information

Article

Online Task Scheduling of Big Data Applications in the Cloud Environment

Laila Bouhouch [1,2,*], Mostapha Zbakh [1] and Claude Tadonki [2]

[1] National School of Computer Science and Systems Analysis, Mohamed V University in Rabat, Rabat 10112, Morocco; m.zbakh@um5s.net.ma
[2] Mines ParisTech-PSL Centre de Recherche en Informatique (CRI), 77305 Paris, France; claude.tadonki@mines-paristech.fr
* Correspondence: laila_bouhouch2@um5.ac.ma

Abstract: The development of big data has generated data-intensive tasks that are usually time-consuming, with a high demand on cloud data centers for hosting big data applications. It becomes necessary to consider both data and task management to find the optimal resource allocation scheme, which is a challenging research issue. In this paper, we address the problem of online task scheduling combined with data migration and replication in order to reduce the overall response time as well as ensure that the available resources are efficiently used. We introduce a new scheduling technique, named Online Task Scheduling algorithm based on Data Migration and Data Replication (*OTS-DMDR*). The main objective is to efficiently assign online incoming tasks to the available servers while considering the access time of the required datasets and their replicas, the execution time of the task in different machines, and the computational power of each machine. The core idea is to achieve better data locality by performing an effective data migration while handling replicas. As a result, the overall response time of the online tasks is reduced, and the throughput is improved with enhanced machine resource utilization. To validate the performance of the proposed scheduling method, we run in-depth simulations with various scenarios and the results show that our proposed strategy performs better than the other existing approaches. In fact, it reduces the response time by 78% when compared to the *First Come First Served* scheduler (*FCFS*), by 58% compared to the *Delay Scheduling*, and by 46% compared to the technique of *Li et al.* Consequently, the present *OTS-DMDR* method is very effective and convenient for the problem of online task scheduling.

Keywords: cloud computing; big data; Cloudsim; task scheduling; data migration; data replication

Citation: Bouhouch, L.; Zbakh, M.; Tadonki, C. Online Task Scheduling of Big Data Applications in the Cloud Environment. *Information* **2023**, *14*, 292. https://doi.org/10.3390/info14050292

Academic Editor: Hamid Reza Arabnia

Received: 9 March 2023
Revised: 8 April 2023
Accepted: 11 May 2023
Published: 15 May 2023

1. Introduction

Big Data analytics is essential to many applications and supports a variety of user services. The advance of internet technology has led to big data analytics, and thus, big data analytics tasks [1]. As a result, managing big data tasks and supporting data-intensive applications is now possible using cloud data centers [2]. Most of the big data applications [3–7] are in the form of online task processing. However, it is clear that these tasks are both computation- and data-intensive [8], hence it becomes a challenge to efficiently handle them.

Furthermore, in a dynamic cloud environment, resources such as virtual machines, storage, and networking components are provisioned and deprovisioned as needed to meet changing demands [9]. This increases the complexity of the task scheduling problem, as response time is a crucial decision-making parameter for data-intensive tasks. Thus, scheduling methods should not only aim to reduce task response time but also consider data migration and replication management to improve response time, throughput, and resource utilization [10]. In order to cope with dynamic cloud environments, researchers proposed several task scheduling strategies [11–14] to find a trade-off between different goals and achieve efficient task planning.

Data migration in cloud environments involves the process of transferring data from one storage or computing system to another within the same cloud infrastructure. The goal of data migration is to ensure that data are available in the right location at the right time to meet task needs. Managing data is a crucial factor to consider when dealing with data-intensive tasks, and there are two cases to consider: local data and remote data. Data locality occurs when the task and its required data are on the same server, while remote data involves accessing required data stored on different servers than those hosting the consumer tasks. Accessing remote data involves additional time [15] due to the migration process that occurs when moving the datasets over the network and writing them to the disks. There are several challenges associated with data migration in cloud environments, such as ensuring data security, maintaining data integrity, handling placement and storage, and managing costs [16–18].

Therefore, to reduce the task response time, it is preferable to schedule the task in the server where all or most of its required datasets are stored. Otherwise, the task has to be scheduled at least in the server ensuring an optimal data migration time. The scheduling process is also related to other metrics such as the heterogeneity of the configuration of the servers [19] in terms of CPU frequency, number of CPUs, size of available memory, etc., as well as the load of each server—to avoid both overloaded and underloaded nodes [20].

Data replication in cloud computing refers to the process of creating multiple copies of data and storing them across different physical locations or servers within a cloud computing environment [21]. This is completed to ensure that data are highly available, resilient, and can be accessed quickly in case of a failure or outage [22]. When it comes to data replication in cloud environments, there are several key challenges that need to be addressed. These include network bandwidth, data consistency, replication latency, and cost [23–26]. It is important to mention that, beside the replicas created during the initial placement of data, in this paper, the data migration process generates duplicated data that should be managed efficiently for better data locality.

Due to dynamic provisioning resources for online tasks, there is a constant queue of tasks waiting to be processed. However, since servers have limited storage capacity, not all incoming tasks can be scheduled to run locally, making it challenging to efficiently utilize available resources for improved response time and throughput. As a result, servers may become either underloaded or overloaded, depending on the demand being lower or higher than their processing capacity [27].

To deal with the above issues, we address the task scheduling problem by proposing an Online Task Scheduling strategy based on Data Migration and Data Replication (*OTS-DMDR*) with the main focus of selecting the most suitable tasks to be executed on each server.

In our proposed algorithm *OTS-DMDR*, we first establish a model to estimate the response times of tasks in different servers. We then decide between the following three actions: (1) achieve data locality by scheduling the task on the server storing the required datasets; (2) delay the task execution so it is scheduled on another server for better data locality; (3) schedule the task on a remote server that gives an optimal response time, including the migration process. Additionally, in the task response time, we consider the replicated datasets, the computational capacity, and the load of each server to prevent underloaded and overloaded machines. Finally, after comparing our online task scheduling *OTS-DMDR* with other existing algorithms in the literature, the corresponding results show that the proposed *OTS-DMDR* can guarantee better average response time by 46% compared to Li et al. [28], by 58% compared to the *Delay Scheduling* method, and acceptable load balancing between machines, improving the overall system efficiency.

In summary, our contributions can be organized as follows:

- Formalize the *OTS-DMDR* problem considering both the heterogeneity and the processing capacity of the servers, together with locality, movement, and replication of datasets.

- Propose algorithms to estimate the different costs and measure the tasks' adequacy to the servers to seek a better task-to-server allocation.
- Conduct extensive simulation experiments to evaluate the efficiency of our algorithm, *OTS-DMDR*.

The rest of this paper is organized as follows. Section 2 describes the related work on various scheduling methods and frameworks. Section 3 presents how our online task model was established. The proposed *OTS-DMDR* is outlined with different implemented algorithms in Section 4. We conduct various experiments and assess the algorithm's effectiveness in Section 5. Finally, Section 6 draws conclusions and some perspectives.

2. Related Work

In this section, we desribe some common metrics studied in different task scheduling methods. Then, we review the two main types of scheduling techniques based, single objective and multiple-objective. Finally, we highlight our motivation.

2.1. Common Used Metrics

Big data processing requires a lot of computing resources; thus, effectively managing the resources is essential due to the heterogeneity and dynamism of the environments. Scheduling algorithms are a set of policies, procedures, and rules, implemented to assign the best resource for task execution with the aim to accomplish the service provider's and cloud user's objectives. Each of the existing scheduling methods [29–32] take into consideration several performance metrics. The most common metrics are mentioned below:

- Throughput [33,34];
- Execution time [35–37];
- Response time [38,39];
- Execution cost [32,40];
- Deadline and Budget constraints [41–43];
- Load balancing [20,44,45];
- Fault tolerance [46,47];
- SLA violation [41,48];
- Energy consumption [49–51];
- Data transfer [28].

2.2. Single-Objective Scheduling Techniques

Some of the earliest scheduling algorithms that have been studied in the literature are [30,52,53].

The First Come First Served (FCFS) scheduling algorithm is the most traditional one. Its idea is that the last arrived tasks have to wait until the end of the execution of earlier ones [52]. Only after a task ends will the next task in the queue be considered. The FCFS method is also the main scheduler used in the Hadoop framework [54]. The disadvantages of this strategy are that the waiting time for tasks is increased and it does not consider task size. Moreover, it fails in balancing the workload among machines and decreases data locality.

In the Shortest Job First (SJF) method [30], it chooses the shortest task to be executed first in order to reduce the execution time. Although, due to uneven load distribution on the servers, the algorithm fails to respect the SLA.

The Round Robin (RR) algorithm [53] circularly distributes tasks and an equal amount of CPU time is given to every task. The round-robin strategy results in a higher average waiting time.

The traditional scheduling algorithms (mentioned above) did not find the best solution to the multi-dimensional scheduling problem since the scheduling algorithm should simultaneously optimize various parameters [11,31] such as *response time* with *resource utilization*, *makespan*, *cost*, *energy consumption*, etc.

2.3. Multi-Objective Scheduling Techniques

To address the issue mentioned above, many scheduling techniques have been proposed, with focus on enhancing multiple parameters simultaneously [32,55–58].

Shyam and Manvi [55] suggested a resource allocation technique that maximizes resource usage while minimizing time and budget. The method relies on VM migration to improve the placement ratio of VMs, which is advantageous for both cloud users and providers.

Wang et al. [32] proposed a dynamic resource provisioning algorithm that is ideal in terms of service availability, migration, and leasing costs. The study considers resources such as CPU, memory, and storage.

For data-intensive applications, Zhao et al. [56] proposed an energy-efficient scheduling technique where datasets and tasks are treated as a binary tree using a data correlation clustering algorithm. By decreasing the number of active VMs and data transmission time, the proposed strategy is used to minimize the energy usage of cloud data centers. However, the online scheduling is not considered.

To minimize the execution time, while increasing resource usage, the work in [57] proposed a scheduling algorithm based on IBA (Improved Backfill Algorithm) and takes into account task priority. Priority is one of the important metrics for users who want to pay more for a quicker answer (VIP request). The limitation of this technique resides in the performance that is decreasing once the number of tasks grows.

The term Online Potential Finish Time was coined in [58] to improve execution time and cost in cloud computing. Tasks are distributed onto powerful virtual machines, which can execute tasks with the least amount of delay.

Reddy G. Narendrababu et al. [59] introduced a modified version of the ant colony optimization algorithm (MACO) that is tailored to multi-objective task scheduling in cloud environments. MACO improves upon the original ACO algorithm by assigning pheromone values to virtual machines (VMs) based on their RAM, bandwidth, storage, processing speed, makespan, and other factors. This approach facilitates the efficient allocation of tasks to VMs that are best suited for the task, resulting in better resource utilization and reduced degree of imbalance. The MACO algorithm outperforms basic ACO, PSO, and GA algorithms in terms of makespan, system load balance, and task assignment efficiency.

A dynamic round robin scheduling algorithm is proposed in [60]. Authors dynamically calculate the time quantum for each round by taking into account the differences among the maximum burst times of the three tasks in the ready queue. One potential issue with this method is that it does not efficiently handle the starvation challenge. Despite this concern, the proposed method offers significant benefits such as reducing the average turnaround time, decreasing the average waiting time, and minimizing the number of context switches.

The research in [61] employed a genetic meta-heuristic algorithm to enhance performance by investigating the environment. The fitness function combined throughput, response time, and cost criteria, producing overall enhancements. To ensure that all parameters were given equal consideration, normalization was employed, resulting in relative optimization. The suggested method improved waiting time, makespan, and utility while slightly reducing costs, resulting in superior service for both providers and users. The main limitation of this work is that it does not address the topic of data-intensive online tasks. Moreover, it could be hard to adapt such a solution for data-intensive online tasks.

The authors in [62] introduce the Hard Disk Drive and CPU Scheduling (HCS) algorithm for devices with multiple cores and hard disks, aiming to optimize execution time and energy consumption while minimizing missed tasks. It considers scheduling multiple parallel tasks with individual deadlines and utilizes multiple stages to execute sorted tasks. However, this study does not consider memory effects, network bandwidth, and latency of multi-core systems.

2.4. Our Motivation

It is obvious from the research works referenced above that the majority of authors focus primarily on resources, especially computing resources, since the main activity of the task is on CPUs. However, the frequent I/O operations required for big data analytics tasks make data locality more crucial, as local I/O can minimize task execution time more effectively than remote ones [1]. The most fundamental scheduling technique used in big data systems is called DLB (Data Location Based) [12]. For that, a delay algorithm or a matchmaking algorithm may be used.

The delay algorithm [63] resolves locality through the waiting method. The goal of this technique is to assign tasks to servers based on the location of their input data, i.e., considering that a node in the cluster is free and asks for a task in the queue. It may be possible that the data required by the selected task are not stored in the given available node. Hence, the delay scheduling technique delays the task until a node containing the required data becomes available (achieving data locality). Although, to prevent starvation, a task that has been waiting for a long time is not executed regardless of the locality of the input data.

The matchmaking algorithm [64] implies that every node has an equal opportunity to take advantage of the local tasks before a new task is assigned to the nodes. A local task's input data are kept at the relevant node.

Generally, the DLB approach attempts to reduce the amount of time spent transferring data and provide fairness by achieving data locality. The problem is that when the data are not spread equally across the nodes, the servers' load may be unbalanced, and thus the execution time may be longer. Yet, hot data spots may affect both the matchmaking and the delay algorithms, meaning that some nodes may be overloaded with tasks as a result of their data storage while others are left idle. The tasks are mostly scheduled on the servers where the majority of their input data are stored. Therefore, a few servers are always used, which makes them overloaded. As a result, the task execution time is larger and the throughput is lower.

The aforementioned scheduling frameworks prioritize the task scheduling problem while ignoring the deployment of incoming data. Because handling resources and tasks are seen as the most expensive, the majority of prior works focused on managing them. However, as scientific applications become more and more data-intensive, handling storage, data management, and computing resources is increasingly critical [65]. The most related scheduling strategy to our work is presented by Li et al. in [28]. They proposed an online job scheduling based on data migration by selecting a proper task to be scheduled when a server becomes available. The authors make a trade-off between two costs: (1) the task is assigned to a remote server with a data transfer cost, (2) the task will wait a certain amount of time for a server that ensures the locality of the data for the task with a waiting time cost.

