BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **25**, No 3
Sofia • 2025 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2025-0026

HLSA – A New Hybrid List Scheduling Algorithm for Fog Computing

Hend Gamal El Din Hassan Ali, Imane Aly Saroit, Amira Mohamed Kotb

Faculty of Computers and Artificial Intelligence, Cairo University, Cairo, Egypt
E-mails: hendgamal@fci-cu.edu.eg a.kotb@fci-cu.edu.eg i.saroit@fci-cu.edu.eg

***Abstract*:** *A hybrid list scheduling algorithm is applied in fog computing with heterogeneity in available resources and incoming scheduling units. The scheduling units that need to be scheduled can be independent with no precedence constraints, so that tasks can be executed in parallel. On the other hand, precedence constraints can be present between tasks and represented by a Directed Acyclic Graph* (*DAG*). *Some scheduling algorithms are efficient for independent tasks, while others excel in handling dependency workflows. This paper proposes a Hybrid List Scheduling Algorithm* (*HLSA*) *for all scheduling unit types and examines the impact of incoming scheduling unit types on the performance of the proposed algorithm. HLSA assigns priority to sensitive time tasks in a cumulative way to achieve minimum latency for sensitive IoT applications in fog computing and to get minimum makespan, computation cost, and communication cost. Also, HLSA aims to achieve the highest utilization of the exploitation of gaps in processors.*

***Keywords*:** *Fog computing, Cloud computing, Internet of Things, List Scheduling Algorithm, Directed acyclic graph, Workflow.*

1. Introduction

Fog computing is considered an extension of cloud computing, used to overcome the drawbacks of cloud computing, particularly in light of the massive data produced by IoT applications [1-6]. Fog computing shifts centralized processing in the cloud to the network edges [7-10]. Because of the limited resources in any computing environment and to satisfy different users' requirements, a scheduling process is needed in fog computing. The scheduling process is considered the most significant NP-complete problem [11-13]. There are many scheduling algorithms used in fog environments with different classifications.

In this paper, our focus will be on heuristic list scheduling algorithms. Heuristic list scheduling algorithms can be classified based on the types of scheduling units to batch, dependency, and batch-dependency algorithms [14]. Batch scheduling algorithms work with independent tasks. They cannot be applied

with precedence constraints between tasks. Dependency scheduling algorithms work with a dependency workflow that is represented by Directed Acyclic Graphs (DAG) and cannot deal efficiently with independent tasks. Batch-dependency scheduling algorithms work first as dependency algorithms with a dependency workflow and form groups of independent tasks, then work as a batch algorithm in each group.

The proposed Hybrid List Scheduling Algorithm (HLSA) aims to efficiently work with different types of scheduling units without requiring assumptions or restrictions on the types of incoming tasks. It also takes into consideration the sensitive time tasks in some IoT applications, such as health care, intelligent traffic control, and monitoring applications [15-21]. The contributions of our work are summarized as follows.

- First, the proposed algorithm is designed to accept any type of scheduling units without any restrictions, so there are three cases studied: first, when all the scheduling units are independent, second, when all the scheduling units are dependent on each other, and third, when the scheduling units are a combination of independent and dependent.
- Second, the proposed algorithm relies on ranking value to schedule tasks, which is increased by increasing the sensitivity of time to tasks and increasing the number of successors to each task. That led to a decrease in the latency of sensitive tasks and reduced waiting time for all the tasks.
- Third, the proposed algorithm intends to utilize the exploitation of gaps in the timing line that formed as a result of precedence constraints between tasks that cause waiting for certain task to its parent tasks to execute first, and transferring data between processors.
- Finally, a complete comparison is applied between the proposed algorithm and other algorithms in the first two cases. In the third case, the changes in the behaviour of the proposed algorithm are declared when increasing the percentage of precedence constraints between tasks from 0% (all tasks are independent) to 20%, 50%, 80% and 100% (all tasks are dependent on each other).

The rest of the paper is organized as follows: Section 2 describes the previous related work used in fog computing and its classification. Section 3 highlights the steps of the proposed algorithm. Section 4 illustrates the performance metrics used to evaluate the proposed algorithm. Section 5 defines the simulation environment parameters. Section 6 shows the performance evaluation of the proposed algorithm compared to the HEFT, Min-Min, TS-QoS, and improved list-based task scheduling algorithms. Conclusion and future work are conducted in Section 7.

2. Related work

Scheduling algorithms used in fog computing can be classified based on how the problem is solved into heuristic, meta-heuristic, hybrid heuristic, and hyper-heuristic algorithms [22]. Heuristic algorithms are used to solve a specific problem with known parameters [23] as in [24] proposed a Critical Task First Scheduler (CTFS), in [25] proposed a Priority and dependency-based DAG Tasks Offloading

Algorithm (PDAGTO Algorithm) and in [26] proposed Efficient Resource Allocation and Management strategies for Energy Efficiency (ERAM-EE).

Meta-heuristic algorithms are used to solve problems in a wide range [27]. Most meta-heuristic algorithms are inspired by nature, classified as Swarm Intelligence (SI), bio-inspired, and physical and chemical algorithms [28]. SI algorithms are inspired by the behaviour of multi-agents, such as Particular Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Improved Particle Swarm Optimization (IPSO) [29]. Bio-inspired algorithms are inspired by biological systems like Genetic Algorithms (GA), Whale Optimization Algorithms (WOA) [30], improved genetic algorithms for permutation-based optimization problems (IGA-POP) [31], Multi-Agent System-based Genetic Algorithms (MAS-GA) [32], and Enhanced JellyFish Algorithm (IJFA) [33]. Physical and chemical algorithms are inspired by physical and chemical systems like harmony search, a simulated annealing algorithm, and an Electric Earthworm Optimization Algorithm (EEOA) [34]. Another type of meta-heuristic algorithm not inspired by natural systems, like Drawer Cosine Optimization (DCO) [35], which is inspired by choosing objects from various drawers to create perfect formations.

Hybrid heuristic algorithms combine two or more of the heuristic algorithms to solve a predefined problem as a Hybrid Heuristic Algorithm (HH Algorithm) [36]. Hyper-heuristic algorithms combine more than one heuristic algorithm, like a hybrid heuristic, but use a selector to decide which one of the heuristic algorithms is best to be used at each iteration, as HH Algorithm [37]. All these scheduling algorithms can be used in fog computing based on the system's requirements to achieve the best mapping between incoming tasks and available resources.

On the other hand, some scheduling algorithms can only work with no prior information about available resources or the tasks that need to be executed. They are applied in real-time and are called dynamic/online algorithms, like dynamic scheduling algorithms [38]. While other algorithms need prior information about resources and tasks before starting scheduling, they are called static or offline algorithms, like come First-Come, First-Served Algorithm (FCFS Algorithm), the Round Robin Algorithm (RR Algorithm), and the Weighted Round Robin Algorithm (WRR Algorithm) [39]. Also, scheduling algorithms can be classified based on the types of incoming scheduling units, into batch, dependency, and batch-dependency algorithms [8]. Batch algorithms work with independent tasks with no precedence constraints, like Min-Min [40] and TS-QoS algorithms [41]. Dependency algorithms work with a dependent workflow represented as DAGs, like Heterogeneous Early Finish Time algorithms (HEFT) [42]. The batch-dependency algorithms work with dependent workflows first as a dependency mode, then divide all workflows into groups of independent tasks, and work as batch mode in each group, like an improved list-based task scheduling algorithm [43].

