symmetry

Article

ROA-CONS: Raccoon Optimization for Job Scheduling

Sina Zangbari Koohi [1,*], Nor Asilah Wati Abdul Hamid [1,2], Mohamed Othman [1,2] and Gafurjan Ibragimov [3]

[1] Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang 43400, Selangor, Malaysia; asila@upm.edu.my (N.A.W.A.H.); mothman@upm.edu.my (M.O.)
[2] Laboratory of Computational Sciences and Mathematical Physics, Institute for Mathematical Research, Universiti Putra Malaysia, Serdang 43400, Selangor, Malaysia
[3] Department of Mathematics, Faculty of Science, Universiti Putra Malaysia, Serdang 43400, Selangor, Malaysia; ibragimov@upm.edu.my
* Correspondence: zangbari@gmail.com; Tel.: +60-113-736-3296

Abstract: High-performance computing comprises thousands of processing powers in order to deliver higher performance computation than a typical desktop computer or workstation in order to solve large problems in science, engineering, or business. The scheduling of these machines has an important impact on their performance. HPC's job scheduling is intended to develop an operational strategy which utilises resources efficiently and avoids delays. An optimised schedule results in greater efficiency of the parallel machine. In addition, processes and network heterogeneity is another difficulty for the scheduling algorithm. Another problem for parallel job scheduling is user fairness. One of the issues in this field of study is providing a balanced schedule that enhances efficiency and user fairness. ROA-CONS is a new job scheduling method proposed in this paper. It describes a new scheduling approach, which is a combination of an updated conservative backfilling approach further optimised by the raccoon optimisation algorithm. This algorithm also proposes a technique of selection that combines job waiting and response time optimisation with user fairness. It contributes to the development of a symmetrical schedule that increases user satisfaction and performance. In comparison with other well-known job scheduling algorithms, the simulation assesses the effectiveness of the proposed method. The results demonstrate that the proposed strategy offers improved schedules that reduce the overall system's job waiting and response times.

Keywords: conservative backfilling; job scheduling; optimization; parallel computer; raccoon optimization algorithm

check for updates

Citation: Koohi, S.Z.; Abdul Hamid, N.A.W.; Othman, M.; Ibragimov, G. ROA-CONS: Raccoon Optimization for Job Scheduling. *Symmetry* **2021**, *13*, 2270. https://doi.org/10.3390/sym13122270

Academic Editor: László T. Kóczy

Received: 1 November 2021
Accepted: 16 November 2021
Published: 29 November 2021

1. Introduction

The high-performance computers (HPCs) commonly comprise hundreds or thousands of computer servers that are in a network together and each server is called a node. These nodes will be access by the user with the assistance of middleware commonly referred to as a scheduler. The middleware is responsible for allocating resources to users, providing a framework for starting, executing, and monitoring work on allocated resources and scheduling work for future execution. Job scheduling on these machines has a major impact on the machines' overall performance. In general, these systems use many resources and a great deal of energy. Optimized job schedules on HPCs reduce the consumption of resources and energy.

The purpose of HPC job planning is to design an operational strategy that makes effective use of resources and eliminates idle time. The machine can therefore complete the necessary procedures and produce the result in a quicker time. In addition, user fairness is an important issue to consider while developing a plan. Jobs are allocated to HPCs by a variety of users, and ensuring fairness among these users is a critical part of job scheduling. A balanced or symmetrical plan strikes a balance between resource usage effectiveness and user satisfaction.

An HPC is made up of a number of discrete servers (computers), known as nodes, that are linked together by a fast interconnect or network. These machines typically use different types of network media for distinct communication features. Furthermore, these supercomputers use numerous processors that have various processing capacities and make the job scheduling more challenging. Generally speaking, a large network, various processors, and non-uniform submitted jobs are the key challenges in dealing with job schedulers in parallel computers.

Many researchers have investigated the impact of job schedules on parallel computer performance [1–4]. These experiments have demonstrated that the superior job planning algorithm improves efficiency and performance.