Paper [28] schedules tasks sequentially, one task after the other, which increases the waiting time for tasks in the queue. Moreover, when migrating data, a set of characteristics were not considered such as machine performance, the network between machines, storage space, and task requirements in terms of CPU, RAM, size, and volume of required data per task. Consequently, this does not guarantee an optimal result. Furthermore, the process of handling data replication has not been discussed in [28].

In summary, the specificity of our *OTS-DMDR* approach is that we use a different concept to assign tasks to nodes. In fact, we choose a set of tasks from the incoming tasks from the queue and assign them to the nodes with potentially the best response time. To calculate the response time, we take into account the data migration time, including the data replication process and the computing power of each server (e.g., CPU usage, RAM availability, and storage capacity). Moreover, we anticipate the execution of a task in a better node by considering the delay time and balancing the load between the servers.

It is important to note that data placement and replication techniques were investigated in many papers such as [66–68]. These papers can not be compared to the scheduling algorithms reviewed in our related work section, but we mention them because they study the importance of data availability in the cloud computing environment. Table 1 summarizes the above analysis.

Table 1. Summary of scheduling methods in the literature

Method	Technique	Advantages	Limitations	Parameters
First Come First Served [52]	The last arrived tasks must wait until the end of the execution of earlier ones.	Simple to implement and efficient.	Increases the waiting time of tasks and tasks size not considered. Imbalance load and decreases data locality.	-
Shortest Job First [30]	Chooses the shortest task to be executed first.	Reduces execution time in comparison with FCFS and RR.	Uneven load distribution on the servers.	Execution time. Response time.
Round Robin [53]	Circularly distributes tasks.	An equal amount of CPU time is given to every task.	A higher average waiting time.	-
Shyam and Manvi [55]	VM Migration.	Maximizes resource usage while minimizing time and budget.	Needed more agents for searching the best resource.	Execution time. Makespan time. Response time. Resource utilization.
Wang et al. [32]	Dynamic resource provisioning.	Considers resources such as CPU, memory and storage.	Data locality and replication not addressed.	Execution cost. Availability.
Zhao et al. [56]	Energy-efficient technique where datasets and tasks are treated as a binary tree using a data correlation clustering algorithm.	Minimizes the energy usage of cloud data centers.	Online scheduling is not considered.	Execution cost. Resource utilization. Energy consumption.
Dubey et al. [57]	Scheduling algorithm based on IBA (Improved Backfill Algorithm).	Minimizes the execution time and increases resource usage.	More tasks imply less performance. Considers task priority.	Execution time. Makespan. Resource utilization.
Elseoud et al. [58]	Online Potential Finish Time heuristic algorithm.	Improves execution time and cost in cloud computing execute tasks with the least amount of delay.	Data locality and replication not addressed.	Execution time/cost. Makespan. Response time. Resource utilization.
Delay algorithm [63]	Assign tasks based on input data location. It delays tasks until it required data is available.	The ease of scheduling. Achieves data locality.	Imbalance load which can cause higher delays in task execution and lower throughput.	Execution time.
Matchmaking algorithm [64]	Before assigning a new task to nodes, every node has a fair chance to utilize its local tasks.	High node utilization. High cluster utilization.	No particular.	Resource utilization. Availability.
Li et al. [28]	Online job scheduling based on data migration based on a trade-off between data transfer cost and the waiting time cost.	Handles data migration.	Schedules tasks sequentially which increases the waiting time. Systems characteristics not considered. Data replication not discussed.	Throughput.
Reddy et al. [59]	Modified Ant Colony optimization.	Considers VMs RAM, bandwidth, storage, processing speed and makespan in the fitness function.	Data-intensive online tasks not addressed.	Resource utilization. Makespan. Load balance.
Biswas et al. [60]	Dynamic round robin.	Dynamically determines Time Quantum.	Starvation not handled.	Turnaround. Waiting time.
Soltani et al. [61]	Genetic meta-heuristic.	Multi-purposed weighted genetic algorithm to enhance performances.	Data-intensive online tasks not addressed.	Response time. Waiting time. Makespan.
Mohseni et al. [62]	Hard Disk Drive and CPU Scheduling (HCS) algorithm.	Schedules multiple tasks among multi-core systems.	Memory, bandwidth and latency of multi-core systems are not considered.	Execution time. Energy consumption.

For further detail on task scheduling techniques, we refer the following reviews: [11–13,20]. The papers primarily focus on examining various scheduling techniques used in cloud computing and presenting a new classification scheme for scheduling algorithms, along with a detailed review of resource scheduling techniques. Additionally, they aim to highlight the advantages and limitations of heuristic, meta-heuristic, and hybrid scheduling algorithms.

3. System Model and Problem Formulation

3.1. System Model

The target system of this study is represented by a heterogeneous machine that consists of two processing types: task processing and data processing.

The machine characteristics and their notations are listed as follows:

- $M = \{m_i\}$ a set of machines, where m_i designates the ith machine.
- sc_i is an integer value that represents the *storage capacity* of machine m_i (MB).
- r_i is a float value that represents the *read speed* of machine m_i (MB/second).
- w_i is a float value that represents the *write speed* of machine m_i (MB/second).
- $RAM[m_i]$ is an integer value that represents the available *memory capacity* of machine m_i (MB).
- $N\_CPU[m_i]$ is the *number of cores* of machine m_i.
- $P\_CPU[m_i]$ is an integer value that represents the *CPU performance* of each core of machine m_i (Million Instructions per second— MIPS).
- b_{ij} is an integer value that represents the *bandwidth of the connection* between machines m_i and m_j (MB/second).
- β_{ij} is the *elementary data transfer time* [68] between machines m_i and m_j, it is defined by:

$$\beta_{ij} = \begin{cases} 0 & \text{if } i = j \\ \frac{1}{b_{ij}} & \text{otherwise} \end{cases} \quad (1)$$

- $PP[m_i]$ is an integer value that represents the *processing power* of machine m_i (in Million Instructions per second—MIPS). $PP[m_i]$ is the overall CPU amount of m_i and is calculated as follows:

$$PP[m_i] = N\_CPU[m_i] \times P\_CPU[m_i] \quad (2)$$

  where $N\_CPU[m_i]$ is the number of cores of m_i and $P\_CPU[m_i]$ is the CPU performance of every core in m_i.
- TP_i defines the list of tasks in progress in m_i.

Many independent users submit tasks for execution. In this paper, we consider that tasks arrive in an online manner to the servers of the different Cloud data centers. All the online tasks share resources and data over the servers. Since the tasks we are handling are data-intensive, two important factors are associated with each task: required data and resources. Tasks are executed in a non-pre-emptive way. However, each task is defined as follows:

- $T = \{t_i\}$ a set of tasks, where t_i is the ith task;
- l_i an integer value that designates the length of ith task (in Million Instructions—MI);
- $RAM[t_i]$ is an integer value that represents the *memory capacity* required by task t_i (in MB);
- $CPU[t_i]$ is an integer value that represents the *quantity of MIPS* required by task t_i;
- $V[t_i]$ is an integer value that represents the *total size* of all the required datasets by task t_i;
- α_i is the index of the final *machine assignment* (m_{α_i}) of task t_i;
- ω_i is a decimal value that represents the arrival time of t_i;
- UR_{ij} is the CPU utilization ratio to determine whether a machine m_j has a sufficient amount of resources to support a task t_i or not.

As mentioned before, load balancing is a critical aspect to take into consideration when designing any task scheduling algorithm in a way that optimizes resource utilization, maximizes throughput, and minimizes response time. For this, we define the workload of each server as follows:

$$Load[m_i] = \frac{\sum_{t_j \in TP_i} l[t_j]}{PP[m_i]} \tag{3}$$

where $Load[m_i]$ is a percentage rate that indicates either m_i is overloaded or underloaded. $Load[m_i]$ is computed by dividing the total of all tasks that are running in m_i on the processing power $PP[m_i]$.

In our work, we assume that a fixed number of datasets are initially stored on the servers. Each dataset is defined as follows:

- $D = \{d_i\}$ a set of datasets where d_i is the ith dataset;
- v_i an integer value that designates the *volume* of ith dataset (in MB);
- $\Psi = \{\psi_{ij}\}$ is the *datasets to machines assignment* matrix. Equation (4) describes the computation of matrix Ψ.

$$\psi_{ij} = \begin{cases} 1 & \text{if } d_i \text{ is stored in } m_j \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

- $F = \{f_{ij}\}$ is the assignment of the datasets to tasks matrix. We set matrix F because a task may require one or multiple datasets for its execution and many tasks may use the same dataset. Matrix F is generated following Equation (5).

$$f_{ij} = \begin{cases} 1 & \text{if } d_i \text{ is required by } t_j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

For a given dataset d_i, there could be two options of use. (1) The local use is when the dataset and its consumer task are on the same node, in that case, the dataset is locally accessed. (2) The remote use is when the required dataset is stored in a different node than the one hosting the task; in that case, data migration is needed from a distant source. We can clearly see that due to the migration process, the execution time of the consumer task is affected by adding a data migration time DMT, where DMT_{ij} is the time to migrate all the datasets required by t_i from their locations to m_j (m_j is also where t_i is assigned) [68].

Our proposed Online Task Scheduling strategy based on Data Migration and Data Replication *(OTS-DMDR)* aims to select online tasks from the queue and schedule them to the appropriate server to ensure better response time as well as a load-balanced system. The task response time includes two main factors: the performance of the resources and, more critically, the management of data regarding their location, movement, and replication within the system.

In addition, the proposed *(OTS-DMDR)* technique is a generic algorithm that could be easily extended to handle different types of data. Mainly, we can use a data adapter component to integrate heterogeneous data types (text, images, logs, videos, etc.) that could be generated by different devices, such as the one used in IoT, financial institutions, and healthcare areas [69]. The next section explains in detail and illustrates the benefit of our approach.

3.2. Problem Formulation

The challenge is how to distribute incoming tasks among servers reduce task response time while avoiding overloaded or underloaded servers. Since data migration requires time, it is obvious that we should seek data locality for tasks as much as possible in order to decrease the response time [1]. When data are migrated to new locations, this will generate new copies of data over the system, called replicated data. In general, data

replication increases the availability of data, thereby achieving more data locality and reducing response time for the next incoming tasks.

We re-examine the issue and discover that the tasks in the queue that must be chosen are the ones that would be carried out on a suitable server with the best response time. In our algorithm, *OTS-DMDR* , the scheduling result combines the data locality method, the data migration method, and the delay scheduling. In other words, the result generates: data locality, i.e., the task will be placed directly in the server containing all its required data; or the task will be placed in a remote server that yields a minimal data migration time; or the task will be delayed until another server having the best response time, via data locality or data migration, becomes available. Simultaneously, the machine load is also taken into account in the *OTS-DMDR* technique to increase the effectiveness of the entire system.

To better illustrate the *OTS-DMDR* technique, we give an example in Figure 1. In Figure 1a, we depict the system configuration. Q is the queue of online tasks. F is the matrix of the assignment of the datasets to the task and Ψ is the matrix of the assignment of datasets to machines.

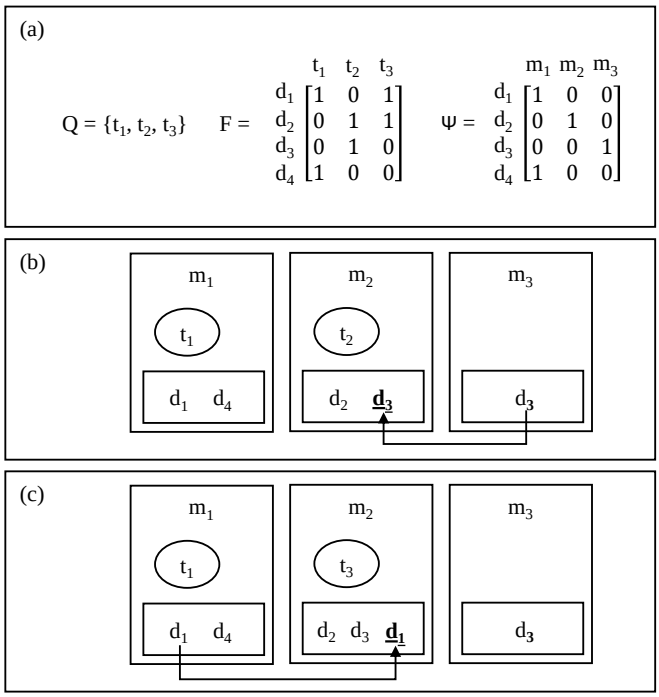

Figure 1. Example to model our proposed scheduling technique OTS-DMDR, where (**a**–**c**) depict respectively the configuration of the system, first iteration of the execution and second iteration of the execution.

According to the *OTS-DMDR* method, machine m_1 is determined to be the optimal choice for task t_1 as shown in Figure 1b, since it achieves perfect data locality with the required datasets d_1 and d_4 already stored on m_1. Similarly, for task t_2, the *OTS-DMDR* method selects machine m_2 as the most efficient solution, as in Figure 1b. Therefore, executing t_2 on m_2 will result in the shortest response time due to the locally stored required data d_2 and the minimal migration time to migrate the required data d_3 from m_3 to m_2. As a result, tasks t_1 and t_2 can be executed at the same time (in parallel). In addition, a replication of d_3 is created in m_2.

Finally, the *OTS-DMDR* algorithm estimates the response time of task t_3 on all machines. Using this approach, the algorithm suggests that it is preferable to delay the execution of t_3 until machine m_2 becomes available. This delay is represented by a time interval denoted as Δ. Despite the delay, executing t_3 on m_2 is expected to result in a lower

response time compared to assigning t_3 to other machines that would require greater data migration times. This is shown in Figure 1c.

It is important to note that in the case of limited computational power of a machine due to different causes (lack of memory, lack of storage, overload of cpu, etc.), the proposed *OTS-DMDR* algorithm, as we will see in Section 4.1.1, proceeds by either skipping that machine for another one that could host the current task, or delaying the task's execution until that machine becomes available to host the current task.