Table 1. Scheduling algorithms in fog computing

Algorithm	Environment	Scheduling	Techniques	Tasks	Performance metrics
DCO ([35], 2024)	Fog-cloud	Dynamic	Meta-heuristic	Independent	Load, Energy, Makespan, Time, Memory.
ERAM-EE ([26], 2024)	Fog	Dynamic	Heuristic	Not mention	Energy efficiency, Response time, Processing time
EEOA ([34], 2023)	Fog-cloud	Dynamic	Meta-heuristic	Dependent	Makespan, Total cost, Energy consumption
IGA-POP ([31], 2022)	Fog-cloud	Semi-dynamic	Meta-heuristic	Independent	Makespan, Execution cost, Failure rate, Average latency
IJFA ([33], 2022)	Fog-cloud	Dynamic	Meta-heuristic	Independent	Completion time, Resource utilization
Dynamic scheduling algorithm ([38], 2022)	Fog-cloud	Dynamic	Heuristic	Not mention	Throughput, Latency
PDAGTO ([25], 2021)	Fog	Static	Heuristic	Dependent	Average latency, Energy consumption
WOA ([30], 2021)	Fog-cloud	Static	Meta-heuristic	Dependent	Energy consumption, Total cost
MAS-GA ([32], 2021)	Fog-cloud	Dynamic	Meta-heuristic	Dependent	Execution time, Response time, Makespan, Cost, Reliability, Availability
WRR ([39], 2021)	Fog-cloud	Static	Heuristic	Not mention	Throughput, Latency, Complexity, Fairness
Improved list-based ([43], 2021)	Fog	Static	Heuristic	Dependent	Average scheduling length ratio, Speedup, Makespan
IPSO ([29], 2019)	Fog-cloud	Static	Meta-heuristic	Dependent	Makespan, Total cost
HH ([36], 2019)	Fog	Static	Hybrid Heuristic	Independent	Execution time, Energy consumption, Reliability
HH ([37], 2017)	Fog	Static	Hyper-heuristic	Independent	Execution time, Cost, Energy consumption, Network usage
HEFT ([42], 2017)	Fog-cloud	Static	Heuristic	Dependent	Makespan
Proposed (HLSA)	Fog	Static	Heuristic	Dependent / Independent	Makespan, Total cost, Latency

The Heterogeneous Earliest Finished Time Algorithm (HEFT Algorithm) and improved list-based task scheduling algorithm are used with a dependency workflow that is represented by a Directed Acyclic Graph (DAG). However, different ways are used to schedule the workflow in both. HEFT [42] has two phases. In the first phase, it gets an upward ranking value for each task in the DAG based on the mean value of both computation and communication costs. Then, it sorts the tasks in descending order according to the ranking value of each task. In the second phase, it chooses a high-ranking task and assigns it to the processor with minimal execution time [44]. An improved list-based task-scheduling algorithm has three phases [43]. In the first phase, independent tasks are clustered in a DAG from up to down into groups so that the tasks in each group can be executed in parallel way. In the second phase, priorities are assigned to each task in each group based on three attributes: cumulative execution cost, data transfer cost, and rank of the predecessor task. Then, it sorts the tasks in each group based on the calculated priority and selects the task with high priority in the first group to be scheduled. The third phase is the processor selection process; if the processor with the earliest finished time has the minimum execution time, the selected task is assigned to that processor. Otherwise, if the processor with the earliest finished time does not have a minimum execution time, it will calculate the cross-over threshold. If the cross-over

threshold is between [0-3], it will assign the selected task to the processor with the earliest finished time; otherwise, it will assign the task to the processor with the minimum execution time. The improved list-based algorithm achieved better performance compared with HEFT, Predicted the Earliest Finish Time (PEFT), Minimal Optimistic Processing Time (MOPT), and the Standard Deviation-Based Algorithm for Task Scheduling (SDBATS). At the same time, the Min-Min and TS-QoS algorithms are both used for independent tasks. Min-Min selects tasks with minimum execution time for all processors to be scheduled first [40]. The TS-QoS algorithm first selects the high-priority tasks to be scheduled based on quality of service driven [41]. Table 1 shows a classification of scheduling algorithms used in fog computing.

3. Proposed scheduling algorithm

The proposed algorithm is considered a static scheduling algorithm, so before starting the scheduling process, the tasks that need to be executed, the available resources (processors), the communication cost, the computation cost, and the precedence constraint between tasks should be known. Also, the proposed algorithm is considered a non-preemptive scheduling algorithm, so once the task is selected, it can't be stopped or interrupted until it is completely executed [45].

If all the available resources are identical, the system is called homogeneous, which means that the same task can be executed on different machines with the same execution time. However, if the available resources have different architectures (capability, memory, computational speed), then the system is called heterogeneous, which means that the same task has different execution times for different machines [46]. Homogeneous systems are easy to manage compared to heterogeneous [47]. The proposed algorithm is considered for a heterogeneous system. It is not applied only to heterogeneity in resources but also to heterogeneity in incoming scheduling units. So it can deal with both dependent and independent workflows.

3.1. Proposed scheduling algorithm parameters

The proposed algorithm is considered as a heuristic list scheduling algorithm that can deal with groups of dependent workflows and independent tasks in a fog environment with size n, where t_i, i = 1, 2, 3,…, n, and assign them to a set of heterogeneous fully connected processors P with size m where P_j, j = 1, 2, 3,…, m. The groups of dependent workflow are represented by groups of Directed Acyclic Graphs (DAGs), whereas each DAG is represented as G= (T, E), where T is a set of executable tasks/nodes and E is a set of directed communication edges e(T_i, T_K) ∈ E, which represents the dependency or precedence constraint between two tasks T_i and T_K. A task with no parent is called an entry task, while a task with no child is called an exit task. Each edge has non non-negative weight representing the amount of data needed to be exchanged between tasks. There are five defined metrics in the proposed algorithm.

First metric. The Expected Execution Matrix (EEM) is a matrix with order $n \times m$, where the n rows are the number of tasks, and the m columns are the number of processors. Each element in the EEM matrix W_{ij} represents the expected execution time of the task t_i in processor P_j. Matrix W_{ij} for given task t_i is not the same for all processors because of the heterogeneous resources.

Second metric. The Data Transfer Matrix (DTM) is a matrix of order n×n where n is the number of tasks. Each element in the DTM matrix (D_{ik}) represents the precedence constraint between two tasks t_i and t_k. Also, D_{ik} defines the amount of data needed to be transferred from one processor to another if and only if the parent task t_i and child task t_k are executed in different processors; otherwise, D_{ik} is neglected because no data needs to be transferred between processors.

- If $\sum_{k=1}^{n} D_{ki} = 0$ and $\sum_{k=1}^{n} D_{ik} > 0$ then t_i is an entry task that has no parents and has at least one child.
- If $\sum_{k=1}^{n} D_{ki} > 0$ and $\sum_{k=1}^{n} D_{ik} = 0$ then t_i is an exit task that has at least one parent and has no children.
- If $\sum_{k=1}^{n} D_{ki} > 0$ and $\sum_{k=1}^{n} D_{ik} > 0$ then t_i is a task in an intermediate layer in the DAG that has at least one parent and one child. The exit and intermediate tasks cannot start execution until their parents have finished execution, and the required data is transmitted from the parent's processors to their successors' processors.
- If $\sum_{k=1}^{n} D_{ki} = 0$ and $\sum_{k=1}^{n} D_{ik} = 0$ then t_i is an independent task that has no parent or child.