Two types of job scheduling algorithms are generally available, fixed, and flexible. Fixed methods seek the optimal spot in the existing schedule for the arrived job. These methods do not modify existing schedules and just obtain the appropriate position for the job. Instead, when the new jobs are assigned a spot, the flexible techniques aim to adjust the existing schedule.

Any scheduling algorithm's principal objective is to gain a high-quality symmetrical parallel machine schedule. The planning algorithm should be quick, accurate, and fair, however. A sluggish programming algorithm can waste system resources and add additional overhead. These are our key reasons for proposing a new algorithm for scheduling jobs. The proposed method is a high-level hybridization of fixed and flexible methods. It combines the advantages of each method while overcoming their flaws. The suggested method's quality and performance are compared to three well-known task scheduling algorithms: CONS, BF-EASY, and CONS-Tabu. The findings show that the suggested scheduling method produces optimal execution plans, which improves overall system performance and user satisfaction, symmetrically.

This paper is arranged as follows. Section 2 provides a background for job programming. Section 3 goes through the proposed scheduling algorithm in detail. Section 4 displays the algorithm's simulation pattern as well as the simulation results. The outcome is also discussed in this section. Section 5 brings the study to a conclusion.

2. Related Work

The job scheduling method has been found to have a strong impact on the overall performance of high-performance parallel computers [5,6]. An optimised algorithm contributes to increasing the performance that reduces user waiting time and saves resources and electricity usage.

The efficiency of the scheduling methods is increased by an optimum meta-heuristic algorithm together with an effective hypergraph partitioning process. First Come First Serve (FCFS) [7] is one of the classic methods for parallel job scheduling. This approach delivers sufficient solutions for situations of small-scale systems but is not good at medium and big scale. To cope with this issue, two basic approaches, the queue based and schedule based methods, have been presented.

Queue methods (often referred to as priority algorithms) [8,9] feature several approaches such as Shortest Job First (SJF) [10,11], Minimum Due Date Time (MTTD) [12], and Extensible Argonne Scheduling System (EASY) [13]. These types of planners are good for modest high performance architectures (HPAs). However, they exhibit their bottlenecks in the large-scale machines. The long waiting times for a large number of submitted jobs are among the major problems with these schedulers [14,15].

The schedule-based algorithms are shown to be more flexible and designed to work more efficiently [9,16,17]. Conservative backfilling algorithms (CONS) [18], a sort of backfilling schedule, are one of the most famous schedule-based algorithms. Various studies have shown their efficiency in static job scheduling for HPAs [19–21].

The backfilling uses a fixed scheduling order. Every submitted job is given room in this way. Thus, the order of executions can be predicted. When the system receives a new job, the algorithm creates a new place for it in an ad hoc manner. However, CONS avoids backfilling short jobs if it means slowing down the entire process, which has limited the CONS. The findings in the prior literature indicate that the conservation algorithm for backfilling is not adequately performed with regard to waiting time [18,22–24]. The key cause for this low efficiency is the use of a fixed procedure in the schedule that does not take into account other jobs in securing a gap [25].

In most cases, BF-EASY [26] and CONS-Tabu [27] outperformed other algorithms such as the CONS algorithm. This result is almost trivial, as most of the other experiments reported [20,28,29]. CONS's reservation approach is less efficient than the aggressive and meta-heuristic methods employed in the BF-EASY and CONS-Tabu methods. BF-EASY employs a locking system, which has a significant impact on reserving jobs and reduces the opportunities for backfilling [19,20]. Other studies attempt to improve on this strategy by employing meta-heuristics. In general, the scheduling in these algorithms is a fixed technique, which means that this reservation has no effect on the prior schedule. However, in terms of job waiting and response time, the fixed technique cannot ensure the best performance of the system [27].

The flexible methods attempt to solve problems with fixed approaches. These solutions employ a heuristic algorithm to reorganise the plan every time a new job enters the system. These strategies enhance scheduling, but they are insufficient for real problems that will be running for a long time [27,30].