3.3. Objective Function

In this section, we design a mathematical formulation for our proposed algorithm *OTS-DMDR*. Our objective function seeks an efficient task scheduling that minimizes the task response time while maintaining a balanced load of the nodes. The response time is the time required for each task to complete its execution from the moment it arrives in the queue. The value is a combination of the following metrics (see Figure 2):

- Scheduling Time (ST): the time between the arrival of the task in the queue and its scheduling.
- Delay Time (Δ): the time that a task can wait for the availability of a given machine.
- Waiting Time (WT): the sum of scheduling time (ST) and delay time (Δ).
- Data Migration Time (DMT): the time a task needs to locally gather all its remote required datasets.
- Data Access Time (DAT): the time it takes for a task to read all its local required datasets.
- Execution Time (ET): the time to execute the task.
- Total Execution Time (TET): the sum of data migration time (DMT), data access time (DAT), and execution time (ET).
- Response Time (RT): the sum of waiting time (WT) and total execution time (TET).

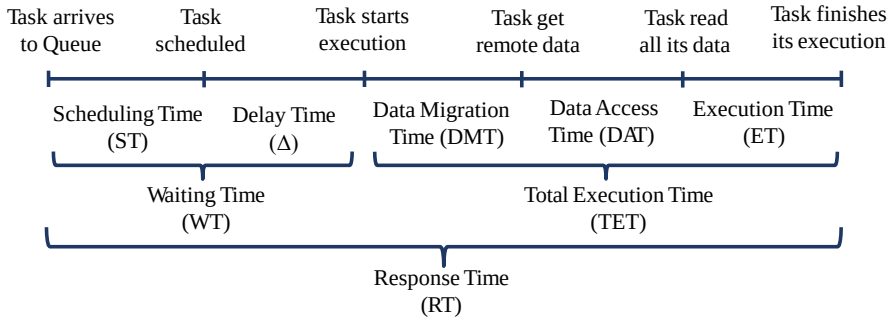

Figure 2. Response Time Scheme.

The problem of reducing the response time of a task t_i when scheduled in m_j can be formulated as:

$$\begin{aligned} \min RT_{ij} &= \min \left(WT_{ij} + TET_{ij}\right) \\ &= \min \left(ST_{ij} + \Delta_{ij} + DMT_{ij} + DAT_{ij} + ET_{ij}\right) \end{aligned} \tag{6}$$

The constraints related to our objective function are shown in Equations (7)–(9).

$$\text{s.t.} \quad RAM[t_i] \leq RAM[m_j] - \sum_{t_k \in TP_j} RAM[t_k] \tag{7}$$

$$\sum_{l=1}^{D} v_l \times f_{li} \leq sc_j - \sum_{t_k \in TP_j} V[t_k], \quad \text{if } \Psi_{lj} = 0 \tag{8}$$

$$Load_{min} \leq Load_j \leq Load_{max} \tag{9}$$

The constraint from Equation (7) guarantees that the remaining amount of RAM in m_j exceeds the requested amount of RAM required by task t_i (TP_j is the list of tasks running in m_j). The constraint from Equation (8) ensures there is enough storage in m_j to store the required datasets for t_i in the case of data migration (t_k is in progress in m_j) and when d_l is a remote dataset. Finally, the constraint from Equation (9) assures the load balancing of the system in such a way that the load of machine m_j should be comprised between two thresholds ($Load_{min}$ and $Load_{max}$) in order to avoid (resp.) underload and overloaded nodes.

For simplicity, we will set the default value of $Load_{max}$ at 70% CPU utilization and the default $Load_{min}$ value at 20% CPU usage [70] for the remainder of the paper.

4. Proposed Approach

In this section, we explain the main steps of our suggested task scheduling strategy *OTS-DMDR*, which selects a set of tasks from the queue and schedule them in machines with the optimal response time.

Our approach consists of the following four steps (see Figure 3):

1. Estimate the response time matrix for the incoming tasks in the queue for all machines;
2. Generate a preference list for task-to-machine assignment;
3. Perform task selection and assignment;
4. Update system state (the availability of the machines and the tasks in the waiting queue Q).

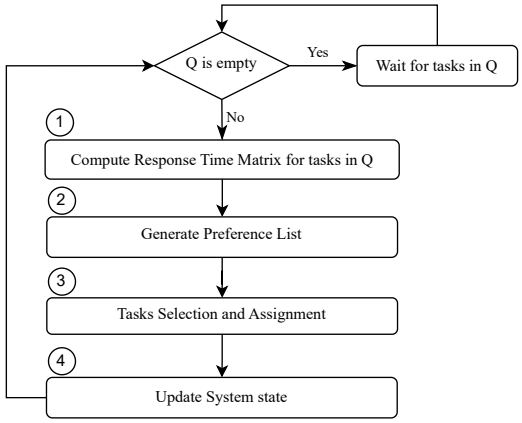

Figure 3. Flowchart of our proposed scheduling strategy.

The steps above are repeated for tasks in the queue. Based on the waiting time of the task, the data migration time, the overall execution time, and the load of machines, a set of tasks will be selected from the queue and will be assigned to servers that best fit.

4.1. Response Time Matrix

The idea of our proposed task scheduling strategy is to choose a set of tasks from the waiting queue and assign each of them to the most appropriate server. In other words, we select the appropriate hosting tasks for each server. This method not only allows us to efficiently use all available servers but also, simultaneously schedule multiple tasks instead of scheduling them task by task.

In order to select tasks and assign them to the most suitable servers, we first have to compute the response time matrix RT for each task in the queue for all the machines. The RT matrix contains the response time RT_{ij} of each task t_i if it is assigned to machine m_j.

Figure 4 and Algorithm 1 show how the response time matrix is computed in detail. For each task t_i in the queue Q, we go through the set of servers in order to estimate RT_{ij}, the response time of the task t_i if assigned to m_j. First, we check if m_j can host t_i by computing the fitness value using the method *MachineFitTask*, as shown in line 9 in Algorithm 1 and step 1 in Figure 4. Then, we compute the response time RT_{ij} based on four main costs:

- Waiting time WT_{ij}, which includes both the delay time Δ_{ij} and the scheduling time ST_{ij} (line 20).
  - Scheduling time ST_{ij} is how much time t_i waits in the queue to be scheduled in m_j.
  - Delay time Δ_{ij} is how much time t_i can wait for m_j to be available. It is measured using the *ComputeDelayTime* function (line 13). Δ_{ij} is computed only if m_j will suit t_i later (see also step 2 in Figure 4). More details are explained in Section 4.1.4.
- Time to migrate required remote data DMT_{ij} is computed when there is no data locality for a given required dataset (line 17 and step 3). Function *ComputeDataMigrationTime* is detailed in Section 4.1.2. Otherwise, if all the required datasets are locally available, $DMT_{ij} = 0$.
- Time to access required data locally DAT_{ij} (step 4 in Figure 4) is computed by calling the *ComputeDataAccessTime* function in line 18. This step aims to measure the needed time to consume the datasets already available locally and the one that has just been gathered via the migration process. More information is depicted in Section 4.1.3.
- Time to execute t_i in m_j defined by ET_{ij} as shown at line 19 in Algorithm 1 and step 5 in Figure 4.

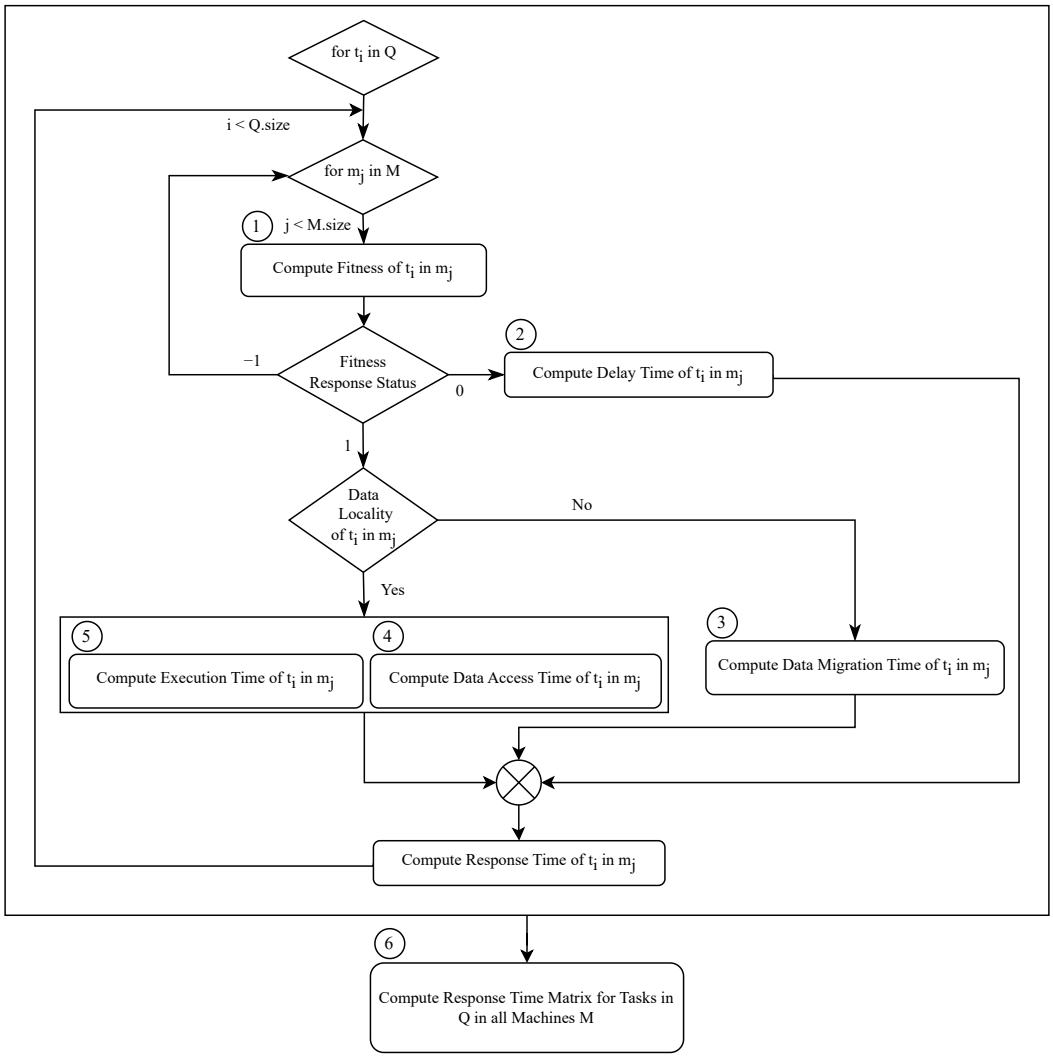

Figure 4. Flowchart of computing response time matrix for incoming tasks in the queue.

Algorithm 1 Compute Total Response Time Matrix for Each Task in the Queue Q

Input:
1: $Q = (t_1, t_2, \ldots, t_n)$: Queue of arrived tasks
2: $M = (m_1, m_2, \ldots, m_p)$: Set of machines
Output:
3: RT_{ij}: Response time of t_i if placed in m_j, where $1 \leq i \leq |Q|$ and $1 \leq j \leq |M|$
4: **if** Q is empty **then**
5: Wait for tasks to arrive to Q (see Figure 3)
6: **else**
7: **for** $(i \in Q)$ **do**
8: **for** $(j \in M)$ **do** *// assume t_i will be placed in m_j*
9: $\phi_{ij} \leftarrow$ MACHINEFITTASK(i, j)
10: **if** $(\phi_{ij} = -1)$ **then**
11: *exit*; *// m_j can't host t_i, move to the next machine*
12: **else if** $(\phi_{ij} = 0)$ **then**
13: $\Delta_{ij} \leftarrow$ COMPUTEDELAYTIME(i, j)
14: **else if** $(\phi_{ij} = 1)$ **then**
15: $\Delta_{ij} \leftarrow 0$
16: **end if**
17: $DMT_{ij} \leftarrow$ COMPUTEDATAMIGRATIONTIME(i, j)
18: $DAT_{ij} \leftarrow$ COMPUTEDATAACCESSTIME(i, j)
19: $ET_{ij} \leftarrow \frac{l_i}{PP_j} + DAT_{ij}$
20: $WT_{ij} \leftarrow ST_{ij} + \Delta_{ij}$
21: $RT_{ij} \leftarrow DMT_{ij} + DAT_{ij} + ET_{ij} + WT_{ij}$
22: **end for**
23: **end for**
24: $PL \leftarrow$ GENERATEPREFERENCELIST(RT)
25: $(Q, \alpha) \leftarrow$ SELECTTASKS(M, Q, PL)
26: **end if**

Afterward, we have a matrix of response time RT (step 6, Figure 4) of all the tasks in the queue, assuming that they are executed in all the machines of the system. The RT matrix will be the basis for our scheduling scheme. Based on matrix RT, the *OTS-DMDR* algorithm generates a preference list PL for the task-to-machine assignment (line 24, Algorithm 1) and is better detailed in Section 4.2. Therefore, a set of tasks is selected to be scheduled in the appropriate servers using method *SelectTasks* (in line 25). The tasks' selection process is described in Section 4.3.

4.1.1. Fitness

The fitness calculation algorithm defines whether the chosen machine m_j is adequate and fit for the execution of task t_i (as shown in Figure 4, step 1). If m_j cannot host t_i (m_j does not fit t_i), m_j is directly discarded. In our work, we consider several metrics to say that t_i can be assigned to m_j or that the fitness of m_j to t_i is achieved. The fitness metrics are: the amount of RAM, storage capacities, and CPU utilization rate (UR). UR_{ij} is the CPU usage rate of t_i in m_j and is computed using Equation (10). $PP[m_j]$ is the processing power of m_j and is calculated using Formula (2).

$$UR_{ij} = \frac{CPU[t_i]}{PP[m_j]} \tag{10}$$

The last metric is the machine load, which determines if the machine is overloaded or underloaded. The load is calculated by Equation (11).

$$Load_j = \frac{\sum_{t_I \in TP_J} l_i}{PP[m_j]} \tag{11}$$

Algorithm 2 depicts how the fitness of task t_i in machine m_j is computed. There are three response states:

1. $\phi_{ij} = -1$, if the utilization ratio is more than 1 (lines 5 and 6).
2. $\phi_{ij} = 1$, if and only if the utilization rate does not exceed 1, when the remaining storage capacity in m_j can accommodate the total amount of required data by t_i (line 9 to 15). The load of m_j must be between the $Load_{min}$ and $Load_{max}$ thresholds (line 16 to 25), and the amount of remaining RAM in m_j should be greater than $RAM[t_i]$.
3. $\phi_{ij} = 0$, if one or more of the conditions above are not verified, i.e., m_j does not have enough CPU or/and not enough RAM to host t_i, or/and the storage capacity of m_j cannot store the remote required datasets of t_i or/and m_j is overloaded or underloaded.

