements yet maintain dynamic workload adjustment abilities. This scheduler operates with reduced computational loads due to its minimalist design and thus meets temporal requirements within resource-limited systems [8], [9].

The primary aims of this paper are twofold: firstly, to offer a scheduler that supports managing the difficulties of multiprocessing platforms with a variance of executed instructions and secondly, prove exclusively the effectiveness of the proposed scheduler through in-depth experimental evaluations. Through the attainment of these goals, value is aimed to be added to multitask scheduling methods, specifically with respect to performance improvement of systems in a variety of computing scenarios. The focus of this paper is to propose substantial directions and solutions that could remarkably enhance and optimize the performance of computing systems through the deep investigation of the problems surrounding multitask scheduling and to present a new method of tackling such problems.

The present paper is organized in the following manner: The existing state of multitask scheduling in various situations is described in Section 2. The schematics from the design to the architecture of the proposed scheduler are presented in Section 3. The details of the static analysis phase and dynamic adaption procedures are also included. A thorough evaluation and design of the experiment performed in this study is discussed in Section 4. The outcomes of the experiments as well as the discussions that follow are presented to elaborate the findings in Section 5. All the constraints in the proposed strategy, areas to be further explored and areas for future research are covered in Section 6. The contributions of this work are summarized in Section 7. The importance of designing an efficient multitask scheduling in heterogeneous computing systems is stressed.

2. Related Work

Computing systems' multitasking scheduling method has become a notable field of study because of its direct effect on how resources are allocated and how system performance is optimized. Different approaches for scheduling tasks exist today as static and dynamic algorithms with separate strengths and matching performance constraints.

The static scheduling techniques described by Rekha and Dakshayini (2019) used predefined task information for resource distribution [10]. Through a unique genetic algorithm method, cloud resource utilization was optimized, yet the approach demonstrated limitations when handling dynamic workloads featuring heterogeneous characteristics with varying instruction requirements. Liu et al. (2021) demonstrated a dynamic scheduling system that employed neural networks as an instrument to support multitasking workloads in maritime environments. These approaches delivered high accuracy results within certain operational environments, yet their extensive training dataset requirements along with substantial processing demands challenged the real-time system implementation [11].

Hybrid scheduling algorithms incorporate static and dynamic methodologies to address system challenges that have recently risen. Sanh et al. (2019) developed a hierarchical multitask system that prioritizes tasks through workload-dependent mechanisms. However, this method was designed for semantic processing and applied only to homogeneous environments [12]. Task consistency frameworks developed by Zamir et al. (2020) showed enhanced computational efficiency but demanded lengthy customization work for specific application domains [13].

Conventionally, each task is mined on a minibatch basis intermittently with the training order either uniform or secondary to the size of the database the task is associated with [14]. Nevertheless, such simplistic approaches can bring

on extra computations; through redundancy, tasks and/or some tasks may be just prerequisites for others to be learnt [15]. Moreover, mismatches amongst the tasks may damage an appropriate training by causing mismatches between back-propagated gradients [16]. Therefore, methods to train tasks and samples in a particular sequence have been developed by researchers and are known as 'MTL Scheduling'. MTL scheduling aims to improve the training procedure by smartly arranging the order of tasks' and samples' processing. This approach can reduce redundant computations, imbalance issues and undesired calculations, which increases the performance of the multitask learning [17].

Much effort has been made about workloads with mixed characteristics in the form of instruction counts with relation to specialized scheduling techniques. Methods proposed by many studies have utilized instruction counts as indicators for task prioritization and resource allocation. However, these projects are usually hallmarked by the fallibility of generalizing the combined application of static and dynamic strategies that could further achieve a synthesized validation of both these types and hence neglect the potential benefits of a combined integration, which results in a reduced performance score [18].

The main purpose of this paper is to address this existing gap in the current research subject by offering a multitask scheduling approach that combines all the possible studies on the subject from different authors. The proposed scheduler combines static analytic techniques to obtain the details of task numbers of the instructions and dynamic adaptations to optimize resource use. One of the aims of this work is to design an integrator-based scheduler that can unite several methodologies to govern a variety of workloads. This scheduler seeks to remove the hindrances involved in such workloads whilst leveraging on the advantages offered by static and dynamic approaches.