This paper provides a hybrid approach for using the benefits of both approaches. It consists of an improved conservative backfilling method (fixed method) as well as a flexible method that employs a meta-heuristic algorithm to create an execution plan that reduces job waiting and response time.

3. ROA-CONS Scheduling Algorithm

The ROA-CONS approach is proposed in this paper for scheduling jobs in heterogeneous high-performance parallel computers. This approach is an extension of the conservative backfilling method, in which an execution plan is devised using both fixed and flexible approaches. The Raccoon Optimization Algorithm (ROA) is combined with conservative backfilling fixed ordering to create ROA-CONS.

Algorithm hybridization is divided into two categories: high-level and low-level. Low-level hybridization is a closely coupled set of algorithms that are interdependent. High-level hybridization, on the other hand, is a loose coupling of algorithms that run independently and the results are used after the algorithms are completed [31]. ROA-CONS is a hybridization of the ROA and CONS at a high stage.

When a job arrives in the system, ROA-CONS uses a fixed method to reserve room for it and produces a fixed schedule. After that, the ROA receives the schedule and tries to improve it. To overcome the overhead bottleneck, this heuristic algorithm has two stop conditions: the number of iterations and the time limit. The ROA creates a plan that is flexible. Finally, a selection algorithm is used to choose the best schedule from the fixed and flexible schedule options. A flowchart of this method is shown in Figure 1.

In order to gauge jobs, the specifications and details of these jobs should be mathematically summarised. Modelling is the way in which the properties of applications are collected. It refers to a mathematical data structure that can be used to describe processes and jobs. The models also contain measures or metrics to evaluate the quality of the schedule. These metrics contribute to the maximisation of the results of scheduling.

To account for the heterogeneity concept, the jobs must be modelled with the topology of the super computer that will do these jobs. In current super computers, different sorts of heterogeneity arise: heterogeneity of the processor and heterogeneity of the network. In order to boost the schedule, both forms of heterogeneity must be considered.

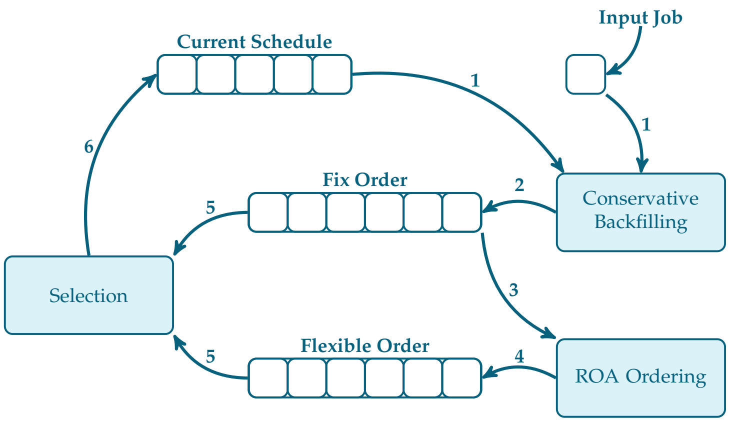

Figure 1. A general flowchart of ROA-CONS (the numbers above each arrow indicate its time sequence).

This article employs the MEMPHA modelling approach. This approach of modelling provides a graph of the topology model $(\mathcal{G} = (\mathcal{V}, \mathcal{E}, w_{\mathcal{G}_c}, w_{\mathcal{G}_s}, c_{\mathcal{G}_d}, c_{\mathcal{G}_b}))$ and a hyper-graphic job model $(\mathcal{H} = (\mathcal{T}, \mathcal{D}, w_{\mathcal{H}}, c_{\mathcal{H}}))$.