The fitness status obtained from Algorithm 2 is returned to the main Algorithm 1 for processing the three different cases of compatibility (fitness):

- $\phi_{ij} = -1$, the machine m_j cannot host the task t_i due to the lack of CPU, and no action can be taken. We start by checking this first case, so we can know from the beginning if we can continue the process of calculating the response time. In this case, the scheduler moves to the next machine.
- $\phi_{ij} = 0$, the machine m_j cannot host the task t_i due to insufficient RAM or/and storage or/and m_j being overloaded or underloaded. The peculiarity here is that the task can be delayed and wait for these conditions to be verified and accomplish the fitness on m_j. In this case, we talk about **delay scheduling** technique. Task t_i can wait for a delay Δ_{ij} so that the resources of m_j become available again to host t_i. The measurement of the delay time Δ will be explained in Section 4.1.4.
- $\phi_{ij} = 1$, the machine m_j can host the task t_i without constraint violations and delay time.

We would like to mention that in the case where the storage of m_j is not enough to store the required datasets of t_i, we select a set of datasets to delete from m_j. For that, we use our previous work [68], based on data replication, for data selection and deletion processes. The idea is based on two factors:

1. **Dependency between tasks and datasets** ($depend_k$): this factor seeks to define how many duplicated datasets in m_j are required for the uncompleted tasks in the queue. In other words, we compute how many tasks in Q are using every replicated dataset in m_j.
2. **Number of existing replicas of the dataset** ($repl_k$): this factor attempts to define how many replicas of each dataset d_k are currently available in the whole system. Therefore, we check if each machine m_j stores d_k as a replica copy. The value of $repl_k$ is raised by one each time a replica of d_k is identified.

Due to the possibility of multiple replications, only datasets with more than $maxRep$ replicas (here equals three) are qualified for deletion.

We select d_i with the lowest $depend_k$ (the least used d_i from the unfinished tasks). If there are multiple datasets with the same $depend$ factor, we take the one with the highest $repl$ into consideration. Based on this, we delete the datasets one after another until the deleted space is greater than the requested size, liberating the storage needed by the datasets that will be migrated for t_i execution.

Algorithm 2 Compute Fitness of m_j to host t_i

Input:
1: i: index of task for which fitness is checked
2: j: index of machine whom we check the placement fitness
Output: ϕ_{ij}: fitness status (1,0,−1)
3: **function** MACHINEFITTASK(i, j)
// CPU Utilization Ratio measurement
4: $UR_{ij} \leftarrow \frac{CPU[t_i]}{PP[m_j]}$
5: **if** $(UR_{ij} > 1)$ **then**
6: $\phi_{ij} \leftarrow -1$
7: exit;
8: **else**
// Storage capacity verification
9: $V[t_i] \leftarrow 0$
10: **for** $(k \in D)$ **do**
11: $V[t_i] \leftarrow V[t_i] + f_{ki} \times v_k$ *// total remote data size required by* t_i
12: **end for**
13: **if** $(sc_j \geq V[t_i])$ **then**
14: $selectedDatasets \leftarrow$ SELECTDATASETSTODELETE($i, V[t_i]$)
15: **end if**
// Load measurement
16: $Load_j \leftarrow 0$
17: **for** $(k \in TP_j)$ **do** *// search tasks in progress in* m_j
18: $Load_j \leftarrow Load_j + \frac{l_k}{PP[m_j]}$
19: **end for**
20: **if** $(Load_j \geq Load_{max})$ **then** *//* m_j *is overloaded*
21: $m_j.overloaded = 1$
22: **end if**
23: **if** $(Load_j \leq Load_{min})$ **then** *//* m_j *is underloaded*
24: $m_j.underloaded = 1$
25: **end if**
// Fitness Measurement
26: **if** $(\sum_{t_k \in TP_j} RAM[t_k] \geq RAM[t_k]$ && m_j.overloaded = 0 && m_j.underloaded = 0) **then**
27: $\phi_{ij} \leftarrow 1$
28: **else**
29: $\phi_{ij} \leftarrow 0$
30: **end if**
31: **end if**
return ϕ_{ij}
32: **end function**

4.1.2. Migration Time

Once the fitness of scheduling a proper task t_i in a proper machine m_j is calculated, we can now start computing the migration time DMT_{ij} in order to estimate the response time of t_i in m_j.

Algorithm 3 is used to compute the time needed to migrate the remote required datasets of task t_i from their remote locations to m_j. There are two potential issues in calculating the migration time. The first is that t_i may need one or more datasets to migrate. The second is that multiple replicas may exist for a single dataset. In Algorithm 3, the block between line 5 and line 16 describes how to solve these two issues.

Algorithm 3 Compute required remote Datasets Migration Time of t_i placed in m_j

Input:
1: i: index of task for which we will estimate the needed time to migrate its required data
2: j: index of machine we assumed t_i will be scheduled and data will be migrated to
Output: DMT_{ij}: Data Migration Time of the remote required datasets of t_i from their distant locations to the local node m_j where t_i is scheduled
3: **function** COMPUTEDATAMIGRATIONTIME(i, j)
4: $DMT_{ij} \leftarrow 0$
5: **for** ($k \in D$) **do**
6: **if** ($f_{ki} = 1$) **then** // d_k *is required by* t_i
7: $\tau_{ij}^{kj} \leftarrow 0$
8: **if** ($\psi_{kj} = 0$) **then** // d_k *is a remote data*
9: $l \leftarrow 0$
10: **for** ($l \in M - \{m_j\}$) **do**
11: $\tau_{ij}^{kl} \leftarrow 0$
12: **if** ($\psi_{kl} = 1$) **then** // d_k *is stored in* m_l
13: $\tau_{ij}^{kl} \leftarrow (\frac{1}{r_l} + \frac{1}{w_j} + \frac{1}{b_{lj}}) \times v_k$ // *time to migrate* d_k *required by* t_i *from distant* m_l *to local* m_j
14: **end if**
15: **end for**
16: **end if**
// Sort migration times of all machines of each d_k $\tau_{ij}(k,:)$ *in ascending order*
17: $[\sigma_{ij}(k,:)] \leftarrow sort(\tau_{ij}(k,:))$ *// $\sigma_{ij}(k,q) = l$, i.e., d_k is migrated from m_l with time of* τ_{ij}^{kl}
18: $s \leftarrow \sigma_{ij}(k,0)$ // m_s *is the machine source with least migration time to move* d_k *to* m_j
19: $DMT_{ij} \leftarrow DMT_{ij} + \tau_{ij}^{ks}$ // *data migration time*
20: **end if**
21: **end for**
return DMT_{ij}
22: **end function**

For each dataset, we check if d_k is required by t_i (line 6) and if d_k is not stored locally in m_j (line 8). In this case, the migration of d_k is required by finding all its locations, calculating the time needed to migrate d_k from each of its locations to m_j, and finally selecting the location m_l with the smallest migration time. Line 13 shows how to calculate the time to migrate d_k from one of the found locations m_l to the local node m_j. This migration time is denoted by τ_{ij}^{kl}.

In fact, the migration depends on the size of the data (v_k) and consists of three processes: (1) reading d_k from the remote node m_l with a read speed of r_l, (2) writing d_k to the local node m_j with a write speed of w_j, and (3) transferring d_k from m_l to m_j via a bandwidth with a transfer rate of b_{lj}.

For now, for each data d_k required by t_i not achieving the data locality, we have its migration time $\tau_{ij}(k,:)$ from all its existing locations to the local machine m_j. The next step is to select the best location from which d_k will be migrated. To do this, we sort the vector $\tau_{ij}(k,:)$ (line 17) into ascending order and pick the first element $\sigma_{ij}(k,0)$, which gives the best machine m_s providing d_k with the lowest migration time τ_{ij}^{ks} (line 18).

Before moving on to the next dataset, the value τ_{ij}^{ks} is added to the DMT_{ij} value (line 19), where DMT_{ij} is the total time needed to migrate all the required remote datasets of t_i affected to m_j.

Finally, after browsing all the remote data and computing their migration time from their best location, the total value of DMT_{ij} is detained for use in Algorithm 1 at line 17.

4.1.3. Data Access Time

It is mandatory that a task accesses and consumes its required data in order to complete its execution, otherwise the task fails. The access time for the local consumption of all data is designed by DAT_{ij} as indicated in Algorithm 4. Once DAT_{ij} has been calculated, its value is returned to Algorithm 1 so that it is taken into account in the response time RT_{ij}.

Algorithm 4 Compute Data Access Time of t_i placed in m_j

Input:
1: i: index of task for which we will estimate time to locally access its required data
2: j: index of machine we assumed t_i will be scheduled at
Output: DAT_{ij}: Data Access Time of all the required datasets of t_i in the local node m_j
3: **function** COMPUTEDATAACCESSTIME(i, j)
4: $DAT_{ij} \leftarrow 0$
5: **for** ($k \in D$) **do**
6: **if** ($f_{ki} = 1$) **then** // d_k *required by* t_i
7: $DAT_{ij} \leftarrow DAT_{ij} + \frac{v_k}{r_j}$
8: **end if**
9: **end for**
return DAT_{ij}
10: **end function**

4.1.4. Delay Time

As mentioned previously, it is possible that a given machine m_j does not fit t_i due to insufficient storage space, RAM, or load of the machine. This incompatibility might be solved if the execution of the task is postponed. This type of scheduling is called **Delay Scheduling** .

The proposed *OTS-DMDR* technique is based on the delay method, which could lead to a better response time. Algorithm 5 employs the delay scheduling, which will allow the computation of the delay time (Δ_{ij}) for the task t_i until the resources of machine m_j are available again.

The measurement of the Δ_{ij} is conducted as follows. First, we sort the tasks in the machine m_j by their estimated finish time in ascending order. The sorting result is in a sorted queue designated by Q'_j (line 4, Algorithm 5). Then, we go through each task in Q' to verify when the fitness of t_i will be achieved (line 9). For each task t_k in Q'_j, we obtain its remaining execution time (RET_k). RET_k is added to the delay time Δ_{ij} (line 11); then, the RAM, storage capacity, and load of m_j are updated in order to add the value consumed by t_k (line 12 to line 14). The goal is to check if this updated state will allow to free more RAM and/or storage and/or load on the machine m_j so that it receives the concerned task t_i.

The process is repeated until the fitness of m_j and t_i is achieved (line 10). Finally, we receive the exact delay time Δ_{ij}, which will be considered subsequently in the response time of task t_i in m_j in the main Algorithm 1 at line 13.

Algorithm 5 Compute Delay Time

Input:
i: index of task for which fitness is checked
j: index of machine whom we check the placement fitness
Output: Δ_{ij}: Delay time so that m_j is available to host t_i
function COMPUTEDELAYTIME(i, j)
  $Q'_j \leftarrow sort(TP_j)$ *// sort in ascending order by estimated finish time the tasks in m_j // or by arrival time of assigned tasks to m_j*
  $\Delta_{ij} \leftarrow 0$
  $newRAM[m_j] \leftarrow RAM[m_j]$
  $newSc_j \leftarrow sc_j$
  $newLoad_j \leftarrow Load_j$
  for ($k \in Q'_j$) **do**
    while (m_j.overloaded = 1 $\|$ m_j.underloaded = 1 $\|$ $newRAM[m_j] < RAM[t_i]$ $\|$ $newSc[m_j] < V[t_i]$) **do**
      $\Delta_{ij} \leftarrow \Delta_{ij} + RET_k$
      $newRAM[m_j] \leftarrow newRAM[m_j] + RAM[t_k]$
      $newSc_j \leftarrow newSc_j + V[t_k]$
      $newLoad_j \leftarrow newLoad_j + \frac{l_k}{PP[m_j]}$
      if ($newLoad_j \leq Load_{max}$) && ($newLoad_j \geq Load_{min}$) **then**
        $m_j.overloaded = 0$
        $m_j.underloaded = 0$
      end if
    end while
  end for
return Δ_{ij}
end function

4.2. Task to Machine Preference List

So far, we have been able to compute the response time RT matrix. In the current work, we aim to efficiently select a set of incoming tasks and assign them to the appropriate servers. Hence, we propose a preference list PL that generates potential association between tasks and the available machines.

To generate the preference list PL, we sort the elements of the matrix RT in ascending order. The elements of PL are represented by a triplet of task t_i, machine m_j, and their corresponding response time RT_{ij}, as follows:

$$PL = \{pl_k\} = \left[(t_i, m_j, RT_{ij})\right] \tag{12}$$

where the first element (plL_1) of the list PL is the lowest response time if we assign t_i to m_j. To better understand the process, we give an example in Figure 5.

$$RT = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{array} \begin{array}{c} \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \begin{bmatrix} 5 & 3 & 4 & 6 \\ 4 & 2 & 1 & 2 \\ 3 & 6 & 9 & 5 \\ 7 & 5 & 8 & 3 \\ 8 & 4 & 5 & 1 \end{bmatrix} \end{array} \rightarrow RT_{23} = 1 \text{ is the response time of executing } t_2 \text{ in } m_3$$

$$\Longrightarrow PL = \begin{array}{l} [\ (t_2,m_3,1),\ (t_5,m_4,1),\ (t_2,m_2,2),\ (t_2,m_4,2),\ (t_1,m_2,3), \\ (t_3,m_1,3),\ (t_4,m_4,3),\ (t_1,m_3,4),\ (t_2,m_1,4),\ (t_5,m_2,4), \\ (t_1,m_1,5),\ (t_3,m_4,5),\ (t_4,m_2,5),\ (t_5,m_3,5),\ (t_1,m_4,6), \\ (t_3,m_2,6),\ (t_4,m_1,7),\ (t_4,m_3,8),\ (t_5,m_1,8),\ (t_3,m_3,9)\ ] \end{array}$$

Figure 5. Example of generating the preference list.