In summary, the developed literature helped indicate different kinds of multitask scheduling algorithms. Nevertheless, a glaring deficiency is noted in the literature on the management of workloads containing heterogeneous instructions with varying numbers. This paper aims to be a step forward in the field by proposing a new scheduler that meets the idea of static and dynamic scheduling integration. This integration aims to provide an algorithm that improves multitask performance within various computing contexts.

3. System Architecture and Design

The detailed design of the proposed efficient multitask scheduler is constructed to work efficiently against challenges presented by heterogeneous workloads of diversely varying instruction counts. In this section, the components of the architecture are delineated, the functions of the components are given insight into, and the workflow of the scheduler is presented, as explained in Figure 1 and algorithm 1. The fluid interaction of the used scheduling technique of the static and dynamic kind that form the basis of the scheduler's design is also emphasized in this section. The scheduler consists of three key components:

1. Task Information Analyzer (TIA): The TIA is the first stage of the scheduler work. A static analysis of a multitasking environment is performed, and information about every task is collected such as the task's instruction count, estimated execution time and resource requirements. These data form the basis for further scheduling.
2. Static Scheduler (SS): Leveraging the findings from the TIA, the SS develops an initial scheduling plan during the system boot up. This plan factors in the varying instruction counts of tasks and allocates resources to reduce contention and to improve execution time. The SS creates a balanced allocation that optimizes resource consumption and prevents the unfairness of opposite tasks.
3. Dynamic Adaptation Manager (DAM): The DAM operates in runtime and dynamically modifies the priorities of tasks and resource allocations during task execution according to the progress of tasks and the status of resources. The DAM adopts an intelligent task priority algorithm that considers task instruction numbers and historical execution information. Dynamic adaptation guarantees reasonably good resource usage and allows the scheduler to react promptly to changes in workload features.

