IoT

MDPI

Article

A Greedy Scheduling Approach for Peripheral Mobile Intelligent Systems

Ghassan Fadlallah, Djamal Rebaine and Hamid Mcheick *

Department of Computer Science and Mathematics, University of Quebec at Chicoutimi, Chicoutimi, QC G7H 2B1, Canada; Ghassan-Mustapha.Fadlallah1@uqac.ca (G.F.); Djamal_Rebaine@uqac.ca (D.R.)
* Correspondence: hamid_mcheick@uqac.ca

Abstract: Smart, pervasive devices have recently experienced accelerated technological development in the fields of hardware, software, and wireless connections. The promotion of various kinds of collaborative mobile computing requires an upgrade in network connectivity with wireless technologies, as well as enhanced peer-to-peer communication. Mobile computing also requires appropriate scheduling methods to speed up the implementation and processing of various computing applications by better managing network resources. Scheduling techniques are relevant to the modern architectural models that support the IoT paradigm, particularly smart collaborative mobile computing architectures at the network periphery. In this regard, load-balancing techniques have also become necessary to exploit all the available capabilities and thus the speed of implementation. However, since the problem of scheduling and load-balancing, which we addressed in this study, is known to be NP-hard, the heuristic approach is well justified. We thus designed and validated a greedy scheduling and load-balancing algorithm to improve the utilization of resources. We conducted a comparison study with the longest cloudlet fact processing (LCFP), shortest cloudlet fact processing (SCFP), and Min-Min heuristic algorithms. The choice of those three algorithms is based on the efficiency and simplicity of their mechanisms, as reported in the literature, for allocating tasks to devices. The simulation we conducted showed the superiority of our approach over those algorithms with respect to the overall completion time criterion.

Keywords: IoT; scheduling; makespan; greedy algorithms; pervasive systems; LCFP; SCFP; Min-Min algorithms

Citation: Fadlallah, G.; Rebaine, D.; Mcheick, H. A Greedy Scheduling Approach for Peripheral Mobile Intelligent Systems. *IoT* **2021**, *2*, 249–274. https://doi.org/10.3390/iot2020014

Academic Editor: Luca Bedogni

Received: 1 April 2021
Accepted: 27 April 2021
Published: 30 April 2021

1. Introduction

Continual and accelerated innovation in communications and information technologies has led to an expansion of the modern Internet and its applications and extensions, such as the Internet of things (IoT). The IoT continues to grow rapidly. It has become the basis for the so-called Fourth Industrial Revolution and the digital transformation of business and society [1]. It is an emerging paradigm aimed at providing appropriate and smart systems of objects [2]. Making objects smart may be interpreted in two ways. The first interpretation is about entrusting computing power to an object, making it autonomous by allowing it to make its own choices. The second interpretation, for its part, consists of allowing objects to communicate with the outside world and, if necessary, to communicate with machines that can calculate and make decisions. These machines can be located in cloud centers to limit their economic impact [3]. However, the proliferation of connected objects generates a huge amount of data that is difficult for resource-limited IoT objects to manage. In addition, the exchange and security of this data is a challenge when it is loaded and uploaded between peripheral devices and cloud server centers. In this regard, artificial intelligence (AI) can make a significant contribution to solving this problem, where it can become, along with the IoT, an important achievement [4].

The IoT supports a wide range of distributed systems that are interconnected with various devices such as servers, database centers, and computers. This is in addition to more powerful and compact portable devices such as smart cell phones and personal digital assistants (PDA) [5]. It is obvious to researchers and specialists that the demand for these devices has increased steadily. Unfortunately, technological progress has not always followed this tendency. Therefore, scheduling and load balancing techniques [6–9] have been developed to close this technological gap.

The scheduling process is concerned with the allocation of tasks to devices over time, whatever the environment may be (deterministic, stochastic, or online), with respect to a given criterion. For the sake of clarity, let us recall the definitions of these environments [10]:

1. Deterministic scheduling: the data defining the problem are known in advance;
2. Stochastic scheduling: all or most of the parameters that describe the problem are random variables over known distributions;
3. Online scheduling: all or some of the parameters describing the model are known only at the time the decision has been made.

Improving the scheduling process requires a better understanding of the environment under study and depends on whether it is static (fixed and non-modifiable scheduling) or dynamic (re-evaluated online to respond to changes). Likewise, the environment treatment criteria on a local or distributed level can be classified into different types: (1) hard constraints (the constraints must be strictly respected), (2) soft constraints (there is a flexibility with respect to the constraints), (3) preemptive (tasks may be interrupted before completion), or (4) non-preemptive (tasks are executed without interruption until completion) [11–13].

Scheduling problems occur in various real-world applications, such as industrial production, hoist scheduling, airport control towers, assignment of processes on processors, and data transfer services. Scheduling problems were originally studied in the industrial sector, such as the manufacture and assembly of large generators, and this led to the introduction of critical path analysis (CPA) and the critical path method (CPM) [14,15]. Recently, scheduling problems has emerged as an active field of research in the IoT community [16] and in mobile collaborative computing models via architectural models of cloud, fog, and edge computing [6,17]. As these emerging architectural models of computing are distributed systems, we focused our study on scheduling parallel machine models.

The development of mobile devices and communication technologies has contributed to their growing utilization in collaborative applications that are complex and resource-intensive [18–20]. This has led to innovative designs in mobile collaborative computing and interaction techniques between devices. Among the most important closely related techniques are those for the scheduling and load balancing of tasks. Indeed, the scheduling problem consists of organizing the task execution order on ready devices, while load balancing aims to balance the tasks between devices [11]. As pointed out in Mishra et al. [21], the load balancing task determines where certain applications should be executed; it is used for distributing workloads across computing resources.

In this paper, we focus on establishing collaborative mobile systems based on modern technologies such as Pycom's LOPY4 and direct radio communication by devices to maintain wireless connectivity in different environments, even in the harshest ones [22,23]. In the present study, we develop a new scheduling algorithm based on a greedy approach, which is well suited to these environments in terms of speed, quality, and ease of decision-making at every step [24–26]. This approach enables small mobile devices to process tasks that match their capabilities where they have resource constraints in the IoT paradigm. Therefore, we adopted a strategy that consists of dividing the task and device groups into corresponding subgroups in terms of task size and device capacity. This algorithm aims to ensure load balancing through a technique that chooses among these corresponding subgroups the devices with the longest execution time first to allocate them to the appropriate tasks.

We conducted an experimental study (simulation) to compare, with respect to criteria based on the overall completion times, three existing solutions with our approach to improve the scheduling and load balancing techniques on the available devices of these systems. Let us note that the choice of these three algorithms was mainly based on the simplicity, efficiency, and speed, as well as the mechanism for allocating devices to tasks [27,28]. We believe that the present study provides a path to further research on the development and efficient use of mobile and pervasive computing devices in crucial and urgent situations, such as highly intensive communications that require high bandwidth, smart healthcare, smart cities, harsh cases, and smart cars.

This paper is organized as follows. Section 2 presents the related works. Section 3 introduces greedy algorithms as a solution in our approach. Section 4 describes our contribution to scheduling and load balancing in peripheral autonomous mobile systems. Through an intensive experimental study, Section 5 validates our approach, then analyzes and undertakes a comparison between our solution and three other algorithms. Finally, Section 6 presents our concluding remarks and perspectives.

2. Related Works

In this section, we present a review of the literature on scheduling techniques applied in the IoT field. The algorithms presented in [11] show that the tendencies in this area are evolving more toward approximate (near-optimal) solutions that are generated within a reasonable time. The main reason for these trends is that most scheduling problems are difficult to solve from the computational point of view [29].

Le et al. [30] pointed out that source code optimization is an emerging technique used to reduce energy consumption in parallel with increasing the storage and power capacity of mobile devices. This improvement may be achieved by partitioning tasks into smaller tasks.

The classification of task scheduling algorithms differs depending on the viewpoints of the authors. The algorithms fall into six categories [31], which are presented in Table 1.

Table 1. Classification of task scheduling algorithms.

Categories of Scheduling Algorithms	Description
Immediate scheduling	Direct scheduling of new tasks upon their arrival on VMs.
Batch scheduling of tasks	The tasks are pregrouped into batches before they are sent.
Static scheduling	The strategies of scheduling within this environment are usually based on information known beforehand about the system's global state.
Dynamic scheduling	Does not require current information about the global state of the system. In dynamic scheduling, tasks are distributed in terms of the capacity of the available VMs.
Preemptive scheduling	Portions of a task executed on a resource are resumed later on the same or another resource.
Non-preemptive scheduling	Requires processing the entire task without interruption until its completion on the same resource.

In cloud computing, there are three levels suitable for the task scheduling system [31]:

1. The first level is the set of tasks (cloudlets) to be executed;
2. The second level is the process of appropriately allocating resources to tasks to optimize the use of these resources with respect to the makespan or overall completion times;
3. The third level is the use of a set of virtual machines (VMs) to perform tasks.

In Sharma et al. [32], it is stated that there are several task scheduling techniques available for scheduling tasks in a cloud environment under the following categories:

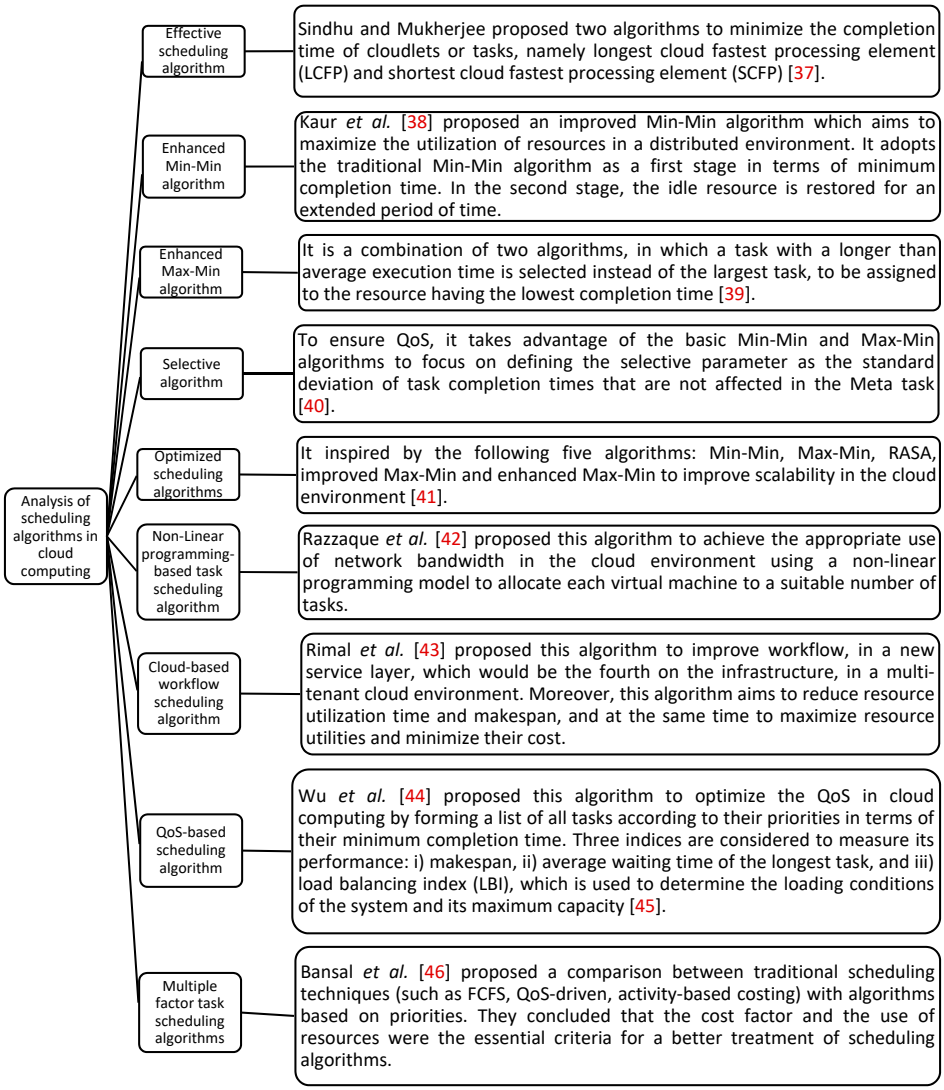
1. Heuristic techniques can be divided into two subcategories. The first one is a traditional technique for scheduling various tasks, such as FCFS, round robin (RR),

and SJF. This approach is simple and imperative, but it always approaches the local optima [33]. The second category uses a random sample to find the optimal or near-optimal solution. Some of its techniques are Min-Min, Max-Min, improved Max-Min [34], and Min-Min based on priorities. These techniques generate better results than traditional approaches [35];

2. Metaheuristic algorithms generally have functionalities that are like aspects of biological science. They are classified into three categories: (1) metaheuristics, such as genetic and transgenic algorithms, based on gene transfer; (2) metaheuristics based on insect behaviors and their interactions, such as ant colony optimization, the firefly algorithm, bee marriage optimization algorithm, and bee colony algorithm; and (3) metaheuristics based on aspects of biological life, such as the tabu search algorithm, simulated annealing algorithm, optimization algorithm for particle swarms, and artificial immune system [36].