The matrix RT gives the PL where the best assignment is represented by the lowest value $RT_{23} = 1$ when t_2 will be scheduled in m_3, followed by placing t_5 in m_4. While the worst assignment is the highest value 9, which happens if t_3 is affected by m_3 for the execution. Hence, in the following subsection, we present an efficient technique to select an optimal assignment task-to-machine based on PL.

4.3. Tasks Selection

In this section, we select the set of tasks that must be scheduled in each of the machines. Since our work is based on online tasks. Therefore, we always have a queue containing tasks that must be executed as soon as possible. For this reason, we have opted for the idea of selecting a set of tasks. Task selection allows us not only to choose the tasks with the best response times but also to take advantage of the use of all the available machines. In this way, we are sure to achieve our goal of minimizing the response time and using the resources efficiently. The selection process of tasks is described in Algorithm 6 and illustrated by Figure 6.

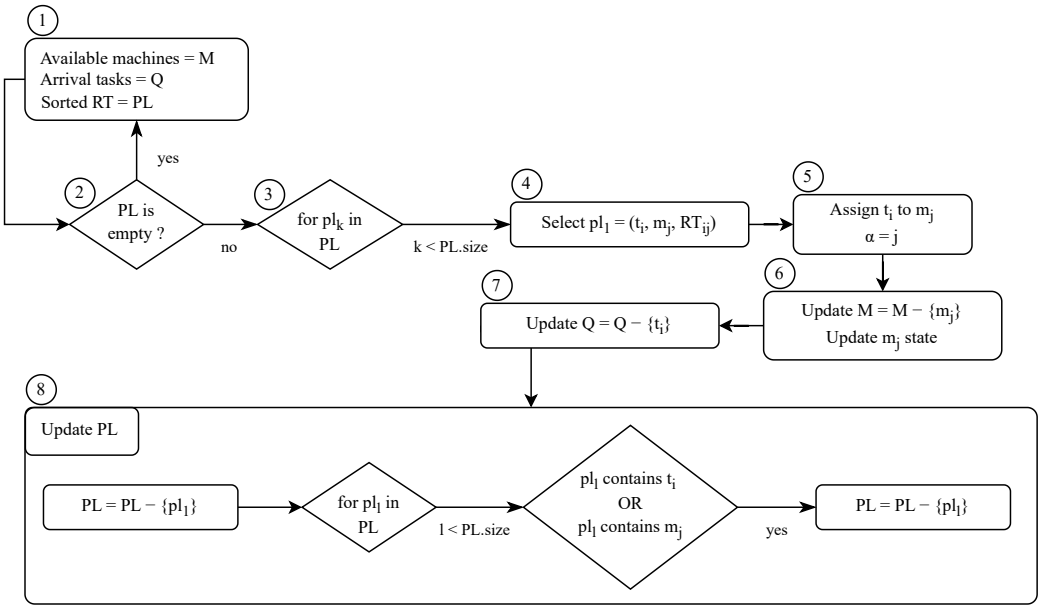

Figure 6. Flowchart of task selection process.

For the task selection procedure, as a first step, we need the available machines M, the arrival tasks in the queue Q, and the preference list PL as input. Then, by going through the preference list PL (step 3 in Figure 6), we select the first element pl_1 (step 4), which has the lowest response time RT_{ij} (line 9, Algorithm 6) and that happens when assigning t_i to m_j. After assigning t_i to m_j (step 5), m_j is marked as the best assignment for t_i as indicated in line 10.

The vector α is used to describe the indices of the final task assignments, i.e., $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_j)$, where α_i is the index of the machine where t_i is assigned. In other words, the best machine to host t_i is m_{α_i}. After that, we perform four updated operations:

1. The available machines M are updated by removing m_j;
2. The characteristics of m_j are updated, i.e., the RAM occupied by t_i is subtracted from the total RAM of m_{α_i} (line 12), then the storage capacity of m_{α_i} is modified by deleting the volume of migrated data required by t_i (line 13) and the used load by t_i is added to the total load of m_{α_i} (line 14);
3. The queue Q of incoming tasks is updated by removing the assigned task t_i;
4. The preference list PL is updated (step 8, lines between 18 and 22) by removing all the triplets concerning the task t_i or the machine m_j.

Updating PL is required to avoid rescheduling an already assigned task and not to use a machine to which we have already assigned a task.

The whole process is repeated until the preference list is empty, which means either no available tasks are in the queue or all the machines were used for the tasks in the queue. In that case, we run the main Algorithm 1 to re-check the queue and repeat the computation of the response time matrix and so on.

To help understand how task selection operates, we illustrate it using the example in Figures 7 and 8.

Algorithm 6 Tasks Selection

Input:
M: Available machines in the system
Q: Arrival tasks in the queue
PL: Preference list issued by sorting RT
Output:
Q: Updated Q
α: Vector of the final assignment of selected tasks

function SELECTTASKS(M, Q, PL)
  while (PL is not empty) **do**
    for ($k \in PL$) **do**
      $selectedPL \leftarrow pl_1$ // $pl_1 = (t_i, m_j, RT_{ij})$ *is lowest response time*
      $\alpha_i \leftarrow j$ // *the best placement of* t_i *is* m_j
      $M \leftarrow M - \{m_j\}$ // *update* M
      $RAM[m_j] \leftarrow RAM[m_j] - RAM[t_i]$
      $sc_j \leftarrow sc_j - V[t_i]$
      $Load_j \leftarrow Load_j + \frac{l_i}{PP[m_j]}$
      $TP_j.add(t_i)$
      $Q \leftarrow Q - \{t_i\}$ // *update* M
      $PL \leftarrow PL - \{pl_1\}$ // *update* PL *by removing the 1st element*
      for ($l \in PL$) **do**
        if (($pl_l.contains(t_i)$ || $pl_l.contains(m_j)$)) **then**
          $PL \leftarrow PL - \{pl_l\}$ // *Update* PL *by removing elements with* t_i *or* m_j *in triplet*
        end if
      end for
    end for
  end while
return (Q, α)
end function

The example begins with the input of four available machines M, five incoming tasks in the queue Q, and the preference list PL generated in Figure 5. A first iteration takes effect to assign one of the tasks to the adequate server (see Iteration 1 in Figure 7). First, we select the first element of PL, which is the triplet $(t_2, m_3, 1)$. This triplet provides the lowest response time in matrix RT and allows us to assign t_2 to m_3 with $RT_{23} = 1$. As result, m_3 is removed from the available machines M, t_2 is deleted from the queue Q, and PL is updated by removing all the triplets containing either $t2$ or m_3. All updates are described by cross marks with red color in the output box. A second iteration is conducted by taking as input the updated values from Iteration 1 of the available machines M, Q, and PL. After selecting the first triplet $(t_5, m_4, 1)$ in PL, t_5 is assigned to m_4 with a response time of 1. The updates are completed by removing m_4 from M, t_5 from Q, and all the triplets concerning t_5 and m_4 from PL. In our example, the process is repeated until iteration 4 (see Figure 8) where all the machines were used ($M = \{\varnothing\}$) and PL is empty. In contrast, this case results in a task t_4 that is not assigned and that will be handled when repeating the main process

from computing the matrix *RT* and where new tasks will be added to the arrival queue. Therefore, we can say that our proposed strategy *OTS-DMDR* assigned four incoming tasks out of five while minimizing their response time and maximizing the system resources.

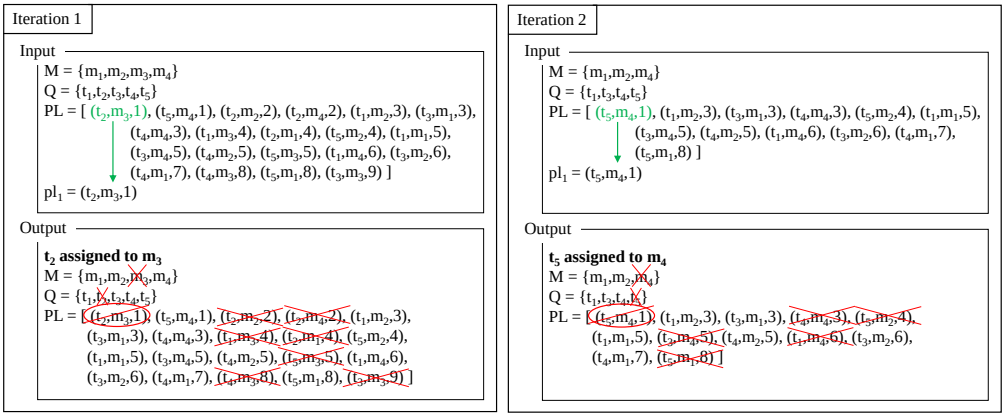

Figure 7. Example of task selection (iterations 1 and 2), where the selected elements are highlited with green color and the deleted elements are highlited with red color.

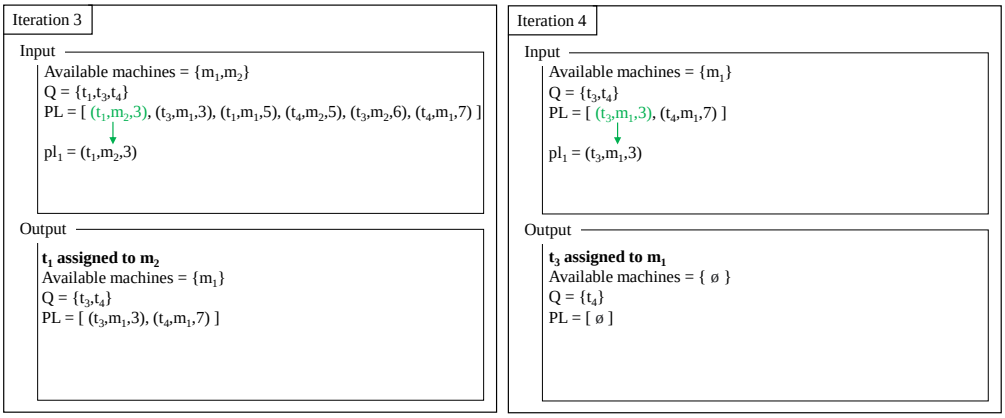

Figure 8. Example of task selection (iterations 3 and 4), where the selected elements are highlited with green color.

5. Simulation Setup and Result Analysis

In this section, we present the experiments performed to assess the effectiveness of the proposed scheduling algorithm *OTS-DMDR*. The following subsections present the performance metrics, the used benchmarks, the experimental setup, and a discussion of the obtained results.

5.1. Simulation Setup

Since the target system is a cloud computing environment, the evaluation of scheduling algorithms is crucial. However, experiments on real cloud platforms would be costly and challenging, especially when it comes to repeating the experiments under the same circumstances in order to compare other algorithms. As a result, a simulator is required to measure the performance of the proposed algorithms. In order to model and simulate cloud-based systems, we used an extensible toolkit CloudSim 3.03 [71,72]. Nevertheless, the Cloudsim framework does not support data management such as data storage, data migration, data replication, and remote data consumption. Due to these limitations, we extended Cloudsim in our previous work [73] so that it can effectively address those needs.

For our experiments, we vary the number of machines between 5 and 100. Each machine is considered with its characteristics (*CPU*, *RAM*, *Storage Capacity*, *Read/Write*

Speed). We also consider a range of tasks between 30 and 2000 tasks, where every task requires at least 1 and at most 10 datasets. The size of datasets is evenly distributed within the range [1–100 GB]. The overall configuration is depicted in Table 2.

Table 2. Setup characteristics.

Characteristic	Value
Number of machines	[5–100]
P_CPU (MIPS)	[1000–5000]
RAM (GB)	[64–2048]
Storage capacity (TB)	[1–25]
Number of tasks	[30–2000]
Size of tasks (MI)	[1000–4000]
Number of datasets	300
Size of datasets (GB)	[1–100]
Number of required datasets	[1–10]

CloudSim offers the flexibility of time-sharing and space-sharing techniques for resource allocation in tasks [71,72]. The appropriate technique can be selected by users depending on their specific requirements, such as performance, cost, and resource utilization, which significantly affects the overall efficiency and performance of cloud computing applications. Our proposed algorithm utilizes the time-sharing technique provided by CloudSim, allowing tasks to be executed in parallel. In time-shared mode, multiple task units, or Cloudlets, can perform multitasking within a machine.

We generate various scenarios, take into account 100 executions for each, and use the average as our final measurement.

We would like to mention that the initial data placement is conducted based on the max–max algorithm [74]. This means, that the data with the largest size is placed in the storage with the maximum remaining storage capacity.

We compare our proposed task scheduling strategy (**OTS-DMDR**) with four other scheduling algorithms: **FCFS** [52], the traditional scheduling algorithm that schedules tasks based on their arrival time, i.e., the first arrived is the first to be executed; **Delay Scheduling** [63], delays the execution of a task in order to assign it to the server achieving the data locality; **Li et al.** method [28], that compromises between waiting time and data migration costs. Finally, we compare *OTS-DMDR* with a proposed algorithm that does not consider the data replication, which we name Online Task Scheduling based on Data Migration **(OTS-DM)**.

5.2. Performance Metrics

To quantitatively evaluate the performance of the *OTS-DMDR* algorithm and compare its effectiveness with other algorithms in the literature, we need to use a variety of metrics, which are listed below.

5.2.1. Response Time (RT)

Needed time for a task to finish its execution. The response time includes the following stages (also see Figure 2):

- Scheduling Time (ST);
- Delay Time (Δ);
- Waiting Time (WT);
- Data Migration Time (DMT);
- Data Access Time (DAT);
- Execution Time (ET);
- Total Execution Time (TET);

5.2.2. Throughput

Number of tasks that can be processed by the whole system within a time slot.