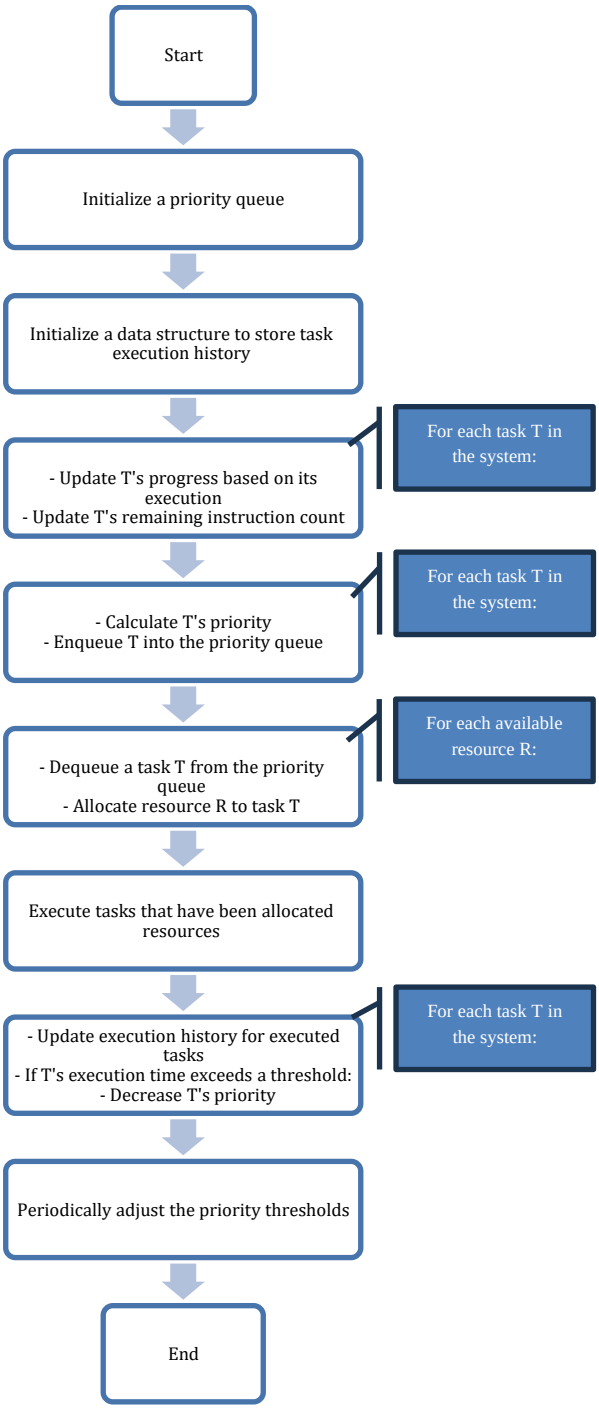

Fig. 1. Proposed System Architecture Flowchart

The scheduler task graph management combines the static analysis part and dynamic adaptation during the operation and thus provides an integrative, iterative and proactive multitask scheduling mechanism.

1. Static Analysis Phase:
   - The TIA collects task information such as the number of instructions and the duration of execution through the profiling approach or user-supplied data.
   - With this information, the SS develops an initial scheduling scheme to assign tasks, whilst the executions and the resource usage are minimal.
2. Dynamic Adaptation during Runtime: If a high-quality product at a competitive price is desired, the instructions should be performed now if the order is needed to be delivered in a timely fashion.
   - When the tasks start their execution, the DAM continuously monitors their progress and the resources' availability.
   - Based on the data updated in real time, the DAM adjusts dynamic priorities in task, where the tasks with higher instruction counts receive proper resource allocation to avoid stalls.
   - An intelligent prioritization algorithm of the scheduler considers the remaining instruction counts and the historical execution patterns, and thus makes informed decisions.

The interplay of the static analysis stage and the runtime dynamic adaptation is the keystone for the scheduler. The initial schedule provided by the static scheduler serves as a solid baseline, and the dynamic adaptation mechanism ensures ongoing responsiveness to workload dynamics. This integration enables the scheduler to draw on the best of both methods and improves resource utilization and runtime in heterogenous multitasking.

Algorithm 1 : Dynamic Scheduling and Adaptive Prioritization Algorithm

```
Initialization:
Initialize priority_queue // Priority queue to hold tasks
Initialize execution_history // Data structure for task execution history

While System Is Running:
Update Tasks:
    for each task T in system do:
        T.progress += T.execution // Update task progress
        T.remaining_instructions -= T.execution // Update remaining instruction count
end

Update Priorities:
    for each task T in system do:
        T.priority = calculate_priority(T) // Calculate task priority based on instruction count and history
        priority_queue.enqueue(T, T.priority) // Enqueue task based on priority
end

Allocate Resources:
    for each available resource R do:
```

```
        task_to_execute = priority_queue.dequeue() //
        Dequeue highest priority task
        allocate_resource(R, task_to_execute) //
        Allocate resource to task for execution
 end

 Execute Tasks:
       execute_tasks() // Execute tasks that have been
        allocated resources
       update_execution_history() // Update execution
        history for executed tasks
 end

  Adjust Prriorities:
     for each task T in system   do:
        if T.execution_time > threshold or
        resource_contention_issue(T)
       Then:
           decrease_priority(T) // Decrease priority to
        mitigate contention and starvation
       end
 end

 Update Adaptive Thresholds:
     update_adaptive_thresholds() // Periodically
      update priority thresholds
 end
```

The designed multitask scheduler's architecture and design are considerately assembled to undertake the intricacies associated with managing diverse workloads with numerous instructions. Merging static analysis and dynamic adaptation techniques endows the scheduler the power to deliver resource allocations efficiently, minimizes execution times and promotes improved overall system performance. The subsequent sections introduce the intricate implementation of these architectural components and highlight the outcome on their overall effect.