The main difference between heuristic and metaheuristic is that the former is problem-specific and generates, step by step, only one solution, while the latter is problem-independent and generates several solutions [32].

Recently, improvements made to scheduling algorithms in cloud computing were discussed and investigated extensively in the literature (see, for example, Sharma et al. [32]). The analysis of scheduling algorithms is summarized in Scheme 1.



Scheme 1. Categories of several task scheduling techniques in the cloud environment [37–46].

The need for the fair sharing of resources in wireless networks increases with both the increased demand for their use and the cost of their equipment as it becomes difficult to install more devices (Sherin et al. [47]). Therefore, the scheduling process becomes the most important factor for improving wireless resource management. Multiple users of the system can access their shared resources efficiently by this scheduling process. These resources are mostly limited when compared with the increasing number of users. Scheduling algorithms offer equitable user access and are essential for ensuring the provision of quality of service (QoS). Let us observe that researchers and practitioners are becoming more concerned with packet scheduling algorithms, which are highly essential for QoS. The design of scheduling algorithms has become more complex due to several factors, including the dynamism of mobile ad hoc networks (MANETs) and vehicular ad hoc networks (VANETs) with frequent topology changes and breakdowns in connectivity. A MANET has many applications, including disaster management, emergency relief, vehicular ad hoc network services (VANETs), war field communications, mobile teleconferencing, and electronic payments. QoS is essential in these real-time applications due to the limited resources and dynamic topology. QoS indicates the service performance level provided by the network to the end user. Depending on the application, it can be based on bandwidth, delay, packet loss, throughput, overhead, jitter, and so on.

One of the key features of ad hoc networks is their capability of operating without a standard infrastructure, and they can be deployed easily for various applications. However, it is difficult to provide QoS for all packets because of the dynamic nature of nodes [48]. Generally, scheduling algorithms manage changes in queuing dynamics and indicate the packets to be submitted from the queue. To ensure the QoS parameters, these algorithms are determined based on network requirements [49]. There are two major problems in multi-hop wireless networks: the packet (or package of information, as it is known in the jargon of computer networks [48]) and the channel scheduling. In this area, scheduling algorithms, generally considered to be non-preemptive, are based on analysis of the QoS parameters, such as the throughput rate, fairness in the network, bounded delay, and jitter. The traditional scheduling approach, which takes advantage of the priority lists, is mainly used in mobile ad hoc networks.

Wireless scheduling algorithms in mobile ad hoc networks are classified according to their application domains into two general classes: packet scheduling and channel-aware scheduling [47], as presented in Tables 2 and 3 and Scheme 2.

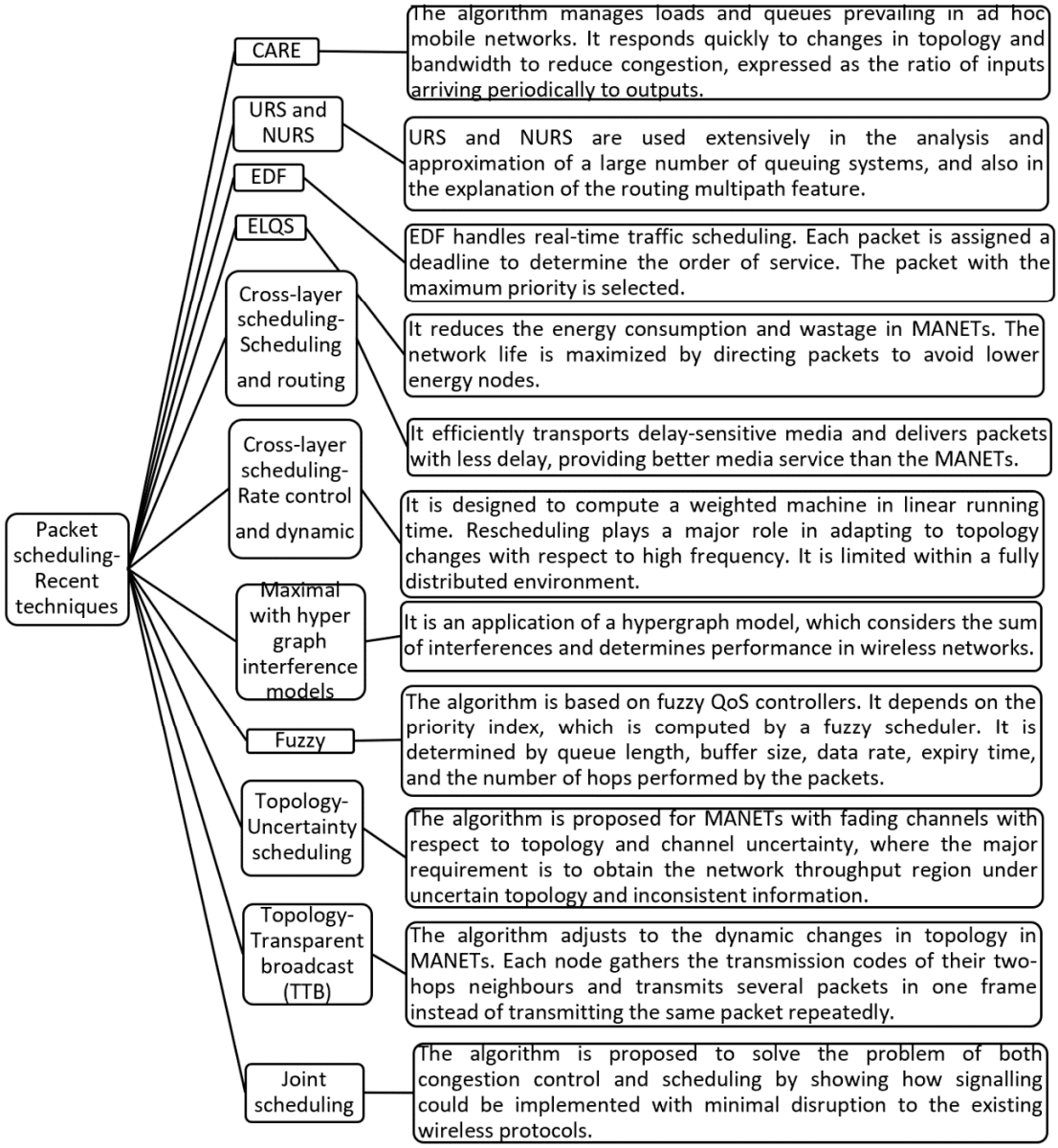
Table 2. Class of channel-aware scheduling algorithms in mobile ad hoc networks.

Class	Algorithm	Description
Channel-aware scheduling	CaSMA	CaSMA concentrates on the awareness and coordination of the end-to-end channel condition to reduce the accumulation of packets in the network and avoid congestion by increasing the number of completely served packets.
	Channel-aware AOMDV	It uses a preemptive handoff technique. In addition, it uses the non-fade duration to select the path during the route. Each node contains a table, which gives the information about the signal strength of the previously received packet.
	PALM	Power-aware link maintenance is based on an ad hoc on-demand distance vector routing algorithm and is responsible for power control with route maintenance. It establishes the routing mechanism in MANETs (Mobile ad hoc networks).

Table 3. Class of techniques for packet scheduling algorithms in mobile ad hoc networks.

Class	Subclass	Algorithm	Description
Packet scheduling	Existing techniques	FIFO	In first-in, first-out, all packets are inserted into a single queue and processed in the order of their arrival. The delay is directly proportional to the length of the queue.
		Priority queuing	Packets are categorized and then grouped into queues with different priorities. Whereas high-priority packets are first processed, low-priority packets are likely to be dropped.
		WFQ	In the weighted fair queuing, packets are scheduled with bandwidth requirements and placed in the respective queues. Packets with a smaller end time are chosen as the next packet for transmission.
		CBWFQ	Class-based weighted fair queuing extends the WFQ functionality and supports user-defined traffic classes. CBWFQ services the class queue fairly based on the weight assigned to the queued packets.
		WHS	Weighted-hop scheduling gives high priority to data packets with only a few remaining hops to pass. A weighted round robin scheduler is used instead of static priority to give a chance to all service classes.
		WDS	The weighted distance scheduling algorithm considers the physical distance using a GPSR, where each data packet contains the destination address.
		RR	The round robin scheduling algorithm preserves the per-flow queues. It provides equal service opportunity among flows.
		Greedy scheduling	Each node redirects its own packet before forwarding the other nodes' packets, which are processed based on FIFO scheduling.
		LLQ	Low-latency queuing has a single strict priority queue for placing separate traffic classes in. This queue allows traffic-sensitive delay treatment before processing other queues. All other queues are regulated by the (percentage of) bandwidth.

Class/subclass	Algorithm/sub-algorithm	Description



Scheme 2. Class of recent techniques for packet scheduling algorithms in mobile ad hoc networks.

3. Greedy Approach

Greedy algorithms are a problem-solving paradigm in which local optimal choices are made at every step, and they are expected to yield an optimal overall solution. However, in many problems, they only produce near-optimal solutions [50]. Let us recall that the greedy approach is well justified for NP-hard problems and it is suited as a solving approach for real problems. However, it is worth noting that for problems with an optimal substructure, greedy algorithms are able to find globally optimal solutions [51].

For the objective of minimizing an unloading cost function, Mazouzi et al. [52] studied discharging policies to choose the tasks that must be discharged before selecting the assigned cloudlet, depending on the network and system resources. The unloading cost is declared to be a combination of task execution time and energy consumption. This

problem is represented with mixed binary programming. As this problem is NP-hard, a distributed linear relaxation-based heuristic approach is proposed. Likewise, a greedy heuristic algorithm is proposed to solve the subproblems, and it calculates the best cloudlet selection and bandwidth allocation based on the task unloading costs.

Ayanzadeh et al. [51] proposed a novel hybrid approach, the quantum-assisted greedy algorithm (QAGA), to improve the performance of physical quantum annealers in the process of finding the global minimum in an Ising Hamiltonian model. Ising models are widely used in many areas of science.They routinely apply a Hamiltonian equation to, among other subjects, alloy thermodynamics and the thermal properties of solids [53]. The QAGA algorithm leverages the quantum annealers to better select candidates at each stage of the greedy algorithm. In fact, it consists of using a quantum annealer at each step to give rise to samples of the ground state of the problem. These samples are used to estimate the probability distribution of the problem variables. Then, it fixes these variables with insignificant uncertainties to move to the next step, where quantum annealing will solve a smaller problem using scattered couplings [51]. In their experimental study, the authors used a D-Wave 2000Q quantum computer with 2000 qubits and new control features to solve larger problems. The results showed that the QAGA approach could find samples with remarkably lower energy values compared to the better-known improvements in the field of quantum annealing.