Third metric. Communication Cost Matrix (CM) is a matrix with order $m \times m$ where m is the number of processors, each element in the CM matrix C_{jy} defines the cost of transferring data between two processors P_j and P_y, if the parent task is executed in the processor P_j and the child task is executed in the processor P_y. Matrix CM is a symmetric matrix as $C_{jy} = C_{yj}$ because the cost of transferring data from the processor P_j to P_y is equal to the cost of transferring data from the processor P_y to P_j. Also, the CM matrix has zero diagonal as $C_{jj} = 0$ because the cost is neglected if parent and child tasks are executed on the same processor j.

Fourth metric. Mapping List (ML) is considered the output of the proposed algorithm that is used to operate the evaluation process by recording the mapping of each task t_i to its assigned processor P_j, execution time for each task in the assigned processor W_{ij}, Start execution Time $ST_{i,}$ and Finished execution Time FT_i. So ML is a matrix with order $n \times 5$, where n rows represent the number of tasks and 5 represents the five outputs that are used as performance metrics.

Fifth metric. Urgent vector U_v is a vector with length n; each element in the U_v vector U_{vi} represents how sensitive task t_i is to time. Based on U_v and the types of scheduling units can define the priority of each task cumulatively and record it in a priority vector P_v. If the task is independent or an entry task and urgent, then it will increment its priority in P_v. If the task is an exit task or in the intermediate layer of DAGs and urgent, then it will increment its priority and increment the priority of its parent tasks until it reaches the entry task in P_v.

3.2. Proposed scheduling algorithm steps

The proposed algorithm is divided into two phases. The first phase calculates the ranking value of each task based on its attributes. The second phase creates the list of tasks ready to be executed. The list is included in the first parent tasks and urgent independent tasks. Then, it selects the task with the highest-ranking value from the list, assigns it to the processor with the earliest finished time, and updates the ML matrix. If the assigned task has successors, then their successors will be added to the list after the parent task has finished its execution and the required data is transferred. After assigning the parent tasks, their successors, and the urgent independent tasks, the remaining normal independent tasks can be assigned to the processors in their waiting time for transferring the required data between the tasks and their successors (gaps), or it can select the task with the minimum execution time to schedule first.

The first phase. The ranking value is calculated for each task based on three attributes: Average Execution Time (AET), Average Communication Cost (ACC), and Priority of tasks (P_{v}). The AET is calculated from EEM by taking the average execution time of each task in all processors to determine the duration of the execution of each task [48]. $\mathrm{AET}(i)$ for a given task t_i is defined as

$$\mathrm{AET}(i) = \frac{\sum_{j=1}^{m} W_{ij}}{m}, \tag{1}$$

where: $\mathrm{AET}(i)$ is the average execution time of the task t_i; W_{ij} is the expected execution time of the task t_i in processor P_j; m is the total number of available processors.

ACC is evaluated by DTM and CM to determine the cost of transferring data for a given task to their successors, if the task and its successors are not executed in the same processor [49]. $\mathrm{ACC}(i)$ for given task t_i is defined as

$$\mathrm{ACC}(i) = \max \forall k \in S \left(\frac{\sum_{j=1}^{m} \sum_{y=1}^{m} C_{jy} \times D_{ik}}{m^2} \right), \tag{2}$$

where: $\mathrm{ACC}(i)$ is the maximum of the average communication cost of the task t_i with all successors "$\forall k$"; C_{jy} is the cost of transferring data from the processor P_j to processor P_y; D_{ik} is the amount of data needed to be transferred from the parent task t_i to its successors $k \in S$ where S is the set of successors of task t_i.

P_{v} is evaluated from the U_{v} vector and the type of scheduling unit to determine the priority of tasks. U_{v} is a binary vector; if $U_{\mathrm{v}i} = 0$ then t_i is a normal task, else if $U_{\mathrm{v}i} = 1$ then t_i is an urgent task. P_{v} is initialized first with zeros, if t_i is an independent task or an entry task, and $U_{\mathrm{v}i} = 1$, then $P_{\mathrm{v}i}$ will be incremented by one. If the t_i is an exit task or in the intermediate layer of DAGs and $U_{\mathrm{v}i} = 1$, then P_{v} of t_i and P_{v} of its parents will be incremented until it reaches the entry task of the DAG.

A normalization process will be applied to merge the three different attributes into one formula with a certain range and convert the dimensional data to non-

dimensional data [41]. The linear conversion formula is used to get the normalized value as

$$Z(i) = \frac{X(i) - \text{Minvalue}}{\text{Maxvalue} - \text{Minvalue}} \quad (3)$$

where: $X(i)$ is the attribute of a given task t_i; $Z(i)$ is the normalized value of $X(i)$; Minvalue, Maxvalue are the minimum and maximum values for all tasks to a given attribute.

According to the normalized attributes NAET, NACC, and NP_v, the ranking value of each task will be calculated as in [43],

$$\text{Rank}(i) = \text{NAET}(i) + \text{NACC}(i) + \text{NP}_\text{v}(i), \quad (4)$$

where: $\text{Rank}(i)$ is the ranking value of the task t_i; $\text{NAET}(i)$ and $\text{NACC}(i)$, and $\text{NP}_\text{v}(i)$ are the normalized attributes of $\text{AET}(i)$, $\text{ACC}(i)$, and $P_\text{v}(i)$, respectively, for a given task t_i.

The second phase. First, create a list that includes urgent independent tasks and parent tasks, then sort them based on their ranking value. The task with a high rank will be scheduled first in the processor with the earliest finished time and recorded in the Mapping List (ML) matrix. If the scheduled task has successors, its successors will be added to the list after the scheduled task is finished and the required data is transferred. Then, apply the sorting process again after adding the successors to the list, and get the next high-priority task to be scheduled. Otherwise, if the scheduled task is independent, we only get the next high-priority task to be scheduled. After finishing all urgent independent tasks, the parent tasks and their successors, the normal independent tasks with low priority will remain. To schedule normal independent tasks, we first need to find gaps of free time with processors in ML from Equation (5). These gaps are formed due to the waiting time for transferring data between processors. If gaps are found, the normal tasks will be sorted in descending order to start with long tasks first, then check for each gap greater than or equal to the execution time of normal tasks. If that condition is met, the ML will be updated by adding the chosen task to be scheduled in the matched gap. After checking all gaps, if there are remaining normal tasks, then we will start to schedule the task with the minimum execution time first for all remaining tasks (use the Min-Min Algorithm),

$$\text{FreeTime}(j) = \forall i \in R(\text{ST}(i) - \text{FT}(i-1)), \quad (5)$$

where: $\text{FreeTime}(j)$ is the gap found in processor j; $\text{ST}(i)$ is the start time of task t_i; $\text{FT}(i-1)$ is the finished time of the previous task t_{i-1}; both tasks t_i, t_{i-1} are scheduled in the processor P_j, while $[t_i, t_{i-1}] \in R$; R is the set of tasks scheduled in processor j. If the start of the task t_i is equal to the finish of the task t_{i-1}, then there is no gap between the two scheduled tasks t_i. Otherwise, it will check whether any remaining tasks can be executed in that gap.

The proposed algorithm can be applied to different types of tasks. If all incoming tasks are independent (batch mode), then the proposed algorithm will give the urgent tasks the highest priority in scheduling. Then it schedules the tasks with the minimum execution time first. On the other hand, if all incoming tasks have precedence constraints (dependency mode), then it will cluster the parent tasks first and start with the urgent task in that cluster. After the execution of each task, it will add its successors to the cluster and sort them based on the ranking value, and then

it will choose the next urgent task. This leads to executing tasks in the critical path first, as these tasks have urgent priority. The detailed steps of the proposed algorithm are described in Algorithm 1.