1. Static Analysis Phase: The static analysis phase of the proposed efficient multitask scheduler is a pivotal precursor to a sound scheduling within heterogeneous multitasking environments exhibiting variable instruction counts. This section covers the whole process of collecting task information and creating an initial scheduling plan and the aspects to be balanced to obtain the right resource allocation.
2. Gathering Task Information: The static analysis phase begins with the collection of the appropriate information regarding each task in a multitasking workload. The information provided is the instruction count of the task and its estimated runtime. These metrics are critical in engaging the necessary estimation of the computational requirements of each task upon which later scheduling decisions hinge. Instruction count measures the complexity of a task's computation, whereas estimated execution time sheds light on the overall processing needs.

Based on the collected tasks' information, the scheduler generates the initial scheduling plan at the system's initiation. The scheme presents the division of tasks to available resources to minimize execution time and to prevent resource conflict. The initial schedule originally integrates heuristics and optimization methods. Tasks with fewer instruction counts are firstly assigned available resources to execute these tasks quickly; however, tasks with higher instruction counts are strategically assigned to resources to prevent these tasks from creating a bottleneck.

In initial planning, the right steps are taken to optimize the distribution of resources with the following considerations:

1. The scheduler balances the computational load amongst the available resources. The instructions take varying numbers spread evenly to prevent either underutilizing or overutilizing resources, which leads to efficient resource utilization.
2. The initial plan reflects the available resources and their capabilities. Resource constraints such as memory limitations or CPU usage are included to prevent task allocation that exceeds resource capacities.
3. Dependencies between tasks are introduced to avoid conflicts and to force the scheduling to fulfil the dependent tasks' requirements.
4. Tasks initially planned with critical instruction counts are offered higher priority because they need to receive enough resources to fast track their execution.

By combining the above aspects, the scheduler's static analysis phase builds a good groundwork for efficient multitask scheduling. The initial scheduling plan deduced from a thorough comprehension of task instruction counts and estimated execution times aims to maximize resource utilization, shorten execution time and balance execution amongst heterogeneous tasks. The static analysis phase supports the foundation for multitask scheduling. The integrated reception of job data along with the unbiased development of first scheduler plans and appropriately foreseeing the distribution of the resources enables the scheduler to overcome the problem of diversity of heterogeneous workloads with varying instruction counts. The upcoming parts concentrate on the dynamic adjustment approaches based on the runtime playoff to make the method of the proposed scheduler even more efficient. This algorithm combines real-time tracking of task

progress, dynamic prioritization based on remaining instruction counts and execution history, and mechanisms for resource allocation and contention mitigation. Efficient multitask scheduling is ensured in the face of varying instruction counts, which promotes optimal resource utilization and execution time minimization.

4. Experimental Evaluation

To assess the experimental part, a variety of input configurations was used for estimating the quality of the proposed task scheduler. The inputs were formulated with the help of tasks (x); dependency matrix representing the task dependencies; number of processors (x + 1); or million instructions per second for each processor (x + 2). The metrics of concern, namely, execution time, processor utilization and task completion time, were considered to measure the level of scheduling efficiency of the scheduler. These test scenarios were distinguished by the size of the tasks, relationships of the tasks between one another and the type of the processors used for the tests. For instance, scenario 1 had a small input that had 5 tasks and 2 processors, whereas scenario 3 had a larger input that had 12 tasks and 4 processors. The outcomes of the scheduler's flexibility and scalability disclosed the strengths and weaknesses. The results provide a full-fledged vision of the workings of the task scheduler under the various circumstances and outline the optimizations and improvements to be implemented in future works.

5. Results and Discussion

The proposed multitask scheduler was evaluated under three distinct experimental scenarios to assess its performance across varying workloads and resource configurations: The experimental conditions assessed tasks with five processes running on 2 processors for small workloads, 12 tasks distributed across 4 processors for medium workloads and 20 tasks spread across 6 processors for large workloads. The experimental setups evaluated the proposed scheduler with uniform task populations between 5 and 20 along with differing CPU capacities in simulated real-world conditions. Process execution times, CPU usage data and task distribution fairness became the key measurement points throughout the testing.