In their study, Durmus et al. [54] aimed to support the assumption that the classic algorithms generate optimal solutions while the greedy heuristic algorithms generate proximate solutions. They argued that not only do the dimensions of problems increase, but the dimensions and number of constraints in packet programs are also limited, so it is difficult for the classic algorithms to provide the appropriate results. However, regardless of the dimensions or the number of constraints, the greedy approach generates appropriate results. As shown in the literature, both the former and the newer versions, such as the ones proposed by Akçay et al. [55] and Zhou et al. [56], work efficiently. Indeed, greedy algorithms are more efficient in comparison with other complex algorithms; they produce solutions to everyday problems within a reasonable time. Durmus et al. [54] applied greedy and some classical algorithms to the problems of integer linear programming and then compared the differences and similarities of the obtained results. Throughout this work, they classified the most-used algorithms in the literature, as displayed in Table 4, to which we added other greedy approaches and algorithms [51,57].

Table 4. Algorithm classifications and descriptions.

Algorithm Category	Algorithm Name	Category Description
Classical algorithms (exact solution algorithms)	Rounding and graphical method, cutting plane, branch-bound, Balas, Lagrange, branch and cut, Benders decomposition method, and all-integer integer programming.	These algorithms are known in the literature as exact solution algorithms. Their drawback is often their high computational cost. However, they are effective for small and medium instances.

Table 4. *Cont.*

Algorithm Category	Algorithm Name	Category Description
Metaheuristic algorithms	Steepest descent, Dantzig and Ramser's method, and tabu search, genetic, simulated annealing, ant colony, artificial bee colony, particle swarm optimization, and artificial neural networks.	These methods provide solutions close to the optimal one. They can solve a large variety of problems. They are conceptually simple. They can be flexed and adapted according to the problem under study. For example, in genetic algorithms, an analogy is developed between an individual in a population and a solution of a problem in the global solution space. In addition, the simulated annealing method is inspired by the process used in metallurgy to cool down steel. Likewise, particle swarm optimization is a cooperative, population-based global search swarm intelligent metaheuristic and population-based stochastic optimization technique, which is used in solving multimodal continuous optimization problems. Moreover, the tabu search algorithm selects a new search movement in such a way that temporally forbids the evaluation of previous solutions [11].
Greedy algorithms	Greedy algorithms are approaches or techniques in which we consider only one choice at each stage. Their strategy is to seek the best for the current state. It is known in the literature that they may produce optimal global solutions. We may cite the Dijkstra, Kruskal, Prim and Huffman algorithms [58].	They are the most-used algorithms in everyday life. They tackle problems with a given objective function; their strategy is to select each stage variable that has the most benefit. They are quite easy to apply and implement, the computational costs are quite low, and they can be applied to all kinds of problems.

4. Scheduling Algorithms in Peripheral Autonomous Mobile and Pervasive Systems

The scheduling problems we discuss in this paper generally occur in the context of collaborative pervasive architectures within mobile and intelligent distributed systems. We focus our study particularly on the emerging architectures and technologies introduced in recent years, especially at the periphery of autonomous mobile systems.

4.1. Autonomous Mobile and Pervasive Architecture at the Periphery

The problem with these architectures is how to make autonomous smart mobile systems more efficient in terms of completing tasks quickly while respecting their limitations (energy consumption, data storage, computing, connection preservation, and mutual charging between terminals, cloud centers, or other structures). The present study aims to efficiently address these issues in terms of partitioning, scheduling, and load balancing. This requires relying solely on these devices to accomplish the required tasks. This should consider their modest capabilities for innovation, on the one hand, in the system architecture in terms of equipment and maintaining connectivity and, on the other hand, in improving the scheduling and load balancing processes [59]. The scheduling problem we are addressing in this study consists in transferring as many tasks as possible to be managed on the appropriate devices of a peripheral autonomous mobile system. This problem is described as follows.

Assume there are m available devices $d_i, i = 1, \ldots, m$ and n tasks $t_j, j = 1, \ldots, n$. With no priority, the n tasks are processed by the m devices according to their due dates $ddt_j, j = 1, \ldots, n$. The devices are also assigned to the tasks based on the device's ability to

accomplish the task. This is done when the device free execution time is greater than the task execution time, as illustrated by Table 5.

Table 5. Parameters of devices and tasks.

Task	t_1	t_2	t_3.	t_4	t_5	-	-	-	t_n
ddt_j	4	2	6	8	10	-	-	-	5
fdt_j.	5	8	3	11	9	-	-	-	13
Device	d_1	d_2	d_3	d_4	d_5	-	-	-	d_m
ftd_i	3	5	14	12	15	-	-	-	9

As usual, we assume that no device may process more than one task at a time, and no task is processed by more than one machine at a time. We seek an assignment of these tasks to these devices mainly to minimize the overall completion time. This kind of problem is known in scheduling theory as the parallel machine problem. This problem is known to be NP-hard with respect to most of the scheduling criteria, including the criterion we are addressing [29].

4.2. Scheduling Algorithms Based on a Greedy Approach

By comparing different classes of scheduling algorithms, we concluded that greedy algorithms are appropriate for performing tasks on mobile systems within the autonomous mobile architecture at the network's periphery or edges. The reason is that these systems are compatible with greedy algorithms, which are suited for use as a selection algorithm to prioritize options in a search in simple problems. They are generally used in situations where the number of possibilities for improvement is too large to be considered in a meaningful way within a reasonable time [10]. Consequently, to remedy the problems of these networks, it is necessary to design an appropriate algorithm inspired by these greedy algorithms, which require reasonable processing time. For this purpose, we designed a scheduling algorithm based on the greedy approach, composed of one choice at each stage. This approach seems to be appropriate for avoiding complex calculations. Our algorithm was inspired by those like FCFS [46], the expected time to compute (ETC) matrix, minimum execution time (MET), minimum completion time (MCT) [60], Min-Min algorithm for task scheduling [33], longest cloudlet fastest processing (LCFP), and shortest cloudlet fastest processing (SCFP), proposed by Sindhu et al. [37].

To evaluate the performance of our algorithm, we selected three algorithms for comparison in an experimental study with our own, with respect to the quality of the generated solutions, simplicity of calculation, and speed of execution. These algorithms were chosen to cover the different mechanisms for allocating devices to tasks. They are described as follows:

1. The Min-Min algorithm selects the smaller tasks, with respect to processing time, to be executed first on the appropriate devices [61];
2. The LCFP algorithm chooses the longest cloudlet or task to be executed on the fastest processing element;
3. The SCFP algorithm directs the shortest cloudlet or tasks to the fastest processing element.

The approach we developed to solve the problem described above consists of the following steps: (1) problem modeling, (2) describing the greedy algorithm, and (3) the operating mechanism for solving the problem.

Before closing this subsection, let us say a word on the time complexity of the three algorithms. Each of these algorithms is dominated by the sorting procedure with respect to either the set of tasks or speed of the devices. However, the assignment of the tasks to the devices can be implemented in O(nm). Therefore, the time complexity of a sorting procedure is O(nm + max(m log m, n log n).

4.2.1. Problem Modeling

Our approach consists of several steps. The first step begins with breaking down large tasks into tasks based on the device's capabilities. Likewise, based on a criterion for the number of instructions, in the case of applications, and a criterion for size in the case of data. Then, the second step groups and classifies them as well as the network devices into three categories: large, medium, and small. In the third step, the tasks are classified into three categories—large, medium, and small—in such a way that the sizes of the tasks in a category are proportional to the capacities of the devices of the analogous category (Figure 1). These capacities are estimated based on a device score calculation. The most important aspect of a device's capacity is that it has sufficient execution time to accomplish the task assigned to it.

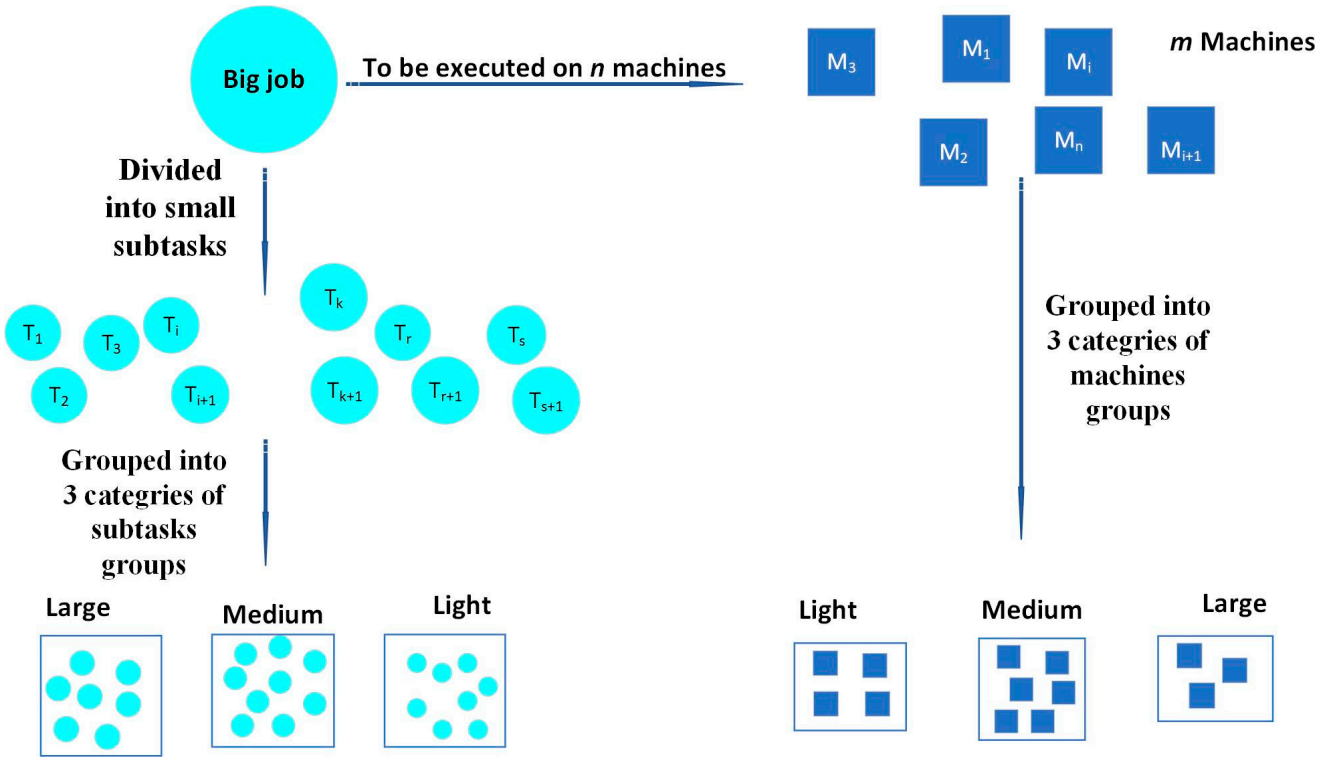


Figure 1. Grouping of devices with tasks based on the device's capabilities.

The score for each device d is determined as a function of the battery energy (ValBattery), storage (ValStok), bandwidth (ValBandwidth), RAM (ValRam), and CPU (ValCPU). To each of these factors is assigned a constant as a percentage value, depending on its importance for device effectiveness. The score is calculated as follows:

$$\begin{aligned} ValScore = \; & ValBattery * PrcBattery + ValStok * PrcStoke \\ & + ValBandwidth * PrcBandwidth + ValRam * PrcRam \\ & + ValCPU * PrcCPU \end{aligned}$$

This formula is implemented as in Algorithm 1: CalculatingScores, which is called within Algorithm 4: SchedulingThroughAnalogousSubgroups. It takes a device d as a parameter and obtains the capacity values of its factors previously mentioned to return its score value (ValScore).