Algorithm 1. Hybrid List Scheduling Algorithm (HLSA)

Initialize

Task(t_i) with number *n*, Processor(P_j)with number *m*, Expected Execution Matrix(EMM), Data Transfer Matrix(DTM), Communication Cost Matrix (CM), Urgent vector (U_v)

Step 1. **For** each $t_i \in$ Task (*n*)

Step 2. Calculate the three attributes: Average Execution Time AET(i), Average Communication Cost ACC(i), Priority Vector (P_{vi}) of t_i according to Equations (1), (2)

Step 3. Normalize three attributes AET(i), ACC(i), P_{vi} to NAET(i), NACC(i), NP_{vi} according to the Equation (3)

Step 4. Compute the Ranking value of t_i according to the Equation (4)

Step 5. End For

Step 6. G0= {}

Step 7. G1= {}

Step 8. For all $t_i \in$ Task (*n*)

Step 9. If

$[(\sum_{k=1}^{n} D_{ki} == 0)\&\&(\sum_{k=1}^{n} D_{ik} > 0)]||[(\sum_{k=1}^{n} D_{ki} == 0)\&\&(\sum_{k=1}^{n} D_{ik} == 0)\&\&(P_{vi} > 0)]$

Step 10. Add t_i to G0

Step 11. Else

Step 12. Add t_i to G1

Step 13. End if

Step 14. End for

Step 15. While G0 is not empty

Step 16. Sort tasks in G0 descending based on their ranking value

Step 17. Get first task t_i in G0 to assign to P_j with the earliest Finished Time (FT)

Step 18. Record in ML: task t_i, processor P_j, execution time of t_i on P_j, W_{ij}, Start execution Time ST(i), and Finished execution Time FT(i)

Step 19. Delete t_i from G0

Step 20. if $(\sum_{k=1}^{n} D_{ik} == 0)$

Step 21. Go to step 17

Step 22. Else

Step 23. Add successors of t_i after t_i finishes execution and transfer the required data needed between processors

Step 24. Go to step 16

Step 25. End if

Step 26. End while

Step 27. While G1 is not empty

Step 28. For each P_j in ML

Step 29. Get freeTime_j according to the Equation (5)

Step 30. If (freeTime$_j > W_{ij}$)
Step 31. Sort tasks in G1 descending based on W_{ij}
Step 32. If (freeTime$_j > W_{ij}$)
Step 33. Record t_i, P_j, W_{ij}, ST$_i$, and FT$_i$ in ML
Step 34. Delete t_i from G1
Step 35. End if
Step 36. End if
Step 37. End for
Step 38. Get minimum W_{ij} for all tasks in G1
Step 39. Record t_i, P_j, W_{ij}, ST$_i$, and FT$_i$ in ML
Step 40. Delete t_i from G1
Step 41. End while
Step 42. Return ML

The objective of the proposed algorithm is to get the best mapping between a set of tasks (dependent and independent tasks) t_i on available heterogeneous processors P_j to minimize the execution time and the overall cost and to achieve low latency for urgent tasks in sensitive applications. Also, HLSA aims to achieve the highest utilization of the exploitation of gaps in processors.

4. Performance metrics

Three performance metrics are evaluated for the proposed HLSA Algorithm and compared to the four algorithms HEFT, Min-Min, TS-QoS, and the improved list-based task scheduling algorithms.

1. Makespan is the duration of the execution time of all tasks and workflows [29]. It is also known as the finished execution time for the last executed task [31]. If all the incoming workflows are represented by one DAG, then the makespan will be defined as the finished execution time of the exit task [29],

$$\text{Makespan} = \text{Max}\{\text{FT}_i \ \forall i \in N\} - \text{Min}\{\text{ST}_i \ \forall i \in N\}, \tag{6}$$

where FT$_i$, ST$_i$ are the finished and the start execution time of t_i in ML, respectively.

2. Total cost is the summation of computation cost and communication cost. The computation cost is the cost of executing the given task t_i in processor P_j [29, 31, 32]. The communication cost is the cost of transferring data between two tasks t_i and t_k having precedence constraints between them [32]. The communication cost will be neglected if two tasks t_i and t_k are independent, or they are mapped to the same processor,

$$\text{ComputationCost}(t_i, P_j) = \text{CP}(i,j) = U_j \times (\text{FT}_i - \text{ST}_i) = U_j \times W_{i,j}, \tag{7}$$

$$\text{TotalComputationCost} = \text{CPtotal} = \sum_{i=1}^{n} \sum_{j=1}^{m} \text{CP}_{i,j} \times X_{i,j}, \tag{8}$$

$$\text{CommunicationCost}\left(\left(t_i, P_j\right), \left(t_k, P_y\right)\right) = \begin{cases} \text{CM}_{i,k}(j,y) = C(j,y) \times D_{ik}, & j \neq y \\ \text{CM}_{i,k}(j,y) = 0. & j = y \end{cases}, \tag{9}$$

$$\text{TotalComputationCost} = \text{CMtotal} = \sum_{i=1}^{n} \sum_{k=1}^{n} \text{CM}_{i,k}(j,y), \tag{10}$$

$$\text{TotalCost} = \text{CPtotal} + \text{CMtotal}, \tag{11}$$

where: U_j is the unit price (execution cost) of the processor P_j that executes task t_i; FT_i, $\mathrm{ST}_{i,}$ and $W_{i,j}$ are the finished execution time of t_i, start execution time of $t_{i,}$ and execution time of t_i on P_j in ML, respectively; $X_{i,j}$ is a binary variable, $X_{i,j}$={0, 1}; if task t_i is scheduled on the processor $P_{j,}$ then $X_{i,j} = 1$, otherwise, $X_{i,j} = 0$; $C_{j,y}$ is the cost of transferring data between two processors P_j and P_y in the CM matrix; $D_{i,k}$ is the amount of data needed to be transferred from t_i mapped to processor P_j to t_k mapped to P_y in the DTM matrix.

3. Average latency for urgent tasks is measured as a ratio of the total waiting time for urgent tasks belonging to sensitive applications to all the incoming urgent tasks [41, 50],

$$\mathrm{Latency}(t_i) = L_i = \mathrm{ST}_i - \mathrm{AT}_i, \tag{12}$$

$$\mathrm{AverageLatencyUrgentTasks} = \frac{\sum_{i=1}^{n} U_{\mathrm{v}i} \times L_i}{\sum_{i=1}^{n} U_{\mathrm{v}i}}, \tag{13}$$

where ST_i, AT_i are the start execution time and the arrival time of t_i, respectively; $U_{\mathrm{v}i}$ is a binary variable in an urgent vector U_{v}, $U_{\mathrm{v}i}$={0, 1}; if task t_i is urgent, then $U_{\mathrm{v}i} = 1$, otherwise $U_{\mathrm{v}i} = 0$. The nominator represents the summation of the latency of only urgent tasks. The summation in the dominator represents the number of urgent tasks in all incoming tasks.

5. Simulation environment

The proposed HLSA Algorithm, the old algorithms HEFT, Min-Min, TS-QoS, and the improved list-based task scheduling algorithms were simulated using the programming language Java. The type of incoming tasks is varied between independent, dependency workflows (DAGs), and a combination of both. DAGs workflow is generated randomly [47]. Simulation programs were applied using different numbers of tasks executed on 25 processors. All parameters used in the simulation environment are explained in Table 2. To achieve accurate results, each point in every chart is the average of ten simulation runs.