MEMPHA's topology model abstracts the parallel machine characters into a graph. In a mathematical graph, it summarises the machine's details, such as the processors, their execution capacity, and the network media that connect processors. MEMPHA's topology model is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w_{\mathcal{G}_c}, w_{\mathcal{G}_s}, c_{\mathcal{G}_d}, c_{\mathcal{G}_b})$ in which:

- The graph's nodes are represented by $\mathcal{V}$. Physical nodes in the target distributed computer are represented by nodes in this model. The physical nodes in this graph are processors and network switches;
- $\mathcal{E}$ is the graph's set of edges. The connections between processors and switches (physical nodes) via network media are represented as graph edges;
- Each node in this graph has a positive weight, $w_{\mathcal{G}_c}$. It denotes the number of processor cores represented by each node. This weight is 0 if the node represents a network switch;
- $w_{\mathcal{G}_s}$ is another weight assigned to each node that reflects the processor's base clock speed in gigahertz (GHz) if the node represents a processor, unless it is zero;
- $c_{\mathcal{G}_d}$ is the cost assigned to each edge, and it indicates the connection delay for each network media when transferring one kilobyte of data;
- $c_{\mathcal{G}_b}$ is the additional cost associated with each edge, representing the bandwidth of the respective network media.

The job model in MEMPHA records the features of the parallel application. In MEMPHA, a job model is a hyper-graph $\mathcal{H} = (\mathcal{T}, \mathcal{D}, w_{\mathcal{H}}, c_{\mathcal{H}})$ in which:

- $\mathcal{T}$ denotes the hyper-graph's set of nodes. Each task in the parallel application is translated into a hyper-graph node;
- $\mathcal{D}$ denotes the hyper-graph's set of hyper-arcs. Each data exchange between tasks in the parallel application is translated into a hyper-arc;
- $w_{\mathcal{H}}$ is a positive weight applied to each node in the hyper-graph that represents the task's execution time on a single-core processor running at 1GHz;
- The volume of the exchanged data element in kilobytes is denoted by $c_{\mathcal{H}}$, a positive cost assigned to each hyper-arc.

MEMPHA also gives a set of metrics to help with scheduling. MEMPHA's metrics are listed in Table 1.

MEMPHA's modelling technique covers heterogeneous architectures and collects details on the heterogeneity of the networks and processors. For more specifics on the MEMPHA modelling approach, please refer to [32].

Table 1. Metrics provided by MEMPHA.

#	Metric	Description
1	$Dilation(t, q)$	Delay of data transmission between two jobs t and q
2	$Traffic(u, v)$	The amount of data that transfers between two processors u and v.
3	$Congestion(u, v)$	The ratio of traffic travelling across a link (between processors u and v) to its capacity.
4	$Congestion(\Gamma)$	A lower bound on the total amount of time required for all communications
5	$DVS(v)$	The total amount of data sent by a processor (v).
6	$DVR(v)$	The total amount of data received by a processor (v).
7	$TCV(\Gamma)$	Total communication volume.
8	$DVSR(v)$	Data volume sent or received by a processor
9	$MVSR(\Gamma)$	Maximum volume of data exchange in application
10	$NM(v)$	Number of messages sent by a processor (v)
11	$TM(\Gamma)$	Total number of messages exchanged in the application

As previously stated, a parallel machine receives a large number of jobs. Any of these jobs are parallel applications that are made up of many tasks that will be run on the parallel machine. These tasks, unlike jobs given to the machine, are interdependent. In addition to scheduling jobs, tasks inside each job should be scheduled based on the resources that have been allocated. This study employs the HATS task scheduling method. This strategy has been found to be more efficient and accurate than competitors, resulting in schedules with superior performance [33]. For further detail on the HATS scheduling algorithm, please see [33].

Conservative Backfilling, ROA Ordering, and Selection are the three phases of the scheduling process, as shown in Figure 1. These steps are described in detail in the following sections.

3.1. Conservative Backfilling

The first phase in scheduling is the backfilling. It assigns a position to each job that is submitted and predicts the order in which they will be executed. When a new job is added to the system, the backfilling method introduces a new place for it. However, since the proposed scheduling approach employs topology-aware multilevel hypergraph partitioning, the backfilling process's parameters and metrics must be redefined to meet the topology-awareness objectives. When a new job is received by the scheduler, the backfilling process places it in the fixed order.