The variables (criteria) used in our algorithms (Algorithms 1–4) are described in Table 6.

Algorithm 1 CalculatingScores (d)

Begin
get factor capacity values of d;
$ValScore = ValBattery * PrcBattery + ValStok * PrcStoke + ValBandwidth * PrcBandwidth + ValRam * PrcRam + ValCPU * PrcCPU$;
return $ValScore$;
End

Table 6. The variables and their description.

Variable	Explanation
ValBattery	Battery energy
ValStok	Storage capacity
ValBandwidth	Bandwidth
ValRam	RAM capacity
ValCPU	CPU capacity
ValScore	Score Value
TaskTime	Task execution time
DeviceTime	Available device execution time

Ejaz et al. [62] formally defined the component execution time in terms of the processor speed, average number of instructions in the component, file size, number of processes running simultaneously on the mobile device, propagation delay, and transmission delay.

Based on [62], we supposed that the available device execution time would be proportional to its score as it depends on the same factors: the bandwidth and the capacities of the battery, storage or warehousing, and computing (RAM and CPU speed). This enabled us to estimate the execution time of a task as well as the available execution time of a device. Thus, we propose Algorithm: DeviceHasExecutionTime for the function that checks if this device has enough time to perform this task for use in our greedy Algorithm 4: SchedulingThroughAnalogousSubgroups. Algorithm 2 uses the following parameters: TaskTime (task rpocessing time), DeviceTime (available device execution time), and a constant β ($0.85 \leq \beta \leq 0.90$), used to ensure that the device's ready execution time is sufficiently larger than the task processing time. It then returns a Boolean value B that indicates whether the machine has sufficient time to process the task in a non-preemptive mode.

Algorithm 2 DeviceHasExecutionTime (TaskTime, DeviceTime)

Begin
Boolean B = False;
Double β such that $0.85 \leq \beta \leq 0.92$;
If TaskTime $\leq$ β*DeviceTime
B = True;
End if
Return B;
End

4.2.2. Proposed Greedy Algorithm

We propose a new approach to map tasks to devices such as laptops, smartphones, tablets, and similar machines, based on their categories and execution times. The approach we propose consists of partitioning each of the groups of tasks and devices into three subgroups and then sorting them in descending order, according to the processing times and device available execution time: LT (longest tasks), MT (medium tasks), ST (shortest tasks), LD (large devices), MD (medium devices), and SD (small devices). The next

phase maps tasks from their subgroup to the appropriate devices in the subgroup of the same category and, if necessary, to other subgroups of devices having greater capabilities. Therefore, the tasks of the LT subgroup are mapped to the devices of the LD device subgroup. The MT subgroup tasks are then assigned to MD subgroup devices and, if required, to LD subgroup devices. The ST subgroup tasks are also mapped to the SD subgroup of devices and, if necessary, to MD and then LD device subgroups, as illustrated in (Figure 2).

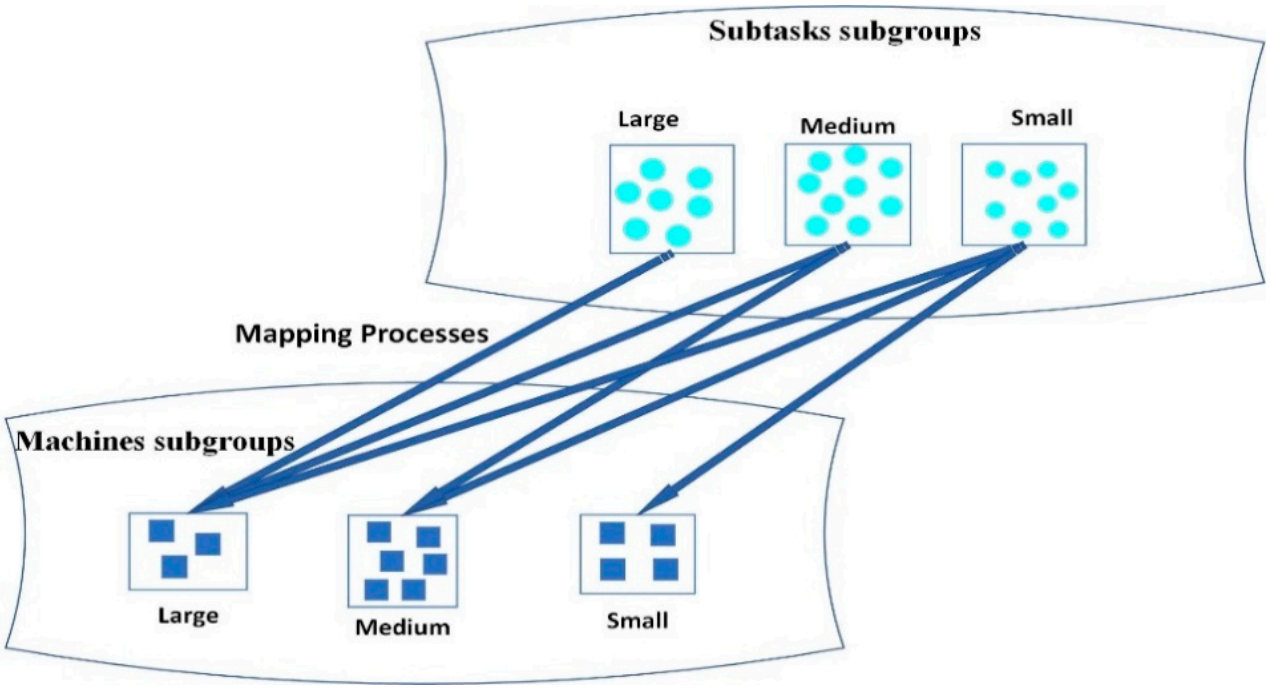


Figure 2. Mapping tasks to appropriate devices.

Using these group patterns, we aimed to initiate and direct parallel processes of assigned devices to tasks in order to reduce the task completion times. We proceeded by restricting the search to the subgroup of devices that was similar to the subgroup of tasks or to larger subgroups of devices. Due to our prior classification, smaller sets of devices did not contain the appropriate devices. As previously mentioned, three parallel processes could be launched. The first was from the LT subgroup to the LD subgroup, but only if the devices in the MD and SD subgroups had no available execution time to perform LT tasks in a non-preemptive mode. Likewise, a second process could be started from the MT subgroup to the MD and LD subgroups. In parallel, a third process could go from the ST subgroup to the SD, MD, and LD subgroups.

The next phase, after the partitioning and the generation of these subgroups, was to proceed through the assignment of the devices to the tasks via Algorithm 4: Scheduling ThroughAnalogousSubgroups.

We structured this algorithm in three parts, which could be executed in parallel with the aim of reducing the overall completion time of the tasks and thus save the device's energy. Many techniques, such as multithread, message passing interface (MPI), or ExaMPI, were used in parallel [63–65]. All the three parts were involved in allocating devices to subtasks from their similar groups at the same time viz. Part I from the large group of subtasks LT to the large group of devices LD, Part II from the medium group of subtasks MT to the medium group MD, and if necessary, to the large group of the devices LD, and Part III from the small group of subtasks ST to the small group SD, and then, as needed, to the MD and LD groups of devices. In addition, we performed load balancing on the devices by updating their information and sorting them again in descending order by the available execution time of the devices.

We also facilitated the understanding and effective use of this algorithm, as mentioned above, through the following algorithms, which will be called by the main algorithm:

- Algorithm 1: CalculatingScores;
- Algorithm 2: DeviceHasExecutionTime;
- Algorithm 3: AllocatingDeviceToTask.

They act as follows. Algorithm 1 calculates the score of a device that would be proportional to its available runtime. This time will be compared to the time of Algorithm 2 to complete a subtask and check whether it is sufficient to execute it. If so, Algorithm 3 will allocate this device to this task.

Algorithm 3 takes a subset of devices and a task as parameters, and it calls Algorithm 2 to check whether the device in this subset has enough time to perform the corresponding task. When such a device is found, it is assigned to the task.

Therefore, the ready execution time of the device is reduced by the execution time of the task. Next, the SubGroupDevice device subgroup is sorted to perform load balancing.

Algorithm 3 AllocatingDeviceToTask (task, SubGroupDevice)

```
Boolean B = false;
Double Dim, DeviceTime, TaskExecutionTime;
Begin
Dim←length (SubGroupDevice);
   For i←0 to Dim-1 Do
   Get DeviceTime of device_i;
   Get TaskExecutionTime of task;
   If DeviceHasExecutionTime (task, device_i) == True
   Assign device_i to task;
   B = True;
   DeviceTime = DeviceTime – TaskExecutionTime;
   Set DeviceTime to device_i;
   Sort SubGroupDevice;
   End If
   End For
   Return B;
End
```

Algorithm 4 works, as is explained above, on the six sorted subgroups already generated (LT, MT, and ST and LD, MD, and SD) to produce a plan for allocating the devices of those subgroups to their tasks while ensuring true load balancing.

4.2.3. Operating Mechanism

Our greedy scheduling algorithm is designed to build the subgroups, as described in Algorithm 4. The operating mechanism is carried out by launching a search from a task subgroup in a similar subgroup of devices and the larger one, if needed, to allocate the appropriate devices to the tasks. It is based on the technique of parallelism, considering the number of processors and kernels of the devices in addition to their load-balancing factor (as illustrated in Figure 3). The score of a device is used as a factor to determine the availability of execution time, which is needed to accomplish specific tasks, as shown in Algorithm 2. This score represents the sum of the percentage of battery usage and the capacity of storage, computing, and connectivity (or bandwidth), as illustrated in Algorithm 1. The mechanism we used to map tasks into the appropriate devices in similar subgroups reduced the completion times. This improves the system performance of mobile devices as it reduces the limited energy consumption of this system's resources.

Algorithm 4 SchedulingThroughAnalogousSubgroups

```
    Integer isdone = 0, lg, j;
    String task;
    CalculatingScores (LD), CalculatingScores (MD), CalculatingScores (SD);
    while isdone < 3
Part I    while LT ≠ Ø Go
              lg←length (LT);
              For j == 0, j < lg Do
                  If AllocatingDeviceToTask (task_j, LD) == True
                      Remove task_j from LT;
                End If
              End For
              If LT == Ø
                  isdone++;
              End If
          End While
Part II   while MT ≠ Ø Go
              lg←length (MT);
              For j == 0, j < lg Do
                  If AllocatingDeviceToTask (task_j, MD) == True
                      Remove task_j from MT;
                Else
                      If AllocatingDeviceToTask (task_j, LD) == True
                      Remove task_j from MT;
                End If
              End For
              If MT == Ø
                isdone++;
              End If
          End While
Part III  while ST ≠ Ø Go
              lg←length (ST);
              For j == 0, j < lg Do
                  If AllocatingDeviceToTask (task_j. , SD) == True
                      Remove task_j from ST;
                Else
                      If AllocatingDeviceToTask (task_j, MD) == True
                Remove task_j from ST;
                Else
                          If AllocatingDeviceToTask (task_j, LD) == True
                            Remove task_j from ST;
                End If
            End If
        End For
        If ST == Ø
            isdone++;
        End If
    End While
End While
```