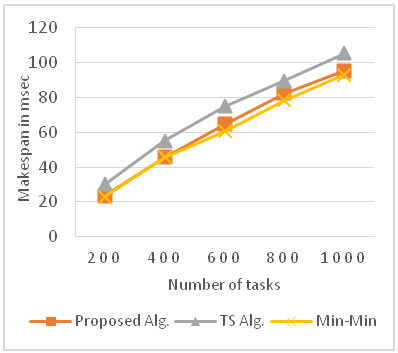
Table 2. Simulation parameters and values

Simulation parameters	Values	
n is the number of tasks [41]	200, 400, 600, 800, 1000	
m is the number of processors [41]	25	
W_{ij} is the expected execution time of t_i in P_j in the Expected Execution Matrix (EEM) [29, 43]	Generated randomly	
D_{ik} is the amount of data needed to be transferred between processors in the Data Transfer Matrix (DTM) [43, 47]	Independent tasks	Dependency workflow
	$\left(\sum_{k=1}^{n} D_{ki} = 0\right) \&\& \left(\sum_{k=1}^{n} D_{ik} = 0\right)$	Generated randomly
C_{jy} is the cost of transferring between processors in the cost communication Cost Matrix (CM) [47]	Generate randomly while taking into consideration that CM is a symmetric matrix with a zero diagonal	
$U_{\mathrm{v}i}$ is a binary number in an urgent vector (U_{v}) [41]	Urgent task	Normal task
	$U_{\mathrm{v}i}$= 1	$U_{\mathrm{v}i}$= 0
U_j is the unit cost of execution t_i in P_j [29]	Constant number	
X_{ij} is a binary number [31]	t_i execute in P_j	t_i not execute in P_j
	X_{ij}= 1	X_{ij}= 0

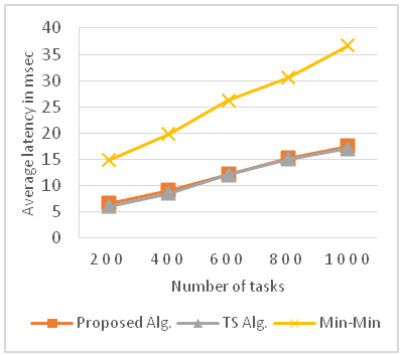
6. Performance evaluation

Three cases were examined to demonstrate that the proposed algorithm, HLSA, can be applied to heterogeneous scheduling units without restricting the types of incoming tasks. In the first case, all incoming scheduling units will be independent tasks; the HLSA will be compared with the Min-Min and TS-QoS algorithms. In the second case, all incoming scheduling units will be dependent workflows with precedence constraints, and the HLSA will be compared with HEFT and improved list-based task scheduling algorithms. In the third case, the incoming scheduling units will combine independent and dependent workflows. It will compare the performance of HLSA with different percentages of the combination.

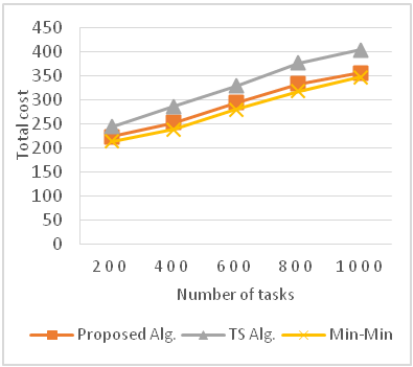
Case 1. If all the incoming tasks are independent, then the proposed algorithm HLSA, will be compared to the Min-Min and TS-QoS algorithms. As all the incoming tasks are independent, no data needs to be transferred between processors, so all the elements in the Data Transfer Matrix (DTM) are equal to zero. Also, the communication Cost Matrix (CM) will not be needed. The Expected Execution Matrix (EEM) and urgent vector (U_v) will be generated randomly.



(a) Makespan



(b) Average latency of urgent tasks



(c) Total cost

Fig. 1. Case 1. Comparison between proposed algorithm, Min-Min, and TS-QoS algorithms

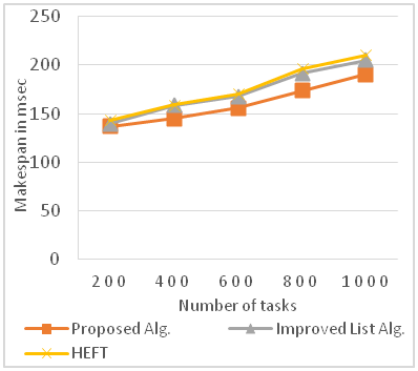
Makespan. Fig. 1 shows that the proposed HLSA and Min-Min algorithms get almost the same minimum makespan. However, the TS-QoS algorithm achieves the highest one. This is because the Min-Min algorithm searches in the whole EEM for

the task with minimum execution time, but HLSA searches first for the tasks with high priority and then works as Min-Min with the remaining tasks. The TS-QoS Algorithm schedules the tasks based on their priority, so there is no search for tasks with minimum execution time in EEM.

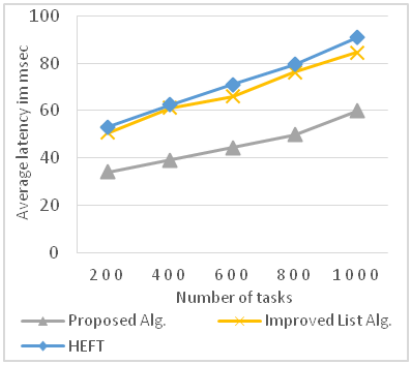
Total cost. The three algorithms achieve the same performance in cost as makespan. In that case, the total cost is equal to the computation cost only, as all tasks are independent, and the CM matrix equals zero. Thus, the total cost will depend on execution time as the makespan.

Latency for urgent tasks. The proposed HLSA and TS-QoS algorithms achieve minimal latency regarding urgent tasks compared to the Min-Min algorithm. HLSA and TS-QoS prioritize scheduling urgent tasks first. On the other hand, the Min-Min algorithm gives the highest priority to tasks with minimal execution time.

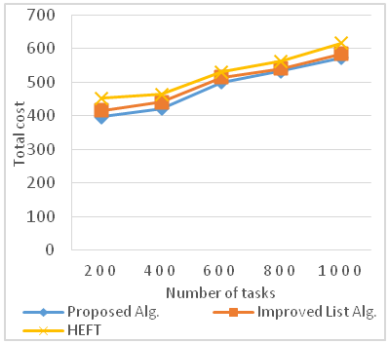
Case 2. If all the incoming tasks are dependency workflow, then the proposed HLSA Algorithm will be compared to HEFT and the improved list-based task scheduling algorithm. All the incoming tasks are dependent workflows represented by a DAG and generated randomly. Also, the Data Transfer Matrix (DTM), the communication Cost Matrix (CM), the Expected Execution Matrix (EEM), and the Urgent vector (U_v) will be generated randomly.



(a) Makespan



(b) Average latency of urgent tasks



(c)Total cost

Fig. 2. Case 2. Comparison between proposed algorithm, HEFT, and improved list-based algorithms

Makespan. Fig. 2 shows that the proposed algorithm HLSA achieves better performance in makespan, followed by the improved list-based algorithm compared

to HEFT because of the scheme used to evaluate the ranking value in each algorithm. In HLSA, the ranking value of each task increases with the number of successors to each task. That leads to lower waiting time for tasks till their parents are executed and the amount of precedence data is transferred.