The schedule is empty at the start of the scheduling process. The empty schedule has been regarded as an endless gap. As a result, every new assignment would be allocated promptly. It has also taken into account the fact that there is always an infinite gap at the end of the schedule. As a result, if the backfilling cannot locate an appropriate place for the job, it will be added to the end of the order.

There are various arrival times for the jobs submitted to the parallel machine. This creates some gaps between jobs where the parallel machine is idle. The primary goal of scheduling is to optimise job execution and resource management. In order to achieve this objective, a conservative backfilling approach applies when a new job requires a gap in order. Consider a job with several suitable gaps in order. Backfilling should choose which gap to fill. This choice affects the system's efficiency. Effective decision making results in an optimised order.

To optimise, any optimisation technique requires one or more parameters. In the event of a job scheduling challenge, the essential factors to optimise are job waiting time and response time. The backfilling approach in this study takes these two primary criteria into account while identifying the optimal gap for the incoming job. These items are defined in the definitions that follow.

Definition 1 (**Job Waiting Time** (***Jwt***))**.** *The waiting time of a job, denoted as Jwt_{J_i}, is the amount of time that the job must wait until the requested resources are assigned. The waiting time for a job is determined by its location in the order.*

Any job is made up of various tasks, which are modelled in hypergraph $\mathcal{H}$. This hypergraph is partitioned by the scheduling algorithm to formulate an execution schedule. Multiple partitions make up an execution schedule. Π refers to the collection of all partitions defined in the schedule. Each element in Π (Π_i) is executed by a single HPC processor. As a result, the cumulative execution time of the tasks in partition ($\sum w_{\mathcal{H}}$) divided by the power of a processor ($w_{\mathcal{G}_s(i)}$) reflects the partition's execution time on that processor. The job execution time is formally defined in Definition 2.

Definition 2 (**Job Execution Time**)**.** *Consider a parallel application submitted to the scheduler J_i, its application model hypergraph $\mathcal{H}_{J_i} = (\mathcal{T}, \mathcal{D}, w_{\mathcal{H}}, c_{\mathcal{H}})$, the partitioning of this hypergraph $\Pi_{\mathcal{H}_{J_i}}$, and the topology model of the parallel machine $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w_{\mathcal{G}_c}, w_{\mathcal{G}_s}, c_{\mathcal{G}_d}, c_{\mathcal{G}_b})$. The execution time of this job Jet is as follows:*

$$Jet_{J_i} = \sum_{i=1}^{|\Pi|} \frac{\sum_{\forall t_q \in \Pi_i} w_{\mathcal{H}}(t_q)}{w_{\mathcal{G}_s}(i)} \tag{1}$$

Any task's response time is the time it takes between the job submission and the execution ends. The job will wait after submission until access to resources is obtained (Job Waiting Time, as defines in Definition 1). Equation (1) shows the time required for any job to be executed after gaining access to resources. The response time for any job is therefore equal to its waiting time plus execution time. The formal response time is described in Definition 3.

Definition 3 (**Job Response Time**)**.** *A job's response time is the time between when the job is submitted and when it is completed, i.e.,:*

$$Jrt_{J_i} = Jwt_{J_i} + Jet_{J_i} \tag{2}$$

When the backfilling receives a job, it creates a list of available gaps that could be used to host the job. Afterwards, for each gap, the response time of the received job is determined. The job is assigned to the gap with the shortest response time, and the order is updated. The backfilling procedure is depicted in Pseudocode 1.