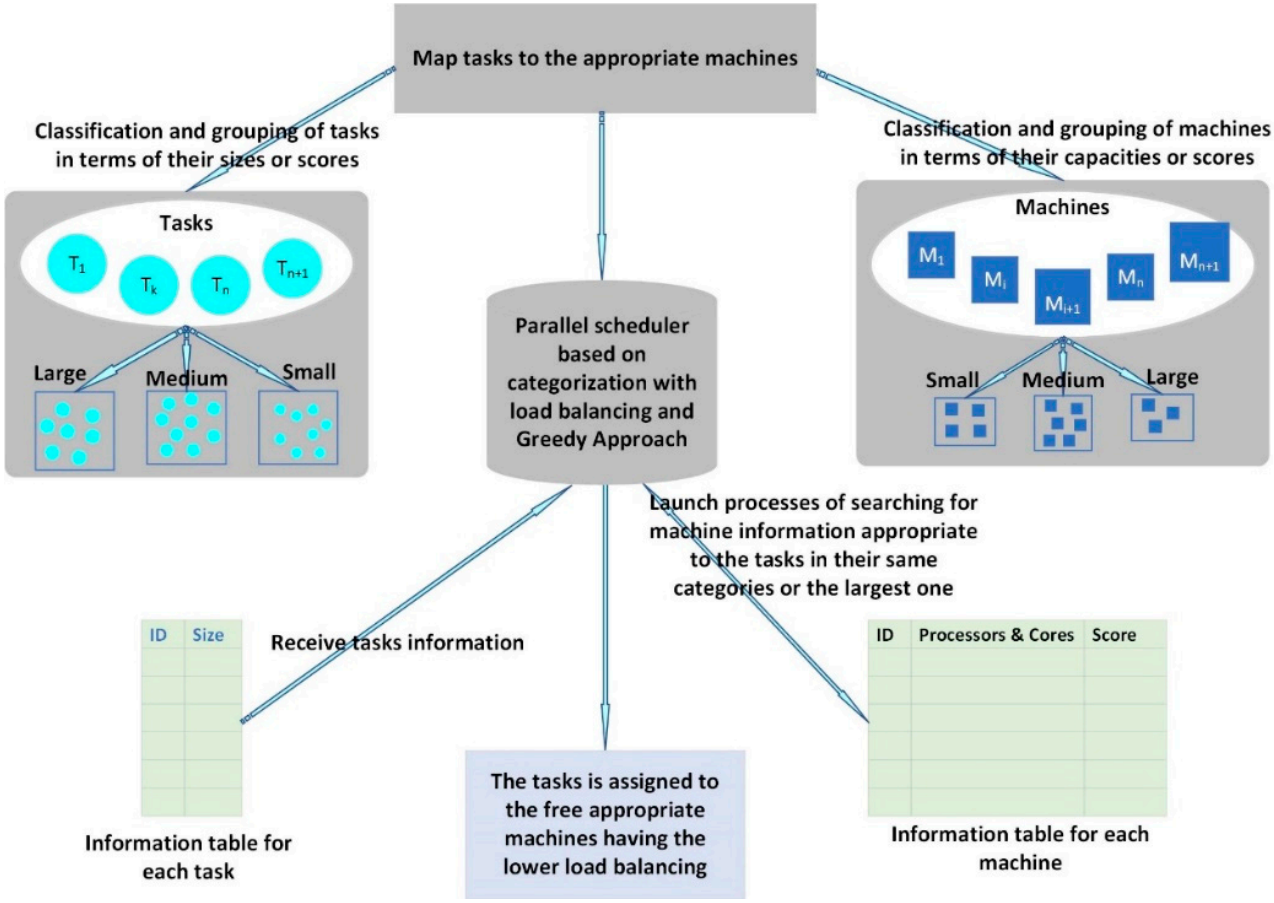


Figure 3. Operating mechanism for mapping tasks to appropriate devices.

We close this section by mentioning the time complexity of our algorithm. To divide the tasks into the three classes, we needed to sort the tasks. This step could be done in O(n log n). The same argument goes for the three groups of devices with respect to their capacity scores. This step could be done in O(m). The rest of the algorithm (i.e., the assignment of those groups of tasks to the corresponding group of devices) could be implemented in O(nm). Therefore, the overall time complexity of our algorithm was O($nm + n \log n$).

5. Simulation

We present in this section the results of a simulation study we conducted to compare the performance of Algorithm 4, as described above, with LCFP, SCFP, and Min-Min. In these algorithms, a variety of factors and their proper values are considered, such as the processing time, bandwidth, number of tasks, number of devices, sending and receiving tasks, waiting times in the queues of the devices, and capacity of each device.

Note that the simulation did not consider some factors in mobile systems such as device communication, bandwidth, the sending and receiving of tasks through networks, and their waiting time in the queues of the devices.

Even though we used a single computer in this experimental study instead of several devices to process all the tasks, we accurately calculated the values of the estimated factors of algorithm performance. We compared and analyzed the four algorithms based on the following criteria:

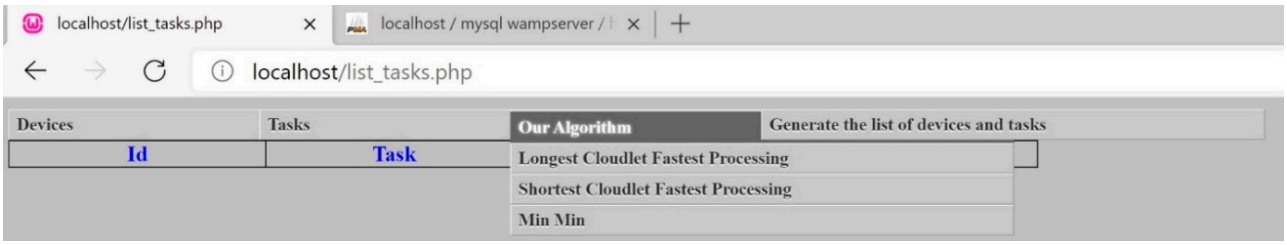
1. The overall completion time, known as the makespan M (M $= \max(C_i : i = 1, \ldots, m)$, where C_i denotes the completion time of device i;
2. The standard deviation SD (SD $= \sqrt{\frac{1}{m}\sum_{i=1}^{m}(C_i - M)^2}$);

3. The absolute difference AD between the maximum completion time M (M $= \max(C_i : i = 1, \ldots, m)$) and the minimum completion time L (L $= \min(C_i : i = 1, \ldots, m)$) (i.e., AD = |M-L|).

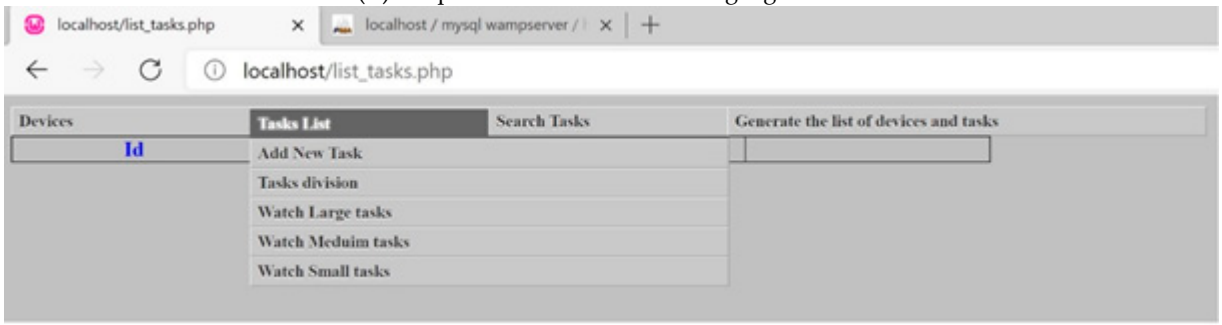
We then computed the percentage for the number of times that each algorithm produced the minimum values of these factors (makespan, SD, and AD). The value of the makespan indicates the time at which the entire set of tasks is completed, thus representing how efficiently the devices are used. The value AD represents the range of the completion times of the devices, thus representing how well-balanced the generated solution is. The standard deviation expresses the dispersion of the completion times of the devices around the makespan. A low value of standard deviation indicates that the completion times tend to be close to the value of the makespan, which is to say that the schedule is well-balanced.

The experimental study we conducted used the WampServer web platform, an SQL server, and the PHP programming language to implement and validate the proposed algorithms. It was implemented on a laptop with the following characteristics: Intel® Core™ i7 processor, x64-based processor, Windows 10 Pro 64-bit OS, 2.3 GHz CPU, and 16 GB of RAM. Figure 4 presents the graphical interface for the simulation.

(**a**) Graphical interface for generating the list of devices and the tasks assigned to them.

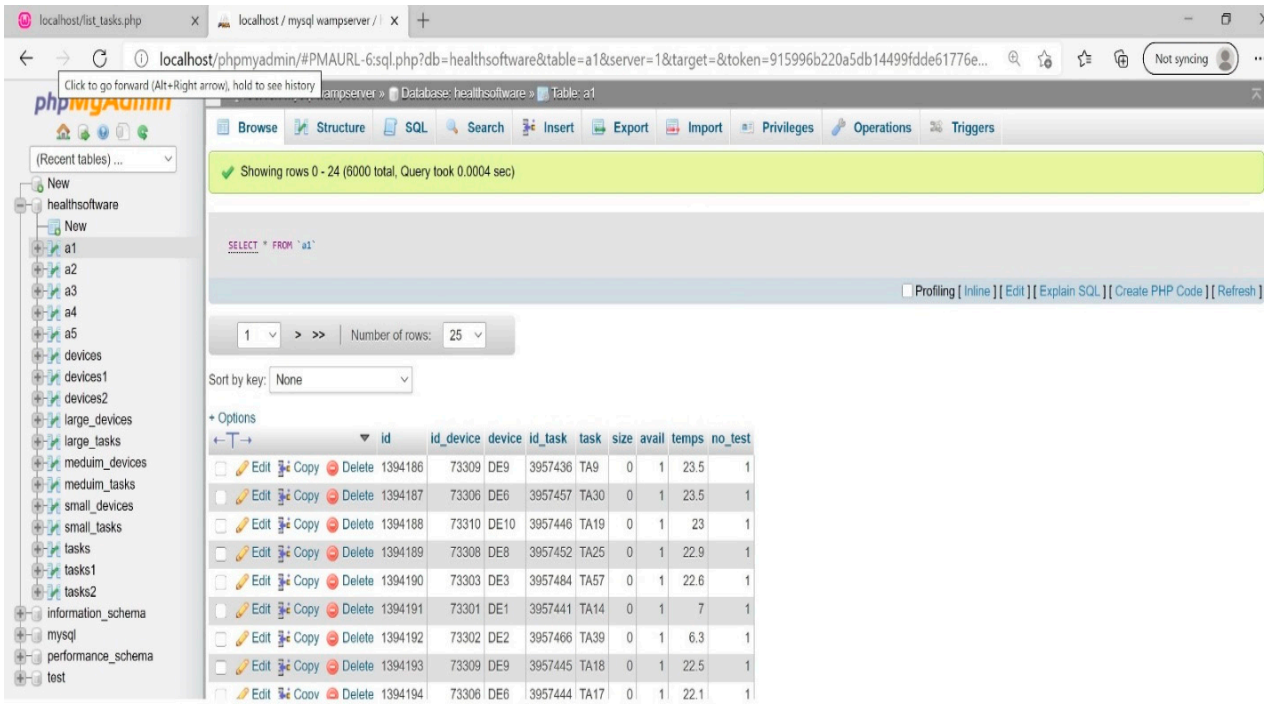


(**b**) Graphical interface for choosing algorithms.



(**c**) Graphical user interface for managing tasks.

Figure 4. *Cont.*



(**d**) Graphical user interface of the SQL database.

Figure 4. Four graphical user interfaces in the menu of the software application.

To ensure the reliability of our method, it is necessary to mention the external factors that might have affected the tested algorithms. Among those factors, we may mention the computer self-configuration resources in execution times. Thus, they influenced the operating system in terms of scheduling their execution in conjunction with these algorithms. As these factors operate automatically while the algorithms are executed, they affect the length of the processing time. Therefore, considering these factors, in terms of the impacts they may have had on the implementation of the algorithms, enabled us to verify and confirm the credence of the results.