Total cost. The proposed algorithm, HLSA, achieves slightly better performance in total cost, followed by an improved list-based algorithm compared to HEFT. In that case, the total cost is the summation of the computation and communication costs. HLSA gets a minimum makespan, followed by the improved list-based algorithm, followed by HEFT, so the computation cost of HLSA will be the minimum, followed by the improved list-based algorithm, followed by HEFT. The three algorithms aim to get minimum communication costs, as it is considered when calculating the ranking value for each task.

Latency to urgent tasks. The proposed algorithm, HLSA, achieves minimum latency to urgent tasks compared to HEFT and the improved list-based algorithm because HLSA considers the priority of the tasks when evaluating the ranking value for each task, while HEFT and the improved list-based algorithm do not.

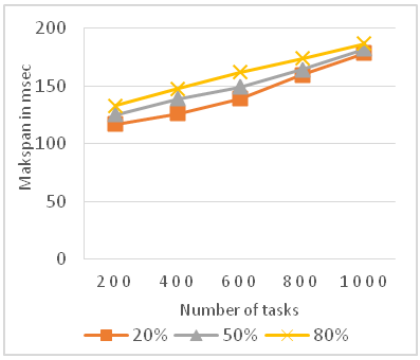
Case 3. The incoming tasks are a combination of both independent and dependent workflows. 20%, 50% and 80% of tasks are dependent on workflow and are used to evaluate the proposed algorithm according to these different percentages.

Makespan and Latency. Fig. 3 shows that the proposed algorithm HLSA achieves better performance in makespan and latency with workflow consist of (20% of dependent workflow and 80% of independent tasks) compared to workflow consist of (50% of dependent workflow and 50% independent tasks, and 80% of dependent workflow and 20% independent tasks). The increase in dependency constraints justifies that, as it increases the waiting time between tasks until their parents are executed, and the delay in transferring precedence data between tasks.

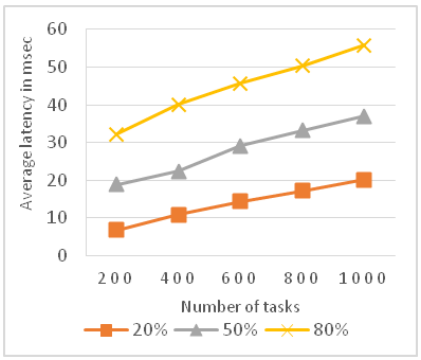
Total cost. Also, the HLSA with (20% of dependent workflow and 80% of independent tasks) achieves minimum total cost as it has a minimum number of tasks that need precedence data before starting execution, so the communication cost will be calculated only for 20% of tasks. By increasing the percentage of dependent workflow, the communication cost will increase, as well as the total cost.

Exploitation of gaps. Using free scheduling gaps will increase with the increase in the number of independent tasks that fill up gaps. When using 20% of the dependent workflow, the percentage of the used time is 100%, then it decreases with the decrease of independent tasks, and vice versa.

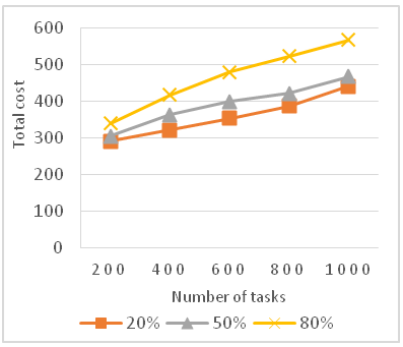
Table 3 summarizes the results obtained in the previous cases. It illustrates the percentage of improvement that HLSA achieves compared to Min-Min and TS-QoS algorithms in case all tasks are independent, and compared to HEFT and improved list-based algorithms in case all tasks are dependent. Also, it illustrates the changes accrued in the used metrics when the incoming tasks combine independent and dependent workflows with different percentages in Case 3.



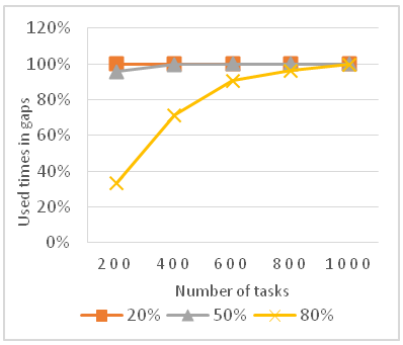
(a) Makespan



(b) Average latency of urgent tasks



(c) Total cost



(d) Percentage of free time

Fig. 3. Case 3. Comparison when applying the proposed algorithm with different percentages of dependent workflow (20%, 50%, and 80%)

As shown in Table 3, in the first case, all the incoming tasks are independent, and the proposed algorithm, HLSA, was compared to the Min-Min and TS-QoS algorithms. The proposed algorithm achieves almost the same performance in makespan and total cost, with differences of 4% and 5% of using the Min-Min Algorithm, while achieving better performance with percentages 14% and 16% of the TS-QoS Algorithm. The proposed algorithm achieves almost the same performance in latency to urgent tasks with a difference 3% of using the TS-QoS Algorithm, while achieving better performance with a percentage 45% of using the Min-Min Algorithm.

In the second case, all the incoming tasks depend on each other. It was compared to HEFT and the improved list-based algorithm. The proposed algorithm performs better in makespan and total cost, with a difference of 7%, 9% in HEFT, and 3%, 6% in the improved list-based algorithm. In comparison, it performs better latency to urgent tasks with a percentage of 37% for the HEFT algorithm and 32% for the improved list-based algorithm.

In the third case, the changes accrued in performance metrics when increasing the percentage of precedence constraints between tasks from 0% (all tasks are independent) to 20%, 50%, 80% and 100% (all tasks are dependent on each other) in the proposed algorithm are declared. The average of changes in makespan and latency increased with increasing precedence constraints because of the increasing

waiting time between tasks until their parents execute and the transfer of precedence data. Also, the average of changes in total cost increased due to an increase in communication cost that was calculated between dependency tasks. On the other hand, the proposed algorithm achieves low efficiency in utilizing gaps of free time by increasing dependency constraints.

Table 3. Summary of the results

Case 1. All tasks are independent					
Performance metrics	Percentage of differences between proposed HLSA Algorithm with				
	Min-Min Algorithm		TS-QoS Algorithm		
Makespan	More by 4%		More by 14%		
Total cost	More by 5%		More by 16%		
Latency	More by 45%		More by 3%		
Case2. All tasks are dependent on each other					
Performance metrics	Percentage of differences between proposed HLSA Algorithm with				
	HEFT Algorithm		Improved List-Based Algorithm		
Makespan	More by 7%		More by 3%		
Total cost	More by 9%		More by 6%		
Latency	More by 37%		More by 32%		
Case 3. Combination of independent tasks and dependent workflow					
Performance metrics	Average increase in performance metrics when increasing dependency percentage from				
	0% to 20%	20% to 50%	50% to 80%	20% to 80%	80% to100%
Makespan (ms)	13	6	8	14	10
Total cost	42	33	47	75	31
Latency (ms)	2	15	17	31	16
Used time of gaps	-	1	21	22	80

7. Conclusion and future work

This paper proposes a hybrid list scheduling algorithm, HLSA, that aims to work with different types of scheduling units to avoid the need to add restrictions on the types of incoming tasks and schedule them in heterogeneous resources to get minimum execution time, total cost, and low latency for tasks that belong to sensitive IoT applications. To evaluate its performance, three cases were studied using simulation: all tasks are independent, all tasks are dependent on each other, and a combination of both. From the results of the three cases, we can see that the proposed algorithm HLSA gets minimum latency to urgent tasks that are needed in sensitive time applications in fog computing and utilizes gaps of free time in the scheduling timeline for each processor. It also achieves minimum makespan and total cost compared with other algorithms.