Pseudocode 1 Conservative Backfilling

```
Input: Current Schedule $S$, Incoming Job $J$, Application Model Hypergraph $\mathcal{H} = (\mathcal{T}, \mathcal{D}, w_{\mathcal{H}}, c_{\mathcal{H}})$, Topology Model Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w_{\mathcal{G}}, c_{d_{(i,j)}}, c_{b_{(i,j)}})$, the partitioning of this hypergraph $\Pi_{\mathcal{H}_{J_i}}$,
Output: Fixed Order $order_{fixed}$
1: procedure ConservativeBackfilling($S$, $J$, $\mathcal{H}$, $\mathcal{G}$, $\Pi$)
2:     $availableGaps \leftarrow findGaps(S, J)$;          # Find all gaps which can host the job
3:     for all $ag \in availableGaps$ do
4:         $Jet_J \leftarrow \sum_{i=1}^{|\Pi|} \frac{\sum_{\forall t_q \in \Pi_i} w_{\mathcal{H}}(t_q)}{w_{\mathcal{G}_i}}$;
5:         $Jrt_J[ag] \leftarrow Jwt_{J_{ag}} + Jet_J$;
6:     end for
7:     $selectedGap \leftarrow MIN(Jrt_J)$;
8:     $order_{fixed} \leftarrow placeJob(S, J, selectedGap)$;          # Place $J$ in $selectedGap$ inside $S$
9: end procedure
```

3.2. ROA Ordering

The conservative backfilling method places the new job in the appropriate open slot. It tries to discover the ideal opening for it, but it is unconcerned about the other jobs in the order. However, if the scheduler re-sorts the position of the jobs in the order after adding a job in a gap, it is feasible to achieve higher performance. By re-sorting the jobs, the ROA ordering attempts to produce a superior order.

The Raccoon Optimization Algorithm is being used to optimise the fixed order generated by Conservative Backfilling. The Raccoon Optimization Technique (ROA) is a multi-population optimization algorithm proposed in [34]. This algorithm has been proved to be accurate and fast [35–37].

The ROA study demonstrated that this algorithm is both faster and more accurate than its competitors. However, two stop criteria have been set for this optimization technique to minimise any additional burden on the scheduler. The number of iterations (itr) is the initial stop condition. This is a user-defined parameter that can be changed during runtime. The time limit (reoffered as $limit$) is the second stop criterion. When the ROA optimization iterates for (itr) times or when the running duration surpasses the ($limit$), it stops.

A fitness function is required for ROA, as it is for any other optimization process. According to its fitness function, this algorithm optimises the order. The average response time of the order is the fitness function for ROA to optimise the fixed order.

ROA ordering attempts to optimise an order (S) generated by the conservative backfilling approach. Order S is a collection of jobs. The response time of any job in S was determined using Equation (2). As a result, the average order response time (ART) is the sum of the job response times divided by the number of jobs in S. Definition 4 provides a formal description of the ART.

Definition 4 (***Order Average Response Time***)**.** *An order's average response time is the average of all tasks in that order's response time. Consider the order S, wherein average response time is determined as:*

$$ART_S = \frac{1}{|S|} \sum_{\forall j \in S} Jrt_j \tag{3}$$

The ROA algorithm is a meta-population optimization method. By re-ordering the $order_{fixed}$, it is used to optimise (minimise) the average response time. ART_S, as defined above, is the fitness function used in this optimization. ROA calculates the ART_S value for various combinations of orders in $order_{fixed}$ and chooses the order with the shortest average response time. As a result, a new order called $order_{flexible}$ has been formed.

3.3. Selection

All of the criteria utilised in the preceding phases were based on the system's or jobs' performance. However, a scheduler's primary goal is not only to perform well from the system's point of view. A competent scheduler should be impartial. A scheduler's fairness refers to how it divides the parallel machine's resources among the users.

There were two types of orders in the previous phases: fixed and flexible. One of these orders is chosen as the system's principal schedule during the selection phase. To determine the final order, this selection method employs a variety of metrics. The selection approach suggested in this paper is an extension of previous works [38–40] that also considers the fairness criterion.

The fair-share principle [40] is one of the indicators utilised by the selection technique. Ref. [27] develops a fair factor for parallel job scheduling algorithms. This factor seeks to provide criteria for evaluating the system's fairness to all users. This paper extends it to match the characteristics of the ROA-CONS, taking into account more aspects of the parallel application and