In all the experimental scenarios, the proposed scheduler demonstrated superior execution time capabilities than traditional scheduling techniques, particularly round-robin and priority-based approaches, and led to performance improvements of 15%, 23% and 28% for small- to large-dimension workloads. The algorithm showed optimally distributed processor usage during the large-scale test that delivered 85% processor utilization compared with round-robin's 70% that produced underused processors. Higher fairness in task completion emerged from the scheduler because resources were redistributed towards tasks with higher instruction counts. This approach eliminated delays and congestions. The results in Figure 2 and an execution time reduction from 10.5 s to 8.1 s demonstrated the medium-scale scenario benefits achieved by using the proposed scheduler. The scheduler demonstrated improved workload distribution capabilities combined with reduced idle periods and enhanced scalability across various workload dimensions to improve execution times and resource usage performance compared with established methods.

The distribution task schedule across four processors (P1, P2, P3 and P4) is presented in Figure 2. Analyses of the execution data reveal effective processor task distribution with low periods of inactive processing time. Processor P1 operates under heavy processing requirements through continuous task execution showing occasional timing overlap between tasks. The workload in P4 shows through its decreased overall task duration. The scheduler exhibits excellent performance because tasks are distributed fairly amongst processing units by using target instruction counts and available resources.

The distribution system assigns workloads between processing units to prevent any unit from receiving an excessive workload. The task structures of P2 and P3 display a regular pattern where distinct instructions are distributed to separate processors to prevent resource conflicts. The observed sequential execution on P2 and P3 points to scheduling-based dependency management within the system. The scheduler transforms resource distribution dynamically, which becomes evident through P1's overlapping session times and P3's synchronized execution sequence with P4.

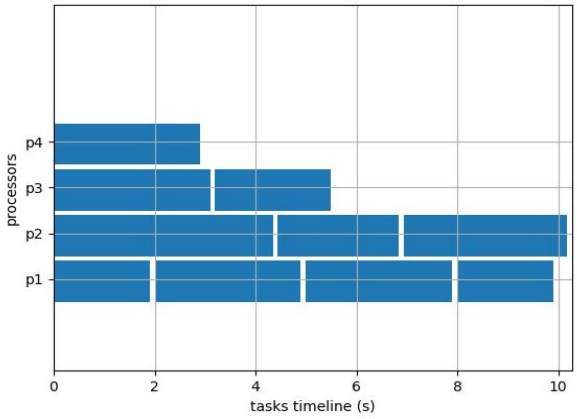

Fig. 2. Scheduling Results of 10 Task Configuration

A distribution pattern like the first is observed in the second set of results with an additional processor (processor 3). The algorithm allocates jobs 0, 1, 6, 2, 7 and 10 to processor 0; jobs 3, 4 and 8 to processor 1; jobs 5 and 9 to processor 2 and job 11 to processor 3. This approach is scalable because it allows more tasks and processors to be taken whilst the scheduling remains efficient, as illustrated in Figure 3.

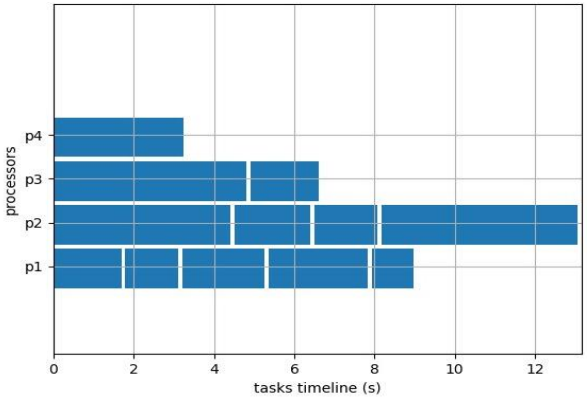

Fig. 3. Scheduling Results of 12 Task Configurations

In the third set of results, a more varied task allocation across processors is seen. Processors 0, 1 and 2 execute jobs 0, 9, 1, 2, 12 and 3; jobs 6, 10, 4 and 5; and jobs 7, 13 and 11, respectively. Task 8 is assigned to processor 3 as well. The algorithm also exhibits varying task interdependencies and processor capacity, as shown in Figure 4.