HLSA can work with different types of incoming tasks and available resources that are defined before starting scheduling. Therefore, HLSA cannot be applied to dynamic scheduling as it is considered a static scheduling algorithm. The future work aims to apply this algorithm to dynamic scheduling.

References

1. A l a d w a n i, T. Scheduling IoT Healthcare Tasks in Fog Computing Based on Their Importance. – In: Proc. of International Learning and Technology Conference, 2019, pp. 560-569. DOI: 10.1016/j.procs.2019.12.138
2. D o l u i, K., S. K. D a t t a. Comparison of Edge Computing Implementations: Fog Computing, Cloudlet, and Mobile Edge Computing. – In: Proc. of IEEE Global Internet of Things Summit Conference (GIoTS'17), 2017. DOI: 10.1109/GIOTS.2017.8016213.
3. M a t r o u k, K., K. A l a t o u n. Scheduling Algorithms in Fog Computing: A Survey. – International Journal of Networked and Distributed Computing, Vol. **9**, 2021, pp. 59-74. DOI: 10.2991/ijndc.k.210111.001.
4. K u m a r, K. D., E. U m a m a h e s w a r i. HPCWMF: A Hybrid Predictive Cloud Workload Management Framework Using Improved LSTM Neural Network. – Cybernetics and Information Technologies, Vol. **20**, 2020, No 4, pp. 55-73.
5. M a d h u m a l a, B. R., H. T i w a r i, D. V e r m a. Virtual Machine Placement Using Energy-Efficient Particle Swarm Optimization in Cloud Datacenter. – Cybernetics and Information Technologies, Vol. **21**, 2021, No 1, pp. 62-72.
6. S a t v e e r, M., S. A s w a l. VM Consolidation Plan for Improving the Energy Efficiency of Cloud. – Cybernetics and Information Technologies, Vol. **21**, 2021, No 3, pp. 145-159.
7. B o n o m i, F., R. M i l i t o, P. N a t a r a j a n, J. Z h u. Fog Computing: A Platform for Internet of Things and Analytics. – Big Data and Internet of Things: A Roadmap for Smart Environments. – In: Studies in Computational Intelligence. Vol. **546**. Springer, 2014, pp. 169-186. DOI: 10.1007/978-3-319-05029-4_7.
8. M o h a m m a d, S., N. R a j e s w a r i. Overview of Cloud Computing and Its Types. – SSNR Electronic Journal, Vol. **6**, 2019, pp. 61-67.
   http://www.jetir.org/papers/JETIRAT06008.pdf
9. A l z o u b i, Y. I., A. A l j a a f r e h. Blockchain-Fog Computing Integration Applications: A Systematic Review. – Cybernetics and Information Technologies, Vol. **23**, 2023, No 1, pp. 3-37.
10. B h a r g a v i, K., S. B a b u, S. G. S h i v a. Type-2-Soft-Set-Based Uncertainty Aware Task Offloading Framework for Fog Computing Using Apprenticeship Learning. – Cybernetics and Information Technologies, Vol. **23**, 2023, No 1, pp. 38-58.
11. C h u, H., S. Y a n g, P. P i l l a i, Y. C h e n. Scheduling in Visual Fog Computing: NP-Completeness and Practical Efficient Solutions. – In: Proc. of AAAI Conference on Artificial Intelligence, Vol. **32**, 2018. DOI: 10.1609/aaai.v32i1.12080.
12. M. E. A Survey of Various Scheduling Algorithms in a Cloud Computing Environment. – Journal of Emerging Technologies and Innovative Research (JETIR), Vol. **2**, 2013, pp. 131-135. DOI:10.15623/IJRET.2013.0202008.
13. B h u t t o, A., A. A. C h a n d i o, K. K. L u h a n o, I. A. K o r e j o. Analysis of Energy and Network Cost Effectiveness of Scheduling Strategies in Datacentre. – Cybernetics and Information Technologies, Vol. **23**, 2023, No 3, pp. 56-69.
14. V a r s h n e y, P., Y. S i m m h a n. Characterizing Application Scheduling on Edge, Fog, and Cloud Computing Resources. – Software Practice and Experience Journal, Vol. **50**, 2019. DOI: 10.1002/spe.2699.
15. K h o d a d a d i, F., A. V a h i d, R. B u y y a. Internet of Things: An Overview. 2017. DOI: 10.4550/arXiv.1703.06409.
16. S l o w, E., T. T i r o p a n i s, W. H a l l. Analytics for the Internet of Things: A Survey. – ACM Computing Surveys, Vol. **1**, 2018, No 1. DOI: 10.1145/3204947.
17. D a n g, L. M., M. J. P i r a n, D. H a n, K. M i n, H. M o o n. A Survey on the Internet of Things and Cloud Computing for Healthcare. – Journal of Electronics, Vol. **8**, 2019. DOI: 10.3390/electronics8070768.
18. Y u, W., F. L i a n g, X. H e, W. G. H a t c h e r, G. L u, J. L i n, X. Y a n g. A Survey on Edge Computing for the Internet of Things. – IEEE Access Journal, Vol. **6**, 2018. DOI: 10.1109/ACCESS.2017.2778504.