For each group of m devices (m = 60, 40, 30, 20, 10, 5), we randomly generated from [11,24] the estimated available time values for each device. Then, for each group of tasks (n = 200, 100, 80, 60, 40, 20, 10), we generated 100 instances, for which the processing times were randomly drawn from [11,53] and on which the four algorithms were executed.

For each instance, we compared the makespan values of the four algorithms. We increased by one the score of the algorithm with the minimum makespan. If two algorithms had the same minimum makespan, we compared their standard deviations and increased by one the score of the algorithm with the minimum standard deviation. If they again produced the same minimum standard deviations, we compared their absolute difference and increased by one the score of the algorithm with the minimum value. If they still had the same absolute difference, we increased by one the score of the algorithm with the minimum score. The best algorithm was the one that had the highest score over the 100 instances.

In the following, we first present a comparison of the minimum makespan, the standard deviation, and the absolute difference with respect to the completion times of the four algorithms (LCFP, SCFP, Min-Min, and Algorithm 4). Next, we also present, for the same instances, the results with respect to the running times of these algorithms.

Displayed in Figure 5 are the different tables that summarize the results of the experimental study which satisfied the criteria (completion time M, standard deviation SD, and absolute difference AD).

Tasks Number	LCFP	SCFP	Min-Min	Our Algorithm
200	**0.51**	0	0	**0.49**
100	0	0	0	**1**
80	0	0	0	**1**
60	0.25	0	0	**0.75**
40	**0.67**	0	0	0.33
30	**0.51**	0	0	**0.49**
20	0.31	0	0	**0.69**
10	0.19	0	0	**0.81**

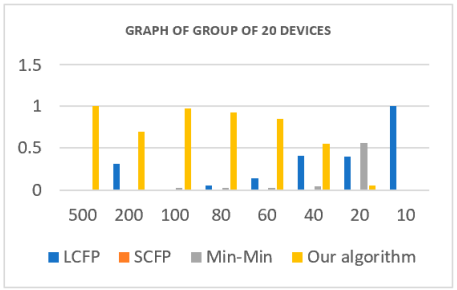
(a) **Group of 5 Devices.**

Tasks Number	LCFP	SCFP	Min-Min	Our Algorithm
200	**1**	0	0	0
100	0	0	0	**1**
80	0.44	0	0	0.56
60	0.101	0	0	**0.899**
40	0.47	0	0	**0.53**
20	**0.92**	0	0.01	0.06
10	0.28	0.32	0.1	0.1

(b) **Group of 10 Devices.**

Tasks Number	LCFP	SCFP	Min-Min	Our Algorithm
200	0.31	0	0	**0.69**
100	0	0	0.027	**0.973**
80	0.054	0	0.027	**0.919**
60	0.136	0	0.023	**0.841**
40	0.408	0	0.041	**0.551**
20	0.393	0	**0.554**	0.054
10	1	0	0	0

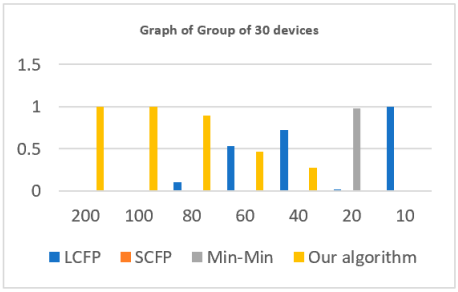
(c) **Group of 20 Devices.**



(d) **Graph of Group of 20 Devices.**

Tasks Number	LCFP	SCFP	Min-Min	Our Algorithm
200	0	0	0	**1**
100	0	0	0	**1**
80	0.107	0	0	**0.893**
60	**0.533**	0	0	0.467
40	**0.724**	0	0	0.276
20	0.022	0	0.978	0
10	**1**	0	0	0

(e) **Group of 30 Devices.**



(f) **Graph of Group of 30 Devices.**

Tasks Number	LCFP	SCFP	Min-Min	Our Algorithm
200	0	0	0	**1**
100	0	0	0	**1**
80	**0.8**	0	0	0.2
60	1	0	0	0
40	**0.696**	0	0	0.304
20	0	0	**1**	0
10	**1**	0	0	0

(g) **Group of 40 Devices.**

Tasks Number	LCFP	SCFP	Min-Min	Our Algorithm
200	0	0	0	**1**
100	0.077	0	0	**0.923**
80	**0.533**	0	0	0.467
60	**1**	0	0	0
40	**1**	0	0	0
20	**1**	0	0	0
10	**1**	0	0	0

(h) **Group of 60 Devices.**

Figure 5. Performance of the algorithms with regard to the quality of the generated solutions.

Figure 5 illustrates the performance of the four algorithms for various number of devices and tasks. Whereas, Figure 6 summarizes the average running times of these four algorithms.

Tasks Number	LCFP	SCFP	Min-Min	Our Algorithm
200	1.752	1.944	1.878	**0.679**
100	0.273	0.304	0.291	**0.058**
80	0.154	0.163	0.155	**0.05**
60	0.125	0.152	0.146	**0.084**
40	0.078	0.093	0.086	**0.065**
20	0.029	0.039	0.035	**0.02**
10	0.011	0.016	0.014	**0.009**

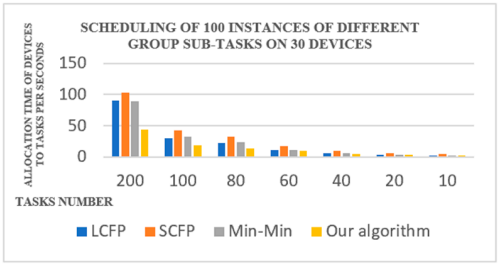
(a) Group of 5 Devices.

Tasks Number	LCFP	SCFP	Min-Min	Our Algorithm
200	1.109	1.298	1.157	**0.441**
100	0.319	0.361	0.31	**0.185**
80	0.193	0.224	0.198	**0.121**
60	0.155	0.19	0.16	**0.089**
40	0.071	0.109	0.078	**0.053**
20	**0.026**	0.055	0.03	0.033
10	**0.012**	0.018	0.013	0.02

(b) Group of 20 Devices.

Tasks Number	LCFP	SCFP	Min-Min	Our Algorithm
200	0.902	1.032	0.898	**0.445**
100	0.297	0.422	0.333	**0.186**
80	0.231	0.323	0.235	**0.139**
60	0.117	0.181	0.119	**0.096**
40	0.065	0.095	0.065	**0.049**
20	0.036	0.063	**0.035**	0.041
10	**0.023**	0.044	0.026	0.031

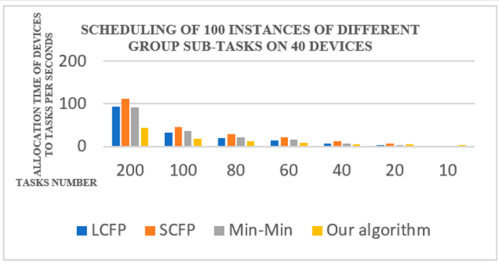
(c) Group of 30 Devices.



(d) Graph of Group of 30 Devices.

Tasks Number	LCFP	SCFP	Min-Min	Our Algorithm
200	0.931	1.129	0.922	**0.43**
100	0.335	0.462	0.371	**0.173**
80	0.197	0.294	0.212	**0.129**
60	0.138	0.214	0.155	**0.097**
40	0.061	0.121	0.063	**0.054**
20	0.035	0.061	**0.034**	0.046
10	**0.019**	0.022	0.021	0.028

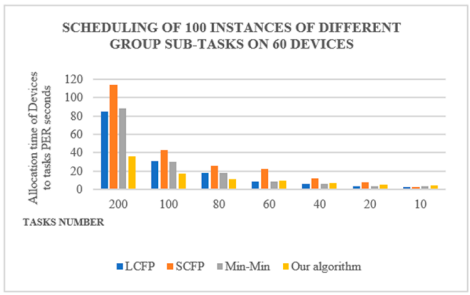
(e) Group of 40 Devices.



(f) Graph of Group of 40 Devices.

Tasks Number	LCFP	SCFP	Min-Min	Our Algorithm
200	0.852	1.137	0.88	**0.362**
100	0.313	0.433	0.298	**0.17**
80	0.185	0.262	0.179	**0.116**
60	**0.086**	0.221	**0.086**	0.097
40	**0.063**	0.123	0.064	0.075
20	**0.039**	0.076	0.04	0.054
10	**0.03**	0.033	0.034	0.045

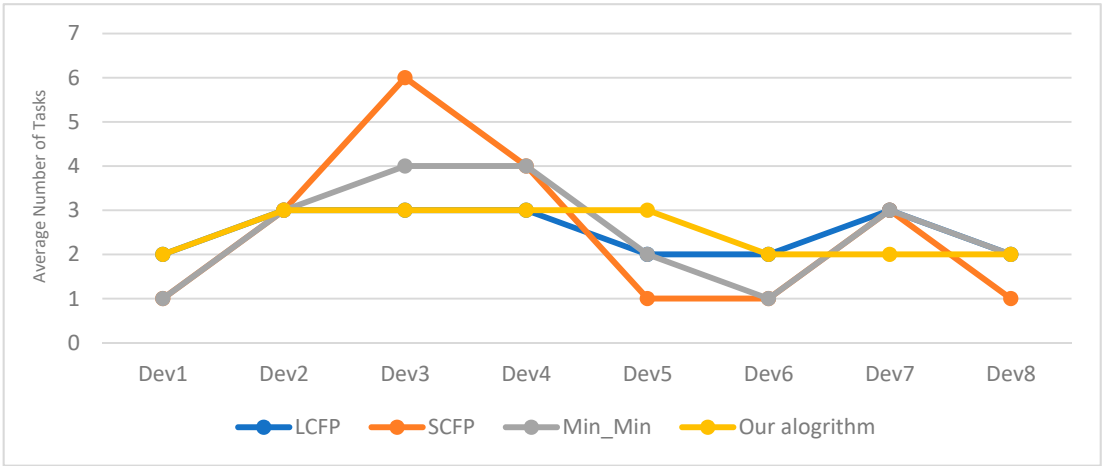
(g) Group of 60 Devices.



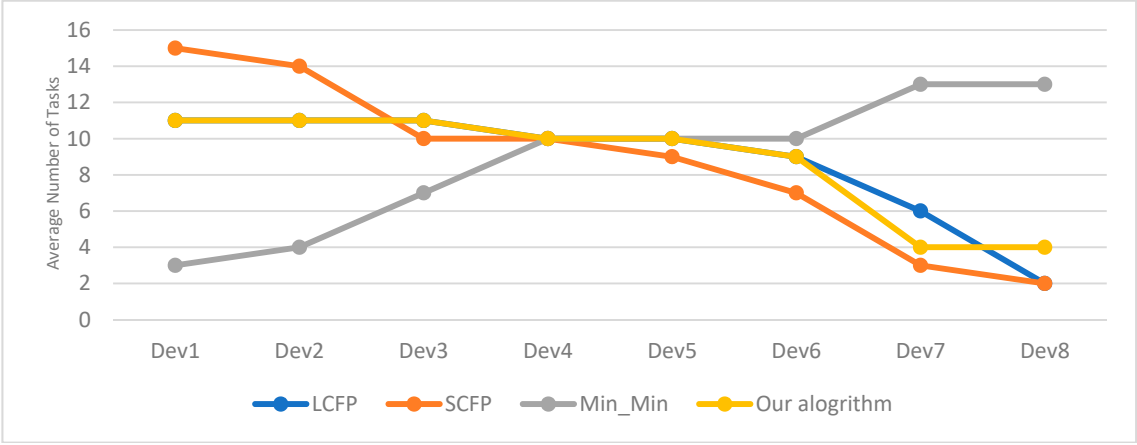
(h) Graph of Group of 60 Devices.

Figure 6. Performance of the algorithms with respect to their running times.