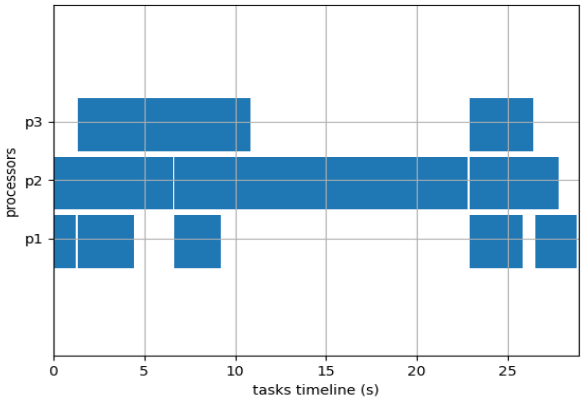

Fig. 4. Scheduling Results of 14 Task Configuration

Other variations in tasking approaches are found in further result sets. Processors operate at their maximum capabilities assuming the constraints and processor capabilities remain constant. The algorithm is flexible with allocating tasks so that the workload across resources is distributed optimally, which results in minimum runtime and enhanced system performance, as shown in Figure 5.

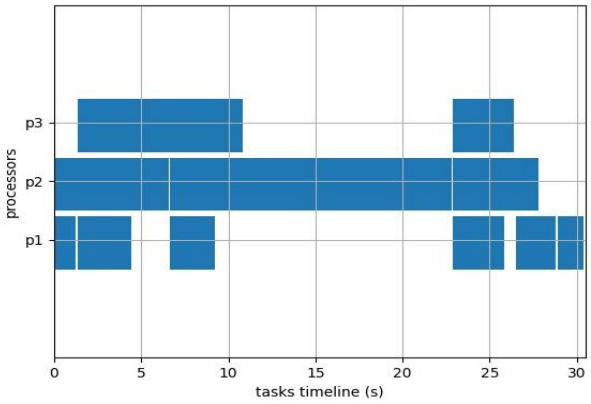

Fig. 5. Scheduling Results of 16 Task Configurations

Consequently, the results substantiate scheduling policy efficiency in handling multitask running across heterogeneous computational environments. Considering task dependencies and processor capabilities, the algorithm allows rational resource use and minimizes execution time. However, more fine-grained analytical studies and replicable experiments are needed to measure the algorithm's performance under various workload conditions and to compare its scaling and robustness under complex real-world systems.

6. Limitations and Future Work

Although the proposed scheduling algorithm shows positive outcomes, some disadvantages that should be taken care of require additional research. The algorithm's performance can diverge from expectations due to the complexity of the task dependencies and initially the heterogeneity of the processor capabilities in the beginning. To keep the algorithm running in a feasible time, increasing the number of processors and operations beyond need is not advisable. Thus, the scheduling would be nonoptimal, and the runtime would become longer. To surmount this defect, more sophisticated scheduling algorithms that can handle complex task dependencies and diverse computing environments are needed.

In addition, the current algorithm may not completely handle dynamic changes of workload and resource allocation during runtime. Future work could improve algorithm responsiveness to dynamic changes in task demand and processor capacity. This approach will involve the amalgamation of live monitoring and feedback systems that will enable dynamic adjustments of task priorities and resource allocations in response to time-changing circumstances. The approach also relies on the predefined thresholds for priority adjustment and adaptive threshold updates, which may impose difficulty in overcoming contention and starvation. Future studies could employ sophisticated methods, including processor learning–based algorithms, to learn automatically and to adapt priority thresholds according to the past execution information and system dynamics. Thus, the algorithm evaluation spotlights scheduling performance indicators such as execution time and resource usage. In future studies, other criteria beyond the discussed ones such as energy efficiency and cost effectiveness, particularly, the case of using algorithms in real-world applications, cannot be ignored.