5.2.3. Degree of Imbalance (DI)

Calculates the imbalance across all of the machines using Equation (13).

$$DI = |M| \times \frac{VET_{max} - VET_{min}}{OET} \tag{13}$$

where $|M|$ is the total number of machines. VET_{max} and VET_{min} are (resp.) the maximum and minimum total execution time among all machines and OET is the overall execution time of all machines and is calculated as follows:

$$OET = \sum_{j \in M} VET_j \tag{14}$$

5.3. Result Analysis

5.3.1. Experiment 1: Task Variation

In the first experiment, we aim to measure the impact of varying the number of incoming tasks that arrived into the queue within the same time slot w.r.t. the aforementioned time metrics (response time, migration time, waiting time, etc.). For that, we fix the number of machines M to 100 and the number of required data RD varying within the range of [1–10], while the number of incoming tasks T takes the values 500, 1000, 1500, and 2000.

Figure 9 presents a comparison between the different scheduling algorithms in terms of the average response time RT. Where the x-axis represents the number of tasks and the y-axis is the measured average response time for a given number of tasks.

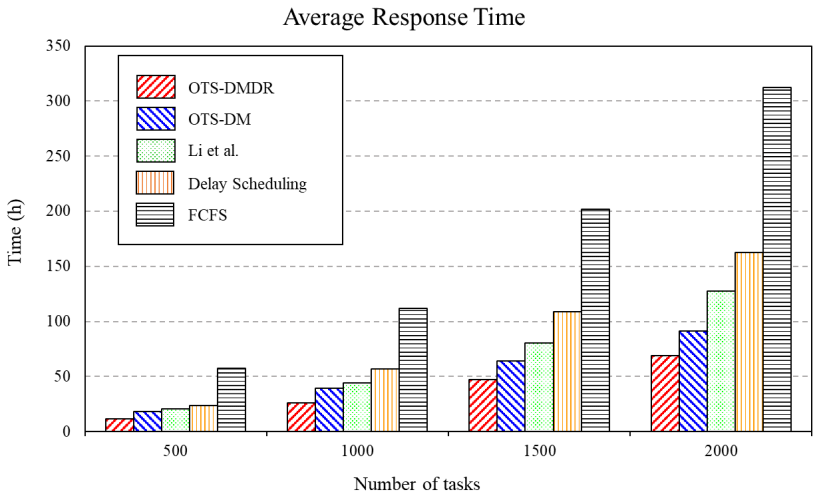

Figure 9. Average Response Time for Task Variation of the proposed methods *OTS-DM* and *OTS-DMDR*, compared to *Li et al.* [28], *Delay Scheduling* [63] and *FCFS* [52].

We can see from the results of Figure 9 that the proposed algorithms *OTS-DMDR* and *OTS-DM* outperformed the rest of the scheduling strategies for all of the test cases, showing a considerable reduction in average response time, particularly for a higher number of tasks (1500 and 2000 tasks). In the meantime, the Li et al. [28] and *delay scheduling* methods have a competitive performance only for 500 and 1000 tasks. Meanwhile, the *FCFS* method exhibits poor performance for all the cases.

To investigate the performance of each method in more detail, we chose the test case of 2000 tasks, then we computed the time spent on each stage (namely, ST, Δ, DMT, DAT, and ET). Figure 10 gives the percentages of each stage for each tested method.

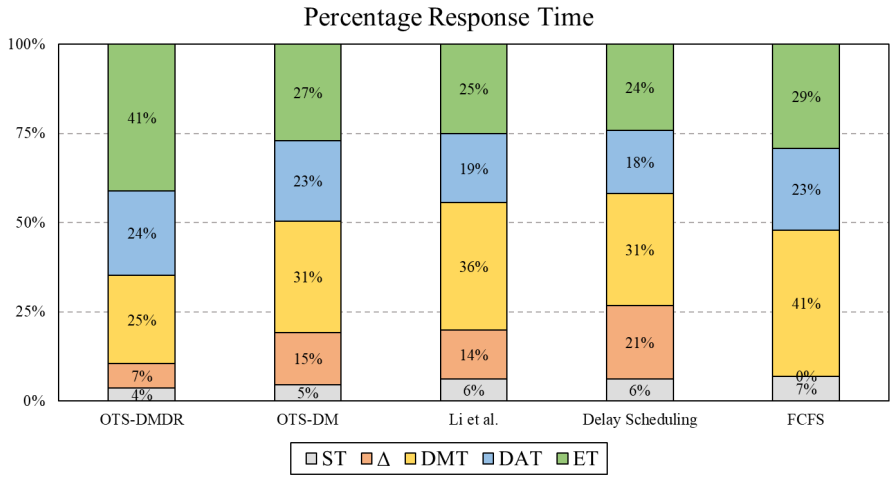

Figure 10. Percentage Response Time for Tasks Variation of the proposed methods *OTS-DM* and *OTS-DMDR*, compared to *Li et al.* [28], *Delay Scheduling* [63] and *FCFS* [52].

The *FCFS* method dedicates more than 41% of the response time to data migration. This is justified by the fact that the *FCFS* method does not consider both data locality and machine performance when scheduling tasks. Moreover, since the *FCFS* method is based on a first come first served stragey to assign the incoming tasks, the time between scheduling and starting the execution of every task is very low, which can be considered 0 ($\Delta = 0$). In contrast, the scheduling time is quite high ($ST = 7\%$) because tasks that arrive may not be immediately scheduled due to the unavailability of machines.

Since the *delay scheduling* method is based on delaying tasks in order to achieve data locality and does not consider any data migration, we can clearly see in Figure 10 that the percentage of waiting time is significant (21%) and helps to gain in terms of local data accesses (representing only 18% of the response time). In addition, to avoid starvation, the migration process takes effect and remote data is not efficiently gathered. Thus, the response time is dominated by data migration by 31%.

The response time of Li et al. [28] is overtaken by both data migration time and execution time with a rate of 36% and 25%, respectively. On the other hand, for the *OTS-DM* method, the data migration rate is decreased to 31%, while the execution time is slightly increased to 27%. The reduction of data migration in the proposed *OTS-DM* method is due to the strategy that chooses the best location from which to pull the data. For *OTS-DMDR*, 41% of the total time is consumed by execution time (ET), while 25% is consumed by data migration. One can recall a huge decrease in terms of *DMT* in comparison with all of the existing strategies; this can be justified by replicating datasets across the machines, as explained in Section 4.1.1.

Finally, from this experiment, we can conclude that the proposed strategy *OTS-DMDR* presents a significant improvement in terms of response time and can be very useful for online task scheduling for big data applications that involve both small and large numbers of tasks. In addition, the decision to schedule some tasks on the most appropriate machine can be determined based on a compromise between data locality and data migration cost, while considering data replication and delay scheduling cost, thus yielding an optimal response time with lower data transfer.

One major advantage of data migration is that it can help to address data accessibility and availability. By having multiple replicas of data across multiple machines, it is possible to leverage multi-task processing and accelerate the execution process. This can be particularly useful for large-scale machine learning problems or for training on big data sets [6].

5.3.2. Experiment 2: Machine Variation

In contrast to the previous experiment, in this scenario, we fix the number of tasks T to 2000, while the number of machines M takes values 25, 50, 100, 250, and 500. The purpose of this experiment is to examine the scheduling behavior of the algorithms under different system configurations.

Figure 11 represents the plotting of the average response time for various numbers of machines corresponding to different scheduling methods. The results demonstrate that our proposed algorithm *OTS-DMDR* outperforms the other scheduling techniques for all of the test scenarios, providing a significant reduction of the average response time, particularly for a higher number of machines (500 machines). The *OTS-DM Li et al.* [28] and *delay scheduling* algorithms currently perform well. The *FCFS* approach, on the other hand, yields consistently poor results.

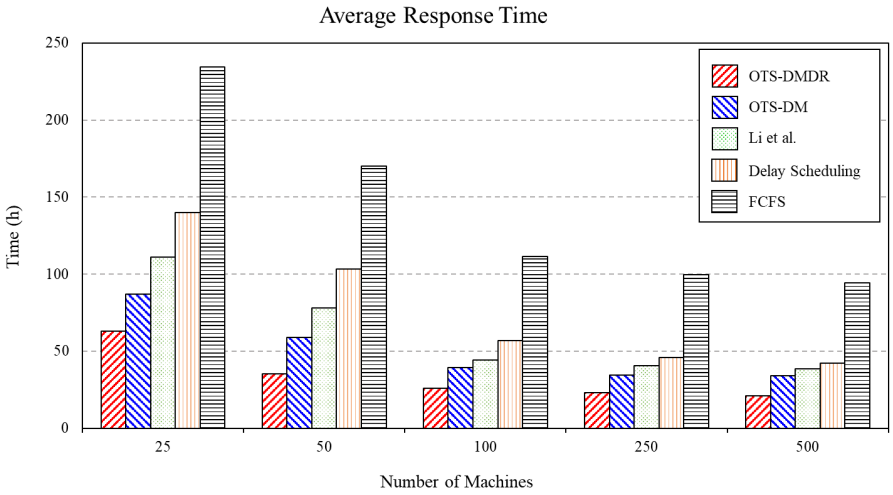

Figure 11. Percentage Response Time for Machine Variation of the proposed methods *OTS-DM* and *OTS-DMDR*, compared to *Li et al.* [28], *Delay Scheduling* [63] and *FCFS* [52].

To further compare the effectiveness of each technique, a detailed experimental analysis is performed in terms of the percentage rate of each metric (WT, DMT, DAT, and ET) compared to the total response time RT. For this, we select the case of 100 machines. In this respect, Figure 12 shows the percentage rate for all the tested algorithms.

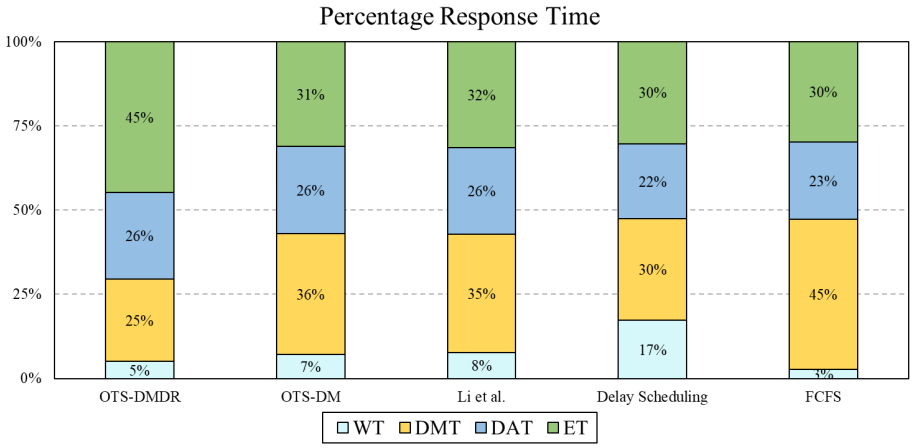

Figure 12. Percentage Response Time for Machines Variation of the proposed methods *OTS-DM* and *OTS-DMDR*, compared to *Li et al.* [28], *Delay Scheduling* [63] and *FCFS* [52].

As can be observed from the results of the *OTS-DMDR* strategy, the execution time took 45% of the total response time, while the percentage of migration time consumed only 25% of the response time. As *OTS-DMDR* is based on a trade-off between optimizing data

locality, delay scheduling, and data migration relying on data replication, this led to a good data migration time rate of 25% as well as a small waiting time (5%) compared to other techniques.

For the *OTS-DM* strategy, the migration time was considered to be the overtaken time (36%). This is a significant difference compared to *OTS-DMDR* because *OTS-DM* does not consider the replicated data while moving it. Li et al. [28] found similar results to the *OTS-DM* strategy for all metrics.

In the *delay scheduling* strategy, the waiting time was greater than the other strategies (17%) since the idea behind the method was based on delaying tasks in order to achieve better data locality (22%). The *DMT* rate still had a noticeable value (30%) in comparison with *OTS-DMDR*.

Finally, for the *FCFS* method, we can clearly see that the data migration time again dominated response time with a percentage of 45%. The reason is that *FCFS* assigns tasks without considering data locality nor data movement.

From the result of Figures 11 and 12, we can notice a strong relationship between the average response time of tasks and the number of machines, as the number of machines increases the average response time decreases. Moreover, *OTS-DMDR* performs competitively with *OTS-DM*, Li et al. [28], and *Delay Scheduling* methods for a higher number of machines, while for the lower number of machines, it is very obvious that the proposed *OTS-DMDR* gives significantly better results than all of the existing algorithms. Eventually, the proposed *OTS-DMDR* algorithm showed sufficient performance to be used as an alternative task scheduling algorithm for big data systems.

5.3.3. Experiment 3: Datasets Variation

In this scenario, we vary the number of required datasets to see how it impacts the total response time. We use three different scales: (**a**) Small number of required data [1–5], (**b**) Medium number of required data [5–10], and (**c**) Large number of required data [10–20]. Figure 13 depicts the plots of the average response time for the three different scales for each scheduling strategy.

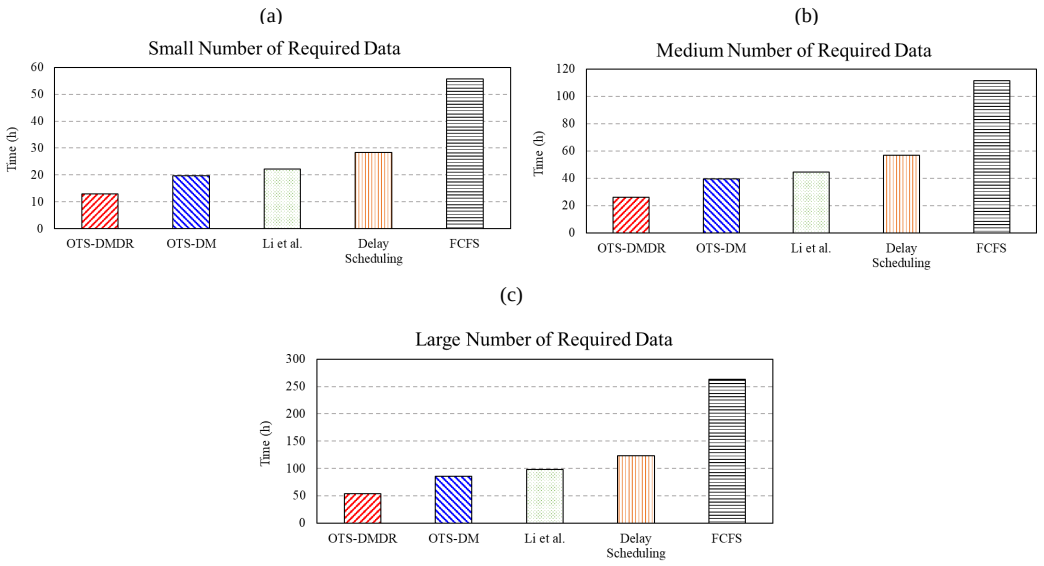

Figure 13. Response Time of Dataset Variation for the proposed methods *OTS-DM* and *OTS-DMDR*, compared to *Li et al.* [28], *Delay Scheduling* [63] and *FCFS* [52], where (**a**) is for small number of required data, (**b**) is for medium number of required data and (**c**) is for large number of required data.