Regarding the scheduling solution, our algorithm succeeded in reducing the completion time of the tasks. At the same time, it improved the device load balancing by preparing devices with larger execution times to be chosen first by the algorithm and accomplish the appropriate tasks. Figure 7 shows our algorithm performance compared with the other algorithms for load balancing in the experimental results, with respect to 10 devices and 20, 70, 100, and 200 tasks. For the correct interpretation of these graphic lines in Figure 7, we proceeded from the fact that the ideal line representing the best performance is the horizontal line crossing them in the middle. Therefore, the best algorithm is the one with the closest graphic line to that ideal line, illustrated by our algorithm for most of the cases.

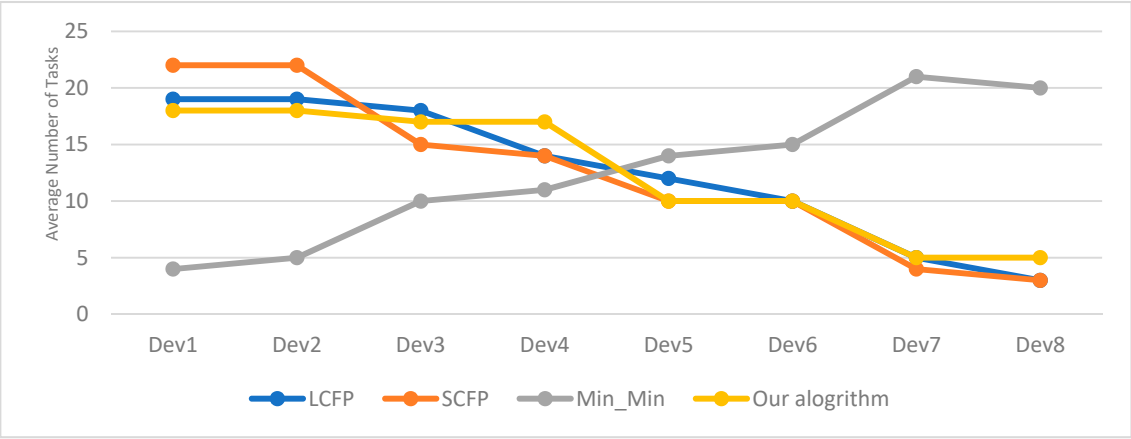


(**a**) Average load balancing among 100 allocation instances of 8 devices to 20 tasks

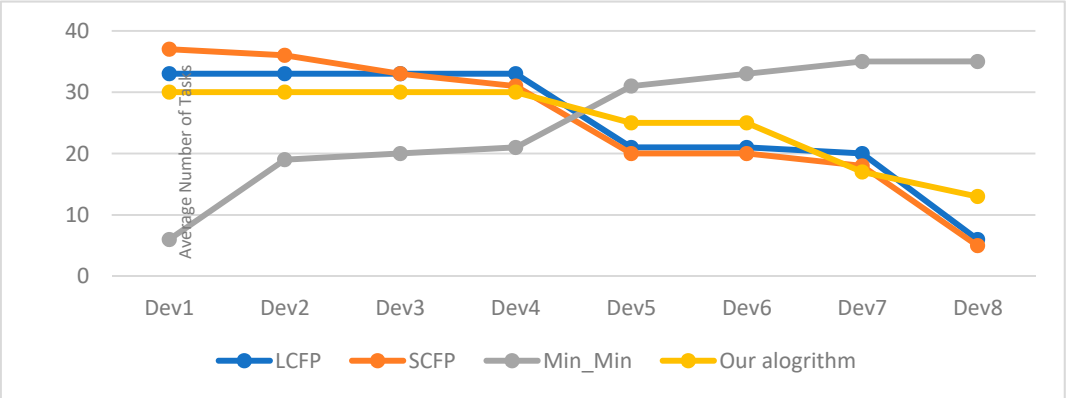


(**b**) Average load balancing among 100 allocation instances of 8 devices to 70 tasks

Figure 7. *Cont.*



(**c**) Average load balancing among 100 allocation instances of 8 devices to 100 tasks



(**d**) Average load balancing among 100 allocation instances of 8 devices to 200 tasks

Figure 7. Experimental results for load balancing with respect to the four algorithms.

The results in these above tables clearly show the superiority of our algorithm. Indeed, our algorithm became more advantageous whenever the difference between the number of tasks and the number of devices became larger (i.e., in cases where the number of tasks is greater than the number of devices). This advantage is directly proportional to the increase in this difference in favor of tasks. Whenever this difference is large, the results generated by our algorithm are the best. The main reason for this is that the device and task groups were partitioned into three similar and relatively small subgroups. This reduced the time it took to navigate through them and find the appropriate devices to execute the tasks. The performance of the remaining algorithms, in decreasing order, is as follows: LCFP, Min-Min, and SCFP. From the above tables, it is easy to observe that our algorithm outperformed the other three algorithms with respect to three criteria: makespan, standard deviation, and the maximum of the absolute difference in completion times, whatever the number of devices or the number of tasks.

To summarize the results, our contribution is made through a dedicated approach adapted to the ubiquitous devices widely used in the IoT paradigm, particularly at the periphery of their communication networks. The goal was to build a new heuristic algorithm and compare it with the other scheduling heuristic algorithmsmostly used in the literature. The main advantages of this approach are brought up through the reduction of the size of the tasks to be performed and breaking them into smaller subtasks. In addition, the process of dividing the groups of subtasks and devices into smaller sub-groups makes it possible for a parallel search to get the desired solution. Proceeding as such makes our algorithm faster and the quality of the generated solutions much better. Consequently, our approach saves a lot of energy, in addition to improving the efficiency of these devices.

6. Conclusions and Future Works

The effectiveness of any scheduling algorithm depends, among other things, upon the overall completion time. Partitioning large tasks into smaller tasks plays an important role in getting these tasks done with shorter completion times. Greedy scheduling algorithms are a popular and widely used approach that produces a balanced, near-optimal solution for scheduling problems. Based on this idea, we proposed a new greedy scheduling algorithm for autonomous smart mobile systems, especially those found at the network's periphery. It takes advantage of the ability of these devices to maintain connectivity through a D2D radio connection. This feature allows devices to continue to communicate and therefore to work with each other, even in the event of wireless failures or communication flooding on the network. Thus, they can always perform the necessary tasks at any time and in any situation. This algorithm is based on the idea of dividing large tasks into smaller tasks to facilitate and speed up the scheduling process and thus their rapid completion. It creates similar subgroups in terms of the size of the tasks and devices. Then, it allocates tasks to devices between similar subgroups and, if necessary, to the larger device subgroups. We have shown through a preliminary study that this algorithm is compatible with mobile systems having resource constraints. Due to the limited capacity of devices, they do not tolerate the long process of computing. Therefore, they are not able to adopt a complex scheduling method. Instead, they require a series of simple scheduling strategies. The preliminary study we conducted showed that our algorithm had a clear advantage over the LCFP, SCFP, and Min-Min algorithms. On the one hand, our algorithm generated small average times for allocating devices to tasks. On the other hand, our algorithm outperformed the three comparison algorithms we presented with respect to the makespan, standard deviation, and range of completion times criteria. Likewise, significant improvements in load balancing could also be added to these advantages.

Current research provides reliability in reducing the completion time for urgent requests and thus in speeding up their completion, especially in harsh environments. For future work, we aim to apply our algorithm to more scenarios and situations. In particular, we aim to use it with more autonomous intelligent mobile systems having a large number of tasks to be performed on multiple devices. These advantages of the proposed method could be added to other functionalities, such as maintaining connectivity and processing via parallelism.

Author Contributions: G.F. designed the proposed greedy algorithms and implemented the four algorithms; G.F., H.M. and D.R. analyzed and validated the computational algorithms; G.F. compared these algorithms and wrote the manuscript in consultation with H.M. and D.R.; All authors have read and agreed to the published version of the manuscript.

Funding: This work is partially funded by the Natural Sciences and Engineering Research council of Canada (NSERC), the 'Fund of the Université du Québec à Chicoutimi (FUQAC) and the Décanat de la Recherche et de la Créaction (UQAC).

Institutional Review Board Statement: Not applicable but the 'Université du Québec à Chicoutimi requires that All research involving the participation of human beings, whether funded or not, conducted or supervised by its professors, employes, and students, be ethically reviewed. Links: http://recherche.uqac.ca/cer/ (accessed on March 2021).

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable but we generated the data in a random way to simulate and compare the algorithms.

Conflicts of Interest: The authors declare that there is no conflict of interest associated with this publication.

References

1. Rayes, A.; Salam, S. *Internet of Things from Hype to Reality—The Road to Digitization*; Springer International Publishing: New York, NY, USA, 2016. [CrossRef]
2. Makori, E.O. Promoting innovation and application of internet of things in academic and research information organizations. *Libr. Rev.* **2017**, *66*, 655–678. [CrossRef]
3. Fadlallah, G.; Mcheick, H.; Rebaine, D.; Adda, M. Towards Mobile Collaborative Autonomous Networks Using Peer-to-Peer Communication. In Proceedings of the ICSENT 2018: 7th International Conference on Software Engineering and New Technologies, Hammamet, Tunisie, 26–28 December 2018; pp. 1–8. [CrossRef]
4. Ghosh, A.; Chakraborty, D.; Law, A. Artificial Intelligence in Internet of Things. *CAAI Trans. Intell. Technol.* **2018**, *3*, 208–218. [CrossRef]
5. Prabhu, C.S.R. Overview—Puting and Internet-of-Things (IOT). *EAI Endorsed Trans. Cloud Syst.* **2017**, *3*, 154378. [CrossRef]
6. Wang, T.; Wei, X.; Tang, C.; Fan, J. Efficient multi-tasks scheduling algorithm in mobile cloud computing with time constraints. *Peer-to-Peer Netw. Appl.* **2018**, *11*, 793–807. [CrossRef]
7. Lim, J.; Lee, D. A load balancing algorithm for mobile devices in edge cloud computing environments. *Electronics* **2020**, *9*, 686. [CrossRef]
8. Arun, C.; Prabu, K. Load Balancing In Mobile Cloud Computing A Review. *Int. J. Comput. Sci. Eng.* **2018**, *6*, 460–465. [CrossRef]
9. Xianglin, W. Application Scheduling in Mobile Cloud Computing with Load Balancing. *J. Appl. Math.* **2013**, *2013*, 409539. [CrossRef]
10. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*, 3rd ed.; MIT Press: Cambridge, MA, USA, 2009.
11. Fadlallah, G.; Rebaine, D.; Mcheick, H. Scheduling problems from workshop to collaborative mobile computing: A state of the art. *Int. J. Comput. Sci. Inf. Secur.* **2018**, *16*, 47–69.
12. Černý, P.; Clarke, E.M.; Henzinger, T.A.; Radhakrishna, A.; Ryzhyk, L.; Samanta, R.; Tarrach, T. From non-preemptive to preemptive scheduling using synchronization synthesis. Formal methods. *System Design* **2017**, *50*, 97–139. [CrossRef]
13. Hosein, P.; Boodhoo, S. Event Scheduling with Soft Constraints and On-Demand Re-Optimization. In Proceedings of the 2016 IEEE International Conference on Knowledge Engineering and Applications (ICKEA), Singapore, 28–30 September 2016; pp. 62–66. [CrossRef]
14. Weaver, P. A Brief History of Sheduling—Back to the Future. In Proceedings of the myPrimavera Conference, Canberra, Australia, 4–6 April 2006; pp. 1–24.
15. Levy, F.K.; Thompson, G.L.; Wiest, J.D. ABCs of the Critical Path Method. *Harv. Bus. Rev.* **1963**, *42*, 98–108.
16. Suresh, P.; Daniel, J.V.; Parthasarathy, V.; Aswathy, R.H. A State of the Art Review on the Internet of Things (IoT) History, Technology and Fields of Deployment. In Proceedings of the 2014 International Conference on Science Engineering and Management Research (ICSEMR), Chennai, India, 27–29 November 2014; pp. 1–8.
17. Rahbari, D.; Nickray, M. Low-latency and energy-efficient scheduling in fog-based IoT applications. *Turk. J. Electr. Eng. Comput. Sci.* **2019**, *27*, 1406–1427. [CrossRef]
18. Zhang, W.; Wen, Y.; Wu, D.O. Collaborative task execution in mobile cloud computing under a stochastic wireless channel. *IEEE Trans. Wirel. Commun.* **2015**, *14*, 81–93. [CrossRef]
19. Coulouris, G.; Dollimore, J.; Kindberg, T.; Blair, G. *Distributed Systems: Concepts and Design*, 5th ed.; Addison Wesley: Boston, MA, USA, 2012.
20. Wang, C.M.; Hong, S.F.; Wang, S.T.; Chen, H.C. A Dual-Mode Exerciser for a Collaborative Computing Environment. In Proceedings of the 11th Asia-Pacific Software Engineering Conference, Busan, Korea, 30 November–3 December 2004; pp. 240–248.
21. Mishra, S.; Mathur, N. Load Balancing Optimization in LTE/LTEA Cellular Networks: A Review. *arXiv* **2014**, arXiv:1412.7273.
22. Pycom LoPy4 Development Board Datasheet. Available online: https://docs.pycom.io/datasheets/development/lopy4 (accessed on 31 January 2021).
23. Bianco, G.M.; Mejia-Aguilar, A.; Marrocco, G. Radio Wave Propagation of LoRa Systems in Mountains for Search and Rescue Operations. In Proceedings of the 2020 XXXIIIrd General Assembly and Scientific Symposium of the International Union of Radio Science, Rome, Italy, 29 August–5 September 2020; pp. 1–3. [CrossRef]
24. Lou, Y.; Chen, J.; Zhang, L.; Hao, D. A Survey on Regression Test-Case Prioritization. *Adv. Comput.* **2019**, *113*, 1–46.
25. Logistic Application of Greedy Algorithms, Vargo, January 2, 2013. Available online: https://vargosolutions.com/logistic-application-greedy-algorithms/ (accessed on 10 January 2021).
26. Le, J. Greedy Algorithm and Dynamic Programming. Expertfy. November 5, 2018. Available online: https://www.experfy.com/blog/bigdata-cloud/greedy-algorithm-dynamic-programming/ (accessed on 10 January 2021).
27. Malik, B.H.; Amir, M.; Mazhar, B.; Ali, S.; Jalil, R.; Khalid, J. Comparison of Task Scheduling Algorithms in Cloud Environment. *Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*. [CrossRef]
28. Zhuravlev, S.; Saez, J.C.; Blagodurov, S.; Fedorova, A.; Prieto, M. Survey of energy-cognizant scheduling techniques. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *24*, 1447–1464. [CrossRef]
29. Brucker, P. *Scheduling Algorithms*, 4th ed.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 1–367, ISBN 978-3-540-20524-1.
30. Le, H.A.; Bui, A.T.; Truong, N.T. An approach to modelling and estimating power consumption of mobile applications. *Mob. Netw. Appl.* **2019**, *24*, 124–133. [CrossRef]