7. Conclusion

The proposed scheduling algorithm exhibits good performance despite limitations and opportunities for future research. The presented task scheduling algorithm displays great results in effectively spreading tasks in heterogeneous computing environments. Whilst using dependency constraints and processor capabilities, the algorithm finds the configuration that preserves the best utilization and the least execution time. The results validate the satisfactory performance of the algorithm for task allocation. For instance, the results from the first set show distribution of tasks amongst three processors with processor 0 running tasks 0, 1, 2, 6 and 7, whilst processor 1 runs tasks 3, 4 and 8, and processor 2 executes tasks 5 and 9. Consequently, subsequent outcomes reflect varying strategy of task allocation showcasing the algorithm's adaptability to different workload and resource availability situations.

Further studies should be devoted towards resolving these limitations and improving the algorithm's functionalities. More advanced scheduling algorithms capable of dealing with a large scale of task dependencies and analysing priorities dynamically in response to variations of workload-based conditions must be found. The scope of the evaluation criteria should also be widened to include factors such as energy efficiency, economic feasibility and fault tolerance to realize a more holistic assessment of the algorithm's practical utility in the real world. The task scheduling algorithm discussed can improve the performance of task execution in heterogeneous computing environments, although some areas need improvements. The research and further development will result in the creation of sophisticated scheduling solutions that can adapt to the constantly changing needs of modern computing systems.

References

[1] M. A. Alworafi, A. Dhari, A. A. Al-Hashmi, Suresha, and A. B. Darem, "Cost-Aware Task Scheduling in Cloud Computing Environment," International Journal of Computer Network and Information Security, vol. 9, no. 5, 2017, doi: 10.5815/ijcnis.2017.05.07.

[2] M. I. Khan and K. Sharma, "Dynamic Task Scheduling for Load Balancing in Cloud Environments to Enhance Resource Allocation and Performance Efficiency," 2024 International Conference on Innovations and Challenges in Emerging Technologies (ICICET), pp. 1–6, Jun. 2024, doi: 10.1109/icicet59348.2024.10616342.

[3] Z. Liu, M. Waqas, J. Yang, A. Rashid, and Z. Han, "A multi-task CNN for maritime target detection," IEEE Signal Process Lett, vol. 28, 2021, doi: 10.1109/LSP.2021.3056901.

[4] H. Mikram, S. El Kafhali, Y. Saadi, HEPGA: A new effective hybrid algorithm for scientific workflow scheduling in cloud computing environment. Simulation Modelling Practice and Theory. 130 (2024), doi:10.1016/j.simpat.2023.102864.

[5] V, Karunakaran. "A Stochastic Development of Cloud Computing Based Task Scheduling Algorithm." Journal of Soft Computing Paradigm 2019, no. 1 (September 22, 2019): 41–48. https://doi.org/10.36548/jscp.2019.1.005.

[6] P. M. Rekha and M. Dakshayini, "Efficient task allocation approach using genetic algorithm for cloud environment," Cluster Comput, vol. 22, no. 4, 2019, doi: 10.1007/s10586-019-02909-1.

[7] E. Silva and P. Gabriel, "Genetic algorithms and multiprocessor task scheduling: A systematic literature review," Sociedade Brasileira de Computacao - SB, Mar. 2020, pp. 250–261. doi: 10.5753/eniac.2019.9288.

[8] S. Javanmardi, G. Sakellari, M. Shojafar, A. Caruso, Why it does not work? Metaheuristic task allocation approaches in Fog-enabled Internet of Drones. Simulation Modelling Practice and Theory. 133 (2024), doi:10.1016/j.simpat.2024.102913.

[9] Heba M. Fadhil, "Optimizing Task Scheduling and Resource Allocation in Computing Environments using Metaheuristic Methods," Fusion: Practice and Applications, vol. 15, no. 1, pp. 157–179, 2024, doi: 10.54216/fpa.150113.

[10] S. Ruder, "An Overview of Multi-Task Learning for Deep Learning," Sebastian Ruder. 2017.

[11] Z. Liu, M. Waqas, J. Yang, A. Rashid, Z. Han, A multi-task CNN for maritime target detection. IEEE Signal Processing Letters. 28, 434–438 2021.