We can see that our proposed algorithm *OTS-DMDR* performed best throughout the three experiments. Furthermore, one can observe a competitive performance between *OTS-DM* and *Li et al.* [28] with a slight advantage of the proposed *OTS-DM*. However, both *delay scheduling* and *FCFS* methods gave higher response times for all scales.

The corresponding percentage response times of this experiment are reported in Figure 14a–c. We observe that, even though the number of required data changed, our method *OTS-DMDR* consistently gave the least *DMT* percentage rate. While a comparable performance was observed between *OTS-DM* and *Li et al.* [28] in the three scales. For the *delay scheduling*, the *WT* percentage was higher compared than the other methods. Similar to previous results, the *FCFS* response time was mostly spent migrating data.

Our method, *OTS-DMDR*, consistently produces the least *DMT* percentage rate regardless of the changing data requirements. This is especially due to considering the replication of data, which creates new copies of data in the system. As a result, scheduling tasks based on data migration and reusing replicated data can offer benefits such as enhanced data availability, improved data locality, and decreased response time for incoming tasks.

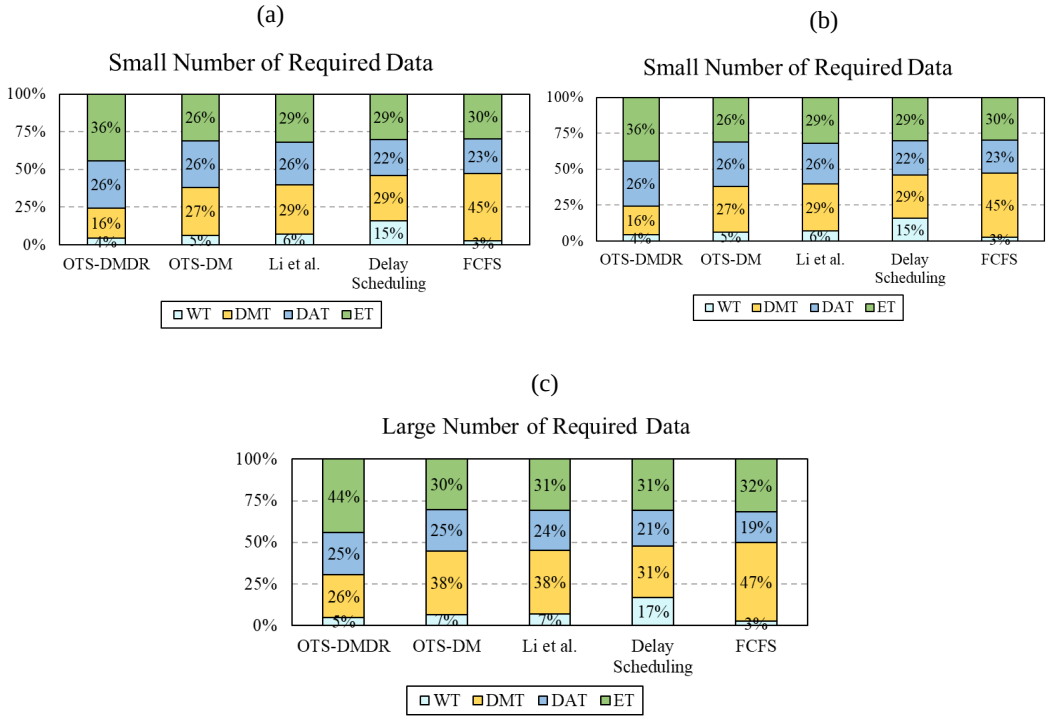

Figure 14. Percentage Response Time of Dataset Variation for the proposed methods *OTS-DM* and *OTS-DMDR*, compared to *Li et al.* [28], *Delay Scheduling* [63] and *FCFS* [52], where (**a**) is for small number of required data, (**b**) is for medium number of required data and (**c**) is for large number of required data.

Finally, we conclude that the obtained results by our proposed algorithm *OTS-DMDR* are more stable than those generated by the other scheduling algorithms. Furthermore, *OTS-DMDR* is very applicable to different scales of required datasets, which ensures the usefulness of the proposed task scheduling and validates the theoretical algorithm design developed in this paper.

5.3.4. Experiment 4: Tasks Arrive in 100 Time-Slot

For this scenario, we aimed to evaluate the throughput metric, which is the percentage of tasks executed for a specific time slot. Importantly, we analyze the load balancing of our system using the Degree of Imbalance (DI) metric. For this, we have 2000 tasks to execute in 100 machines. The tasks will arrive every 100 time slots. The throughput, the percentage of tasks that were completed for a given time slot, is indicated for each point in Figure 15.

As expected, *FCFS* did not perform well. Meanwhile, the *Delay Scheduling*, *Li et al.*, and *OTS-DM* methods were comparable and gave acceptable results. *OTS-DMDR* achieved

the best performance due to the efficient management of data replication throughout task scheduling.

Figure 16 shows the degree of imbalance for the *FCFS*, *Delay scheduling*, *Li et al.*, *OTS-DM*, and *OTS-DMDR* algorithms, where the lower value of DI indicated a higher load balancing performance.

According to the reported results, it can be concluded that the degree of imbalance of the proposed *OTS-DMDR* algorithm had the smallest value. Since the *OTS-DMDR* strategy considers the load of each machine when assigning tasks, as a result, it avoids imbalanced workload situations.

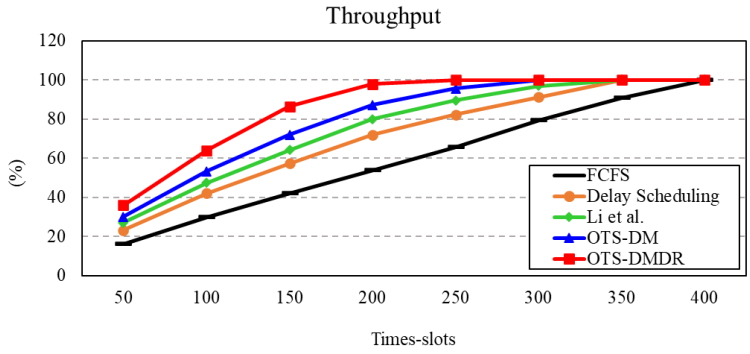

Figure 15. Throughput [28].

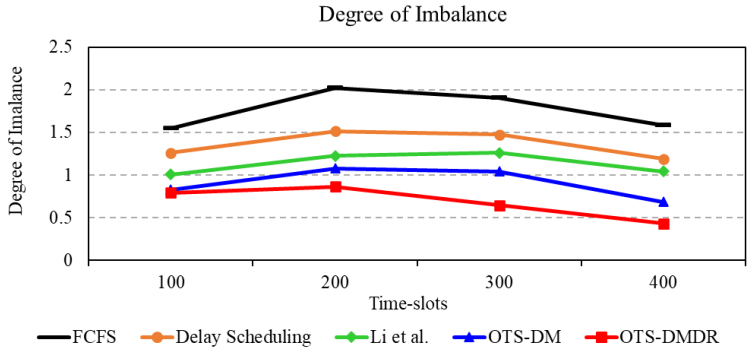

Figure 16. Degree of Imbalance for the proposed methods *OTS-DM* and *OTS-DMDR*, compared to *Li et al.* [28], *Delay Scheduling* [63] and *FCFS* [52].

However, a remarkable topic to be discussed is how to combine the proposed algorithm with other meta-heuristic algorithms to further enhance scheduling results by choosing optimal weight for parameters involved in the objective function, as discussed in [59,61].

6. Conclusions

Big data analytics tasks are now feasible due to advances in internet technology and the use of cloud data centers. However, managing these data-intensive tasks is challenging, especially in dynamic cloud environments. To address this challenge, it becomes highly demanding to consider data aspects when designing task scheduling algorithms. This paper introduces a new method named Online Task Scheduling based on Data Migration and Data Replication (*OTS-DMDR*). It considers various metrics to select the appropriate task for the appropriate machine, including, data access time, data migration time, tasks requirement, performance power, and load of the machines. By combining data migration and data replication features with delay scheduling, the *OTS-DMDR* method achieves better data locality, minimizes the response time, and improves the task throughput.

Accordingly, extensive simulations are carried out to demonstrate the validity of our proposed *OTS-DMDR* method. The results show that the proposed *OTS-DMDR* method outperforms existing scheduling techniques, reducing response time by 78% when

compared to the *First Come First Served (FCFS)* scheduler, by 58% compared to the *Delay Scheduling*, and by 46% compared to the technique of *Li et al*.—all of this while ensuring a balanced load over the machines. Consequently, this demonstrates the effectiveness and convenience of the proposed approach for the problem of online task scheduling.

The study on online task scheduling combined with data migration and replication in the cloud presents an important research implication for the development of efficient task scheduling algorithms for data-intensive applications. The study's findings indicate the importance of considering data locality in task scheduling, which can be further explored in future research. Furthermore, as future work, it will be important to investigate how to dynamically place the initial datasets and handle data replicas in order to enhance the performance of the system. In conclusion, it can be inferred that the performance of the proposed *OTS-DMDR* algorithm is adequate to be utilized as an alternative online task scheduling algorithm for big data systems.

Author Contributions: All authors contributed equally to this work. L.B., M.Z. and C.T. designed and performed the experiments and prepared the manuscript. L.B., M.Z. and C.T. supervised the work and contributed to the writing of the paper. All authors read and approved the final manuscript.

Funding: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Data Availability Statement: The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Acknowledgments: The authors thankfully acknowledge the Laboratory of the Smart Systems Laboratory (SSLab) ENSIAS for his support to achieve this work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Barika, M.; Garg, S.; Zomaya, A.Y.; Wang, L.; Moorsel, A.V.; Ranjan, R. Orchestrating Big Data Analysis Workflows in the Cloud: Research Challenges, Survey, and Future Directions. *ACM Comput. Surv.* **2019**, *52*, 1–41. [CrossRef]
2. Rjoub, G.; Bentahar, J.; Wahab, O.A. BigTrustScheduling: Trust-aware big data task scheduling approach in cloud computing environments. *Future Gener. Comput. Syst.* **2020**, *110*, 1079–1097. [CrossRef]
3. Cao, K.; Liu, Y.; Meng, G.; Sun, Q. An Overview on Edge Computing Research. *IEEE Access* **2020**, *8*, 85714–85728. [CrossRef]
4. Petrolo, R.; Loscrì, V.; Mitton, N. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Trans. Emerg. Telecommun. Technol.* **2017**, *28*, e2931. [CrossRef]
5. Fedushko, S.; Ustyianovych, T.; Syerov, Y.; Peracek, T. User-Engagement Score and SLIs/SLOs/SLAs Measurements Correlation of E-Business Projects Through Big Data Analysis. *Appl. Sci.* **2020**, *10*, 9112. [CrossRef]
6. Zhang, C.; Li, M.; Wu, D. Federated Multidomain Learning With Graph Ensemble Autoencoder GMM for Emotion Recognition. *IEEE Trans. Intell. Transp. Syst.* **2022**, 1–11. [CrossRef]
7. Luo, X.; Zhang, C.; Bai, L. A fixed clustering protocol based on random relay strategy for EHWSN. *Digit. Commun. Netw.* **2023**, *9*, 90–100. [CrossRef]
8. Chen, H.; Wen, J.; Pedrycz, W.; Wu, G. Big Data Processing Workflows Oriented Real-Time Scheduling Algorithm using Task-Duplication in Geo-Distributed Clouds. *IEEE Trans. Big Data* **2020**, *6*, 131–144. [CrossRef]
9. Arunarani, A.; Manjula, D.; Sugumaran, V. Task scheduling techniques in cloud computing: A literature survey. *Future Gener. Comput. Syst.* **2019**, *91*, 407–415. [CrossRef]
10. Amini Motlagh, A.; Movaghar, A.; Rahmani, A.M. Task scheduling mechanisms in cloud computing: A systematic review. *Int. J. Commun. Syst.* **2020**, *33*, e4302. [CrossRef]
11. Kumar, M.; Sharma, S.; Goel, A.; Singh, S. A comprehensive survey for scheduling techniques in cloud computing. *J. Netw. Comput. Appl.* **2019**, *143*, 1–33. [CrossRef]
12. Liu, J.; Pacitti, E.; Valduriez, P. A Survey of Scheduling Frameworks in Big Data Systems. *Int. J. Cloud Comput.* **2018**, *7*, 103–128. [CrossRef]
13. Gautam, J.V.; Prajapati, H.B.; Dabhi, V.K.; Chaudhary, S. A survey on job scheduling algorithms in Big data processing. In Proceedings of the 2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India, 5–7 March 2015; pp. 1–11. [CrossRef]
14. Mishra, S.K.; Puthal, D.; Sahoo, B.; Jena, S.K.; Obaidat, M.S. An adaptive task allocation technique for green cloud computing. *J. Supercomput.* **2017**, *74*, 370–385. [CrossRef]