31. Aladwani, T. *Types of Task Scheduling Algorithms in Cloud Computing Environment, Scheduling Problems—New Applications and Trends*; da Rosa Righi, R., Ed.; IntechOpen: London, UK, 2020; pp. 1–12.
32. Sharma, N.; Tyagi, S.; Atri, S. A Survey on Heuristic Approach for Task Scheduling in Cloud Computing. *Int. J. Adv. Res. Comput. Sci.* **2017**, *8*, 1089–1092.
33. Shimpy, E.; Sidhu, M.J. Different Scheduling Algorithms in Different Cloud Environment. *Int. J. Adv. Res. Comput. Commun. Eng.* **2014**, *3*, 8003–8006.
34. Bhoi, U.; Ramanuj, P.N. Enhanced Max-min Task Scheduling Algorithm in Cloud Computing. *Int. J. Appl. Innov. Eng. Manag.* **2013**, *2*, 259–264.
35. Raj, R.J.S.; Prasad, S.V.M. Survey on Variants of Heuristic Algorithms for Scheduling Workflow of Tasks. In Proceedings of the International Conference on Circuit, Power and Computing Technologies [ICCPCT], Nagercoil, India, 18–19 March 2016; pp. 1–4.
36. Ruiz-Vanoye, J.A.; Díaz-Parra, O. Similarities between meta-heuristics algorithms and the science of life. *J. Cent. Eur. J. Oper. Res.* **2011**, *19*, 445–466. [CrossRef]
37. Sindhu, S.; Mukherjee, S. Efficient Task Scheduling Algorithms for Cloud Computing Environment. In *High Performance Architecture and Grid Computing*; Springer: Berlin/Heidelberg, Germany, 2011; Volume 169, pp. 79–83.
38. Kaur, R.; Patra, P.K. Resource Allocation with improved MinMin Algorithm. *Int. J. Comput. Appl.* **2013**, *76*, 61–67.
39. Santhosh, B.; Manjaiah, D.H. An Improved Task Scheduling Algorithm based on Max-min for Cloud Computing. *Int. J. Innov. Res. Comput. Commun. Eng.* **2014**, *2*, 84–88.
40. Etminani, K.; Naghibzadeh, M. A Min-Min Max-Min Selective Algorithm for Grid Task Scheduling. In Proceedings of the 2007 3rd IEEE/IFIP International Conference in Central Asia on Internet, Tashkent, Uzbekistan, 26–28 September 2007; pp. 1–7.
41. Mittal, S.; Katal, A. An Optimized Task Scheduling Algorithm in Cloud Computing. In Proceedings of the IEEE 6th International Conference on Advanced Computing (IACC), Bhimavaram, India, 27–28 February 2016; pp. 197–202.
42. Razaque, A.; Vennapusa, N.R.; Soni, N.; Janapati, G.S.; Vangala, K.R. Task Scheduling in Cloud Computing. In Proceedings of the 2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT), Farmingdale, NY, USA, 29 April 2016.
43. Rimal, B.P.; Maier, M. Workflow scheduling in multi-tenant cloud computing environments. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 290–304. [CrossRef]
44. Wu, X.; Deng, M.; Zhang, R.; Zeng, B.; Zhou, S. A task Scheduling Algorithm Based on QoS-driven in Cloud Computing. *Procedia Comput. Sci.* **2013**, *17*, 1162–1169. [CrossRef]
45. Lantharthong, T.; Rugthaicharoencheep, N. Network Reconfiguration for Load Balancing in Distribution System with Distributed Generation and Capacitor Placement. *World Acad. Sci. Eng. Technol. Int. J. Electr. Comput. Eng.* **2012**, *6*, 396–401.
46. Bansal, N.; Awasthi, A.; Bansal, S. Task Scheduling Algorithms with Multiple Factor in Cloud Computing Environment. In *Information Systems Design and Intelligent Applications*; Springer: New Delhi, India, 2016; Volume 433, pp. 619–627.
47. Sherin, B.C.; Mary Anita, E.A. A Survey of Scheduling Algorithms for Wireless Ad-hoc Networks. *Int. J. Adv. Sci. Eng.* **2018**, *4*, 776–787. [CrossRef]
48. Enzai, N.I.M.; Anwar, F.; Mahmoud, O. Evaluation study of QoS-enabled AODV. In Proceedings of the International Conference on Computer and Communication Engineering, Kuala Lumpur, Malaysia, 13–15 May 2008; pp. 1254–1259.
49. Kurose, J.F.; Ross, K.W. *Computer Networking a Top Down Approach Featuring the Internet*, 2nd ed.; Addison Wesley: Boston, MA, USA, 2004.
50. DeVore, R.A.; Temlyakov, V.N. Some remarks on greedy algorithms. *Adv. Comput. Math.* **1996**, *5*, 173–187. [CrossRef]
51. Ayanzadeh, R.; Halem, M.; Dorband, J.; Finin, T. Quantum-assisted greedy algorithms. *arXiv* **2019**, arXiv:1912.02362.
52. Mazouzi, H.; Achir, N.; Boussetta, K. DM2-ECOP: An Efficient Computation Offloading Policy for Multi-user Multi-cloudlet Mobile Edge Computing Environment. *ACM Trans. Internet Technol.* **2019**, *19*, 1–24. [CrossRef]
53. Huang, W.; Kitchaev, D.A.; Dacek, S.; Rong, Z.; Urban, A.; Cao, S.; Luo, C.; Ceder, G. Finding and proving the exact ground state of a generalized Ising model by convex optimization and MAX-SAT. *Phys. Rev. B* **2016**, *94*, 134424. [CrossRef]
54. Durmus, B.; Guneri, O.; Incekirik, A. Comparison of Classic and Greedy Heuristic Algorithm Results in Integer Programming: Knapsack Problems. *Mugla J. Sci. Technol.* **2019**, *5*, 34–42. [CrossRef]
55. Zhou, Y.; Chen, X.; Zhou, G. An Improved Monkey Algorithm for a 0-1 Knapsack Problem. *Appl. Soft Comput.* **2016**, *38*, 817–830. [CrossRef]
56. Pinedo, M. Offline Deterministic Scheduling, Stochastic Scheduling, and Online Deterministic Scheduling: A Comparative Overview. In *Handbook of Scheduling*; Leung, J.Y.-T., Ed.; Chapman & Hall/CRC: London, UK, 2004.
57. Curtis, S.A. The classification of greedy algorithms. *Sci. Comput. Program.* **2003**, *49*, 125–157. [CrossRef]
58. Sahni, S.; Horowitz, E. *Fundamentals of Computer Algorithms*; Computer Science Series; W. H. Freeman and Company: New York, NY, USA, 1984.
59. Chen, Y.; Yang, S.; Hwang, J.; Wu, M. An Energy-Efficient Scheduling Algorithm for Real-Time Machine-to-Machine (M2M) Data Reporting. In Proceedings of the IEEE Global Communications Conference, Austin, TX, USA, 8–12 December 2014; pp. 4442–4447.
60. Sheikh, S.; Nagaraju, A. A Comparative Study of Task Scheduling and Load Balancing Techniques with MCT using ETC on Computational Grids. *Indian J. Sci. Technol.* **2017**, *10*, 1–14. [CrossRef]
61. Sharma, N.; Tyagi, S.; Atri, S. A Comparative Analysis of Min-Min and Max-Min Algorithms based on the Makespan Parameter. *Int. J. Adv. Res. Comput. Sci.* **2017**, *8*, 1038–1041.

62. Ahmed, E.; Naveed, A.; Ab Hamid, S.H.; Gani, A.; Salah, K. Formal analysis of seamless application execution in mobile cloud computing. *J. Supercomput.* **2017**, *73*, 4466–4492. [CrossRef]
63. Xie, G.; Xiao, X.; Peng, H.; Li, R.; Li, K. A Survey of Low-Energy Parallel Scheduling Algorithms. *IEEE Trans. Sustain. Comput.* **2021**. [CrossRef]
64. Mahafzah, B.A.; Jabri, R.; Murad, O. Multithreaded scheduling for program segments based on chemical reaction optimizer. *Soft Comput.* **2021**, *25*, 2741–2766. [CrossRef]
65. Skjellum, A.; Rüfenacht, M.; Sultana, N.; Schafer, D.; Laguna, I.; Mohror, K. ExaMPI: A Modern Design and Implementation to Accelerate Message Passing Interface Innovation. In *High Performance Computing. CARLA 2019. Communications in Computer and Information Science*; Crespo-Mariño, J., Meneses-Rojas, E., Eds.; Springer: Cham, Switzerland, 2020; Volume 1087. [CrossRef]