19. B a r o t, V., V. K a p a d i a, S. P a n d y a. QoS-Enabled IoT-Based Low-Cost Air Quality Monitoring System with Power Consumption Optimization. – Cybernetics and Information Technologies, Vol. **20**, 2020, No 2, pp. 122-140.
20. S u d h a, K. S., N. J e y a n t h i. A Review on Privacy Requirements and Application Layer Security in Internet of Things (IoT). – Cybernetics and Information Technologies, Vol. **21**, 2021, No 3, pp. 50-72.
21. S a r i t h a, S a r a s v a t h i V. Reliability Analysis of an IoT-Based Air Pollution Monitoring System Using Machine Learning Algorithm-BDBN. – Cybernetics and Information Technologies, Vol. **23**, 2023, No 4, pp. 233-250.
22. T s a i, C., W. H u a n g, M. C h i a n g, M. C h i a n g, C. Y a n g. A Hyper-Heuristic Scheduling Algorithm for Cloud. – IEEE Transactions on Cloud Computing, Vol. **2**, 2014, pp. 236-250. DOI: 10.1109/TCC.2014.2315797.
23. R e z a, M., V. K h a j e h v a n d, A. M a s o u d, E. A k b a r i. A Task Scheduling Approach in Fog Computing: A Systematic Review. – International Journal of Communication Systemic, Vol. **33**, 2020. DOI: 10.1002/dac.4583.
24. K h a n, A., A. A b b a s, H. A. K h a t t a k, F. R e h m a n, I. U d D i n, S. A l i. Effective Task Scheduling in Critical Fog Applications. – In: Scientific Programming. 2022. DOI: 10.1155/2022/9208066.
25. F u, X., B. T a n g, F. G u o, L. K a n g. Priority and Dependency-Based DAG Tasks Offloading in Fog/Edge Collaborative Environment. – In: Proc. of 24th IEEE International Conference on Computer Supported Cooperative Work in Design, Dalian, China, 2021. DOI: 10.1109/CSCWD49262.2021.9437784.
26. P e r i a s a m y, P., R. U j w a l a, K. S r i k a r, Y. V. D. S a i, K. S. P r e e t h a, D. S u m a t h i, M. S. S a y e e d. ERAM-EE: Efficient Resource Allocation and Management Strategies with Energy Efficiency under Fog-Internet of Things Environments. – Connection Science Journal, Vol. **36**, 2024. DOI: 10.1080/09540091.2024.2350755.
27. F i s t e r, I., X. Y a n g, I. F i s t e r, J. B r e s t, D. F i s t e r. Brief Review of Nature-Inspired Algorithms for Optimization. – Elektrotehniski Vestnik/Electrotechnical Review Journal, Vol. **3**, 2013. DOI: 10.48550/arXiv.1307.4186.
28. S a l e m, A. H., G. A l-G a p h a r i. Meta-Heuristic Algorithms for Resource Allocation in Fog Computing. – International Journal for Modern Trends in Science and Technology, Vol. **8**, 2022, pp. 134-143. DOI: 10.46501/IJMTST0802022.
29. X u, R., Y. W a n g, Y. C h e n g, Y. Z h u, Y. X i e, A. S a d i q, D. Y u a n. Improved Particle Swarm Optimization-Based Workflow Scheduling in Cloud-Fog Environment. – Springer Nature Switzerland, Vol. **342**, 2019, pp. 337-347. DOI: 10.1007/978-3-030-11641-5_27.
30. L i n, Y., C. C h e n g, F. X i a o, K. A l s u b h i, H. M o a i t e q. A DAG-Based Cloud-Fog Layer Architecture for Distributed Energy Management in Smart Power Grids in the Presence of PHEVs. – Journal of Sustainable Cities and Society, Vol. **75**, 2021. DOI: 10.1016/j.scs.2021.103335.
31. A b o h a m a m a, A. S., A. E l-G h a m r y, E. H a m o u d a. Real-Time Task Scheduling Algorithm for IoT-Based Applications in the Cloud-Fog Environment. – Journal of Network and Systems Management, Vol. **30**, 2022. DOI: 10.1007/s10922-022-09664-6.
32. M o k n i, M., S. Y a s s a, J. E d d i n e, R. C h e l o u a h, M. N a z i h. Cooperative Agents-Based Approach for Workflow Scheduling on Fog Cloud Computing. – Springer Journal of Ambient Intelligence and Humanized Computing, Vol. **13**, 2021. DOI: 10.1007/s12652-021-03187-9.
33. J a n g u, N., Z. R a z a. Improved Jellyfish Algorithm-Based Multi-Aspect Task Scheduling Model for IoT Tasks over Fog-Integrated Cloud Environment. – Journal of Cloud Computing: Advances, Systems and Applications, Vol. **11**, 2022. DOI: 10.1186/s13677-022-00376-5.
34. K u m a r, M. S., G. R. K a r r i. EEOA: Cost and Energy Efficient Task Scheduling in a Cloud-Fog Framework. – Journal of Sensors, Vol. **23**, 2023. DOI: 10.3390/s23052445.
35. A m e e n a, B., L. R a m a s a m y. Drawer Cosine Optimization Enabled Task Offloading in Fog Computing. – Expert Systems with Applications Journal, Vol. **259**, 2025. DOI: 10.1016/j.eswa.2024.125212.
36. W a n g, J., D. L i. Task Scheduling Based on a Hybrid Heuristic Algorithm for Smart Production Line with Fog Computing. – Journal of Sensors, Vol. **19**, 2019. DOI: 10.3390/s19051023.

37. K a b i r z a d e h, S., D. R a h b a r i, M. N i c k r a y. A Hyper-Heuristic Algorithm for Scheduling of Fog Networks. – In: Proc. of IEEE Open Innovations Association FRUCT Conference, Vol. **562**, Finland, 2017, pp. 148-155. DOI: 10.23919/FRUCT.2017.8250177.
38. K r i v i c, P., M. K u s e k, I. C a v r a k, P. S k o c. Dynamic Scheduling of Contextually Categorized Internet of Things Services in a Fog Computing Environment. – Journal of Sensors, Vol. **22**, 2022. DOI: 10.3390/s22020465**.**
39. M o h a m m a d, A., R. M a h m o u d, N. J a m a l, A. A l S m a d i, M. A l s h a b a n a h, D. A l r a j h i, H. A l k h a l d i, M. K. A l s m a d i. Fog Computing Scheduling Algorithm for a Smart City. – International Journal of Electrical and Computer Engineering (IJECE), Vol. **11**, 2021, pp. 2219-2228. DOI: 10.11591/ijece.v11i3.pp2219-2228.
40. Y u, J., R. B u y y a, K. R a m a m o h a n a r a o. Workflow Scheduling Algorithms for Grid Computing, Metaheuristics for Scheduling in Distributed Computing Environments. – Springer, Vol. **146**, 2008, pp. 173-214. DOI: 10.1007/978-3-540-69277-5_7.
41. W u, X., M. D e n g, R. Z h a n g, B. Z e n g, S. Z h o u. A Task Scheduling Algorithm Based on QOS-Driven in Cloud Computing. – In: Proc. of International Conference on Information Technology and Quantitative Management, China, Vol. **17**, 2013, pp. 1162-1169. DOI: 10.1016/j.procs.2013.05.148.
42. P h a m, X., N. D o a n, N. D a o, N. Q u a n g, E. H u h. A Cost and Performance-Effective Approach for Task Scheduling Based on Collaboration between Cloud and Fog Computing. – International Journal of Distributed Sensor Networks, Vol. **13**, 2017. DOI: 10.1177/1550147717742073.
43. M a d h u r a, R., L. E l i z a b e t h, R. U t h a r i a r a j. An Improved List-Based Task Scheduling Algorithm for a Fog Computing Environment. – Computing Journal, Vol. **103**, 2021. DOI: 10.1007/s00607-021-00935-9.
44. T a r i q, R., F. A a d i l, M. F. M a l i k, S. E j a z, M. U. K h a n, M. F. K h a n. Directed Acyclic Graph-Based Task Scheduling Algorithm for Heterogeneous Systems. – In: Proc. of Intelligent Systems Conference, London, Vol. **2**, 2019, pp. 936-947. DOI: 10.1007/978-3-030-01057-7_69.
45. G o e l, H., N. C h a m o l i. Job Scheduling Algorithms in Cloud Computing: A Survey. – International Journal of Computer Applications, Vol. **95**, 2014, No 23, pp.19-22. DOI: 10.5120/16735-6981.
46. V i j a y a, C., P. S r i n i v a s a n. A Hybrid Technique for Server Consolidation in Cloud Computing Environment. – Cybernetics and Information Technologies, Vol. **20**, 2020, No 1, pp. 36-52.
47. S a k e l l a r i o u, R., H. Z h a o. A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems. – In: Proc. of IEEE International Parallel and Distributed Processing Symposium Conference, USA, 2004. DOI: 10.1109/IPDPS.2004.1303065.
48. A h m a d, W., B. A l a m, S. M a l i k. Performance Analysis of List Scheduling Algorithms by Random Synthetic DAGs. – SSRN Electronic Journal, 2019. DOI: 10.2139/ssrn.3349016.
49. Z h a o, H., R. S a k e l l a r i o u. An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm. – In: Proc. of Conference of Parallel Processing. Vol. **2790**. Springer, 2003, pp. 189-194. DOI: 10.1007/978-3-540-45209-6_28.
50. A l i, H. G. E. H., I. A. S a r o i t, A. M. K o t b. Grouped Tasks Scheduling Algorithm Based on QoS in Cloud Computing Network. – Egyptian Informatics Journal, Vol. **18**, 2016. DOI: 10.1016/j.eij.2016.07.002.

Fast-track. Received: 07.07.2025. First Revision: 06.08.2025. Accepted: 12.08.2025