15. Stavrinides, G.L.; Karatza, H.D. Scheduling Data-Intensive Workloads in Large-Scale Distributed Systems: Trends and Challenges. In *Modeling and Simulation in HPC and Cloud Systems*; Kołodziej, J., Pop, F., Dobre, C., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 19–43. [CrossRef]
16. Yang, C.; Huang, Q.; Li, Z.; Liu, K.; Hu, F. Big Data and cloud computing: Innovation opportunities and challenges. *Int. J. Digit. Earth* **2017**, *10*, 13–53. [CrossRef]
17. Hashem, I.A.T.; Yaqoob, I.; Anuar, N.B.; Mokhtar, S.; Gani, A.; Ullah Khan, S. The rise of "big data" on cloud computing: Review and open research issues. *Inf. Syst.* **2015**, *47*, 98–115. [CrossRef]
18. Mazumdar, S.; Seybold, D.; Kritikos, K.; Verginadis, Y. A survey on data storage and placement methodologies for Cloud-Big Data ecosystem. *J. Big Data* **2019**, *6*, 1–37. [CrossRef]
19. Natesan, G.; Chokkalingam, A. Task scheduling in heterogeneous cloud environment using mean grey wolf optimization algorithm. *ICT Express* **2019**, *5*, 110–114. [CrossRef]
20. Jafarnejad Ghomi, E.; Masoud Rahmani, A.; Nasih Qader, N. Load-balancing algorithms in cloud computing: A survey. *J. Netw. Comput. Appl.* **2017**, *88*, 50–71. [CrossRef]
21. Alami Milani, B.; Jafari Navimipour, N. A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions. *J. Netw. Comput. Appl.* **2016**, *64*, 229–238. [CrossRef]
22. Ahmad, N.; Che Fauzi, A.A.; Sidek, R.; Zin, N.; Beg, A. Lowest Data Replication Storage of Binary Vote Assignment Data Grid. *Commun. Comput. Inf. Sci.* **2010**, *88*, 466–473. [CrossRef]
23. Mohammadi, B.; Navimipour, N.J. Data replication mechanisms in the peer-to-peer networks. *Int. J. Commun. Syst.* **2019**, *32*, e3996. [CrossRef]
24. Campêlo, R.A.; Casanova, M.A.; Guedes, D.O.; Laender, A.H.F. A Brief Survey on Replica Consistency in Cloud Environments. *J. Internet Serv. Appl.* **2020**, *11*, 1. [CrossRef]
25. Long, S.Q.; Zhao, Y.L.; Chen, W. MORM: A Multi-objective Optimized Replication Management strategy for cloud storage cluster. *J. Syst. Archit.* **2014**, *60*, 234–244. [CrossRef]
26. Mokadem, R.; Hameurlain, A. A data replication strategy with tenant performance and provider economic profit guarantees in Cloud data centers. *J. Syst. Softw.* **2020**, *159*, 110447. [CrossRef]
27. Wang, D.; Chen, J.; Zhao, W. A Task Scheduling Algorithm for Hadoop Platform. *J. Comput.* **2013**, *8*, 929–936. [CrossRef]
28. Li, X.; Wang, L.; Lian, Z.; Qin, X. Migration-based Online CPSCN Big Data Analysis in Data Centers. *IEEE Access* **2018**, *6*, 19270–19277. [CrossRef]
29. Dubey, K.; Kumar, M.; Sharma, S. Modified HEFT Algorithm for Task Scheduling in Cloud Environment. *Procedia Comput. Sci.* **2018**, *125*, 725–732. [CrossRef]
30. Mondal, R.; Nandi, E.; Sarddar, D. Load Balancing Scheduling with Shortest Load First. *Int. J. Grid Distrib. Comput.* **2015**, *8*, 171–178. [CrossRef]
31. Lakra, A.V.; Yadav, D.K. Multi-Objective Tasks Scheduling Algorithm for Cloud Computing Throughput Optimization. *Procedia Comput. Sci.* **2015**, *48*, 107–113. [CrossRef]
32. Wang, H.; Wang, F.; Liu, J.; Wang, D.; Groen, J. Enabling customer-provided resources for cloud computing: Potentials, challenges, and implementation. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 1874–1886. [CrossRef]
33. Gill, S.S.; Chana, I.; Singh, M.; Buyya, R. CHOPPER: An intelligent QoS-aware autonomic resource management approach for cloud computing. *Clust. Comput.* **2018**, *21*, 1203–1241. [CrossRef]
34. Thomas, A.; Krishnalal, G.; Raj, P.V. Credit Based Scheduling Algorithm in Cloud Computing Environment. *Procedia Comput. Sci.* **2015**, *46*, 913–920. [CrossRef]
35. Sajid, M.; Raza, Z. Turnaround Time Minimization-Based Static Scheduling Model Using Task Duplication for Fine-Grained Parallel Applications onto Hybrid Cloud Environment. *IETE J. Res.* **2015**, *62*, 402–414. [CrossRef]
36. Hadji, M.; Zeghlache, D. Minimum Cost Maximum Flow Algorithm for Dynamic Resource Allocation in Clouds. In Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 876–882. [CrossRef]
37. Elzeki, O.; Reshad, M.; Abu Elsoud, M. Improved Max-Min Algorithm in Cloud Computing. *Int. J. Comput. Appl.* **2012**, *50*, 22–27. [CrossRef]
38. Fernández Cerero, D.; Fernández-Montes, A.; Jakóbik, A.; Kołodziej, J.; Toro, M. SCORE: Simulator for cloud optimization of resources and energy consumption. *Simul. Model. Pract. Theory* **2018**, *82*, 160–173. [CrossRef]
39. Ma, T.; Chu, Y.; Zhao, L.; Otgonbayar, A. Resource Allocation and Scheduling in Cloud Computing: Policy and Algorithm. *IETE Tech. Rev.* **2014**, *31*, 4–16. [CrossRef]
40. Carrasco, R.; Iyengar, G.; Stein, C. Resource Cost Aware Scheduling. *Eur. J. Oper. Res.* **2018**, *269*, 621–632. [CrossRef]
41. Coninck, E.; Verbelen, T.; Vankeirsbilck, B.; Bohez, S.; Simoens, P.; Dhoedt, B. Dynamic Auto-scaling and Scheduling of Deadline Constrained Service Workloads on IaaS Clouds. *J. Syst. Softw.* **2016**, *118*, 101–114. [CrossRef]
42. Yi, P.; Ding, H.; Ramamurthy, B. Budget-Minimized Resource Allocation and Task Scheduling in Distributed Grid/Clouds. In Proceedings of the 2013 22nd International Conference on Computer Communication and Networks (ICCCN), Nassau, Bahamas, 30 July–2 August 2013; pp. 1–8. [CrossRef]
43. Reddy, G. A Deadline and Budget Constrained Cost and Time Optimization Algorithm for Cloud Computing. *Commun. Comput. Inf. Sci.* **2011**, *193*, 455–462. [CrossRef]

44. Xin, Y.; Xie, Z.Q.; Yang, J. A load balance oriented cost efficient scheduling method for parallel tasks. *J. Netw. Comput. Appl.* **2017**, *81*, 37–46. [CrossRef]
45. Yang, S.J.; Chen, Y.R. Design adaptive task allocation scheduler to improve MapReduce performance in heterogeneous Clouds. *J. Netw. Comput. Appl.* **2015**, *57*, 61–70. [CrossRef]
46. Smara, M.; Aliouat, M.; Pathan, A.S.; Aliouat, Z. Acceptance Test for Fault Detection in Component-based Cloud Computing and Systems. *Future Gener. Comput. Syst.* **2016**, *70*, 74–93. [CrossRef]
47. Fan, G.; Chen, L.; Yu, H.; Liu, D. Modeling and Analyzing Dynamic Fault-Tolerant Strategy for Deadline Constrained Task Scheduling in Cloud Computing. *IEEE Trans. Syst. Man Cybern. Syst.* **2017**, *50*, 1260–1274. [CrossRef]
48. Zhou, Z.; Abawajy, J.; Chowdhury, M.; Hu, Z.; Li, K.; Cheng, H.; Alelaiwi, A.; Li, F. Minimizing SLA violation and power consumption in Cloud data centers using adaptive energy-aware algorithms. *Future Gener. Comput. Syst.* **2017**, *86*, 836–850. [CrossRef]
49. Pradhan, R.; Satapathy, S. Energy-Aware Cloud Task Scheduling algorithm in heterogeneous multi-cloud environment. *Intell. Decis. Technol.* **2022**, *16*, 279–284. [CrossRef]
50. Chen, H.; Liu, G.; Yin, S.; Liu, X.; Qiu, D. ERECT: Energy-Efficient Reactive Scheduling for Real-Time Tasks in Heterogeneous Virtualized Clouds. *J. Comput. Sci.* **2017**, *28*, 416–425. [CrossRef]
51. Duan, H.; Chen, C.; Min, G.; Wu, Y. Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems. *Future Gener. Comput. Syst.* **2017**, *74*, 142–150. [CrossRef]
52. Shaikh, M.B.; Waghmare Shinde, K.; Borde, S. Challenges of Big Data Processing and Scheduling of Processes Using Various Hadoop Schedulers: A Survey. *Int. J. Multifaceted Multiling. Stud.* **2019**, *III*, 1–6.
53. Mohapatra, S.; Mohanty, S.; Rekha, K. Analysis of Different Variants in Round Robin Algorithms for Load Balancing in Cloud Computing. *Int. J. Comput. Appl.* **2013**, *69*, 17–21. [CrossRef]
54. Li, R.; Hu, H.; Li, H.; Wu, Y.; Yang, J. MapReduce Parallel Programming Model: A State-of-the-Art Survey. *Int. J. Parallel Program.* **2016**, *44*, 832–866. [CrossRef]
55. Shyam, G.K.; Manvi, S.S. Resource allocation in cloud computing using agents. In Proceedings of the 2015 IEEE International Advance Computing Conference (IACC), Banglore, India, 12–13 June 2015; pp. 458–463. [CrossRef]
56. Zhao, Q.; Xiong, C.; Yu, C.; Zhang, C.; Zhao, X. A new energy-aware task scheduling method for data-intensive applications in the cloud. *J. Netw. Comput. Appl.* **2016**, *59*, 14–27. [CrossRef]
57. Dubey, K.; Kumar, M.; Chandra, M.A. A priority based job scheduling algorithm using IBA and EASY algorithm for cloud metaschedular. In Proceedings of the 2015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, India, 19–20 March 2015; pp. 66–70. [CrossRef]
58. Nasr, A.A.; El-Bahnasawy, N.A.; Attiya, G.; El-Sayed, A. A new online scheduling approach for enhancing QOS in cloud. *Future Comput. Inform. J.* **2018**, *3*, 424–435. [CrossRef]
59. Reddy, G.; Kumar, S. MACO-MOTS: Modified Ant Colony Optimization for Multi Objective Task Scheduling in Cloud Environment. *Int. J. Intell. Syst. Appl.* **2019**, *11*, 73–79. [CrossRef]
60. Biswas, D.; Samsuddoha, M.; Asif, M.R.A.; Ahmed, M.M. Optimized Round Robin Scheduling Algorithm Using Dynamic Time Quantum Approach in Cloud Computing Environment. *Int. J. Intell. Syst. Appl.* **2023**, *15*, 22–34. [CrossRef]
61. Soltani, N.; Barekatain, B.; Soleimani Neysiani, B. MTC: Minimizing Time and Cost of Cloud Task Scheduling based on Customers and Providers Needs using Genetic Algorithm. *Int. J. Intell. Syst. Appl.* **2021**, *13*, 38–51. [CrossRef]
62. Mohseni, Z.; Kiani, V.; Rahmani, A. A Task Scheduling Model for Multi-CPU and Multi-Hard Disk Drive in Soft Real-time Systems. *Int. J. Inf. Technol. Comput. Sci.* **2019**, *11*, 1–13. [CrossRef]
63. Zaharia, M.; Borthakur, D.; Sen Sarma, J.; Elmeleegy, K.; Shenker, S.; Stoica, I. *Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling*; EuroSys'10; Association for Computing Machinery: New York, NY, USA, 2010; pp. 265–278. [CrossRef]
64. He, C.; Lu, Y.; Swanson, D. Matchmaking: A New MapReduce Scheduling Technique. In Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, Athens, Greece, 29 November–1 December 2011; pp. 40–47. [CrossRef]
65. Kosar, T.; Balman, M. A new paradigm: Data-aware scheduling in grid computing. *Future Gener. Comput. Syst.* **2009**, *25*, 406–413. [CrossRef]
66. Vobugari, S.; Somayajulu, D.V.L.N.; Subaraya, B.M. Dynamic Replication Algorithm for Data Replication to Improve System Availability: A Performance Engineering Approach. *IETE J. Res.* **2015**, *61*, 132–141. [CrossRef]
67. Bouhouch, L.; Zbakh, M.; Tadonki, C. A Big Data Placement Strategy in Geographically Distributed Datacenters. In Proceedings of the 2020 5th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech), Marrakesh, Morocco, 24–26 November 2020; pp. 1–9. [CrossRef]
68. Bouhouch, L.; Zbakh, M.; Tadonki, C. Dynamic data replication and placement strategy in geographically distributed data centers. *Concurr. Comput. Pract. Exp.* 2022, *early view*. [CrossRef]
69. Mohamed, A.; Najafabadi, M.K.; Wah, Y.B.; Zaman, E.A.K.; Maskat, R. The state of the art and taxonomy of big data analytics: view from new big data framework. *Artif. Intell. Rev.* **2020**, *53*, 989–1037. [CrossRef]
70. Samadi, Y.; Zbakh, M.; Tadonki, C. DT-MG: Many-to-one matching game for tasks scheduling towards resources optimization in cloud computing. *Int. J. Comput. Appl.* **2021**, *43*, 233–245. [CrossRef]

71. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.F.; Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **2011**, *41*, 23–50. [CrossRef]
72. Calheiros, R.; Ranjan, R.; De Rose, C.; Buyya, R. CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. *arXiv* **2009**, arXiv:0903.2525.
73. Bouhouch, L.; Zbakh, M.; Tadonki, C. Data Migration: Cloudsim Extension. In Proceedings of the ICBDR 2019: 2019 the 3rd International Conference on Big Data Research, Cergy-Pontoise, France, 20–22 November 2019; pp. 177–181. [CrossRef]
74. Niznik, C.A. Min-max vs. max-min flow control algorithms for optimal computer network capacity assignment. *J. Comput. Appl. Math.* **1984**, *11*, 209–224. [CrossRef]