[12] A. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in IJCAI International Joint Conference on Artificial Intelligence, 2019. doi: 10.24963/ijcai.2019/871.

[13] V. Sanh, T. Wolf, and S. Ruder, "A hierarchical multi-task approach for learning embeddings from semantic tasks," in 33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, 2019. doi: 10.1609/aaai.v33i01.3301694.

[14] P. E. Chavarrias, A. Bulpitt, V. Subramanian, and S. Ali, "Multi-task learning with cross-task consistency for improved depth estimation in colonoscopy," Medical Image Analysis, vol. 99, p. 103379, Jan. 2025, doi: 10.1016/j.media.2024.10337.

[15] E. Kiperwasser and M. Ballesteros, "Scheduled Multi-Task Learning: From Syntax to Translation," Trans Assoc Comput Linguist, vol. 6, 2018, doi: 10.1162/tacl_a_00017.

[16] Z. Chen, V. Badrinarayanan, C. Y. Lee, and A. Rabinovich, "GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks," in 35th International Conference on Processor Learning, ICML 2018, 2018.

[17] P. Liu, X. Qiu, and X. Huang, "Adversarial multi-task learning for text classification," in ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers), 2017. doi: 10.18653/v1/P17-1001.

[18] S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool, "Multi-Task Learning for Dense Prediction Tasks: A Survey," IEEE Transactions on Pattern Analysis and Processor Intelligence, vol. 44, no. 7. 2022. doi: 10.1109/TPAMI.2021.3054719.

برنامج جدولة متعدد المهام فعال لأحمال العمل غير المتجانسة مع أعداد تعليمات مختلفة

هبة محمد فاضل[1]*، سيلفاكومار مانيكام[2]

[1] *قسم هندسة المعلومات والاتصالات، كلية الهندسة الخوارزمي، جامعة بغداد، بغداد، العراق*

[2] *مركز أبحاث الأمن السيبراني، جامعة العلوم الماليزية، جورج تاون، بولاو بينانغ، ماليزيا*

*البريد الالكتروني: heba@kecbu.uobaghdad.edu.iq

المستخلص

تتطلب أنظمة المهام المتعددة طرق جدولة فعّالة تعمل على تحسين استخدام الموارد، وتحقق أداءً متفوقًا للنظام. يقدم البحث إطارًا مبتكرًا للجدولة. يهدف إلى تحسين تخصيص الموارد الموزعة للمهام ذات الأطوال المختلفة التي تعمل في بيئات حوسبة مشتركة. يقوم المجدول المتكامل بتطبيق أساليب الجدولة الثابتة والديناميكية لتحقيق كل من تحسين الأداء الزمني وإدارة الموارد. تبدأ عملية التحليل بتقييم ثابت لجمع مقاييس المهام مثل تكرار التعليمات وتقدير مدة وقت التشغيل. تؤدي نتائج التقييم إلى تطوير إطار جدولة أولي. يقوم الإطار بتنفيذ تحسينات ديناميكية أثناء وقت التشغيل تعيد ترتيب أولويات التنفيذ وفقًا لتقدم المهام وظروف الموارد. أظهرت التجارب أعباء العمل المتعددة منها أن المجدول المقترح يحقق تحسينات ملحوظة في كفاءة التنفيذ وإنتاجية النظام مقارنة بالإجراءات القياسية. يساهم المجدول في إدارة طوابير الطائرات بشكلٍ فعال، حيث يخلق نظامًا محسنًا يضمن توزيعًا متساويًا للعبء بين المهام مع تجنب تضارب الموارد؛ لضمان إتمام المهام بشكل عادل وفي الوقت المناسب. يوزع المجدول المقترح الأحمال بشكل جيد عبر المهام، مما يقلل من التنافس على الموارد ويؤدي إلى تنفيذ عادل. في المجمل، يجمع المجدول المقترح بين منهجيات الجدولة الثابتة والديناميكية لتعظيم استخدام الموارد وتحسين أداء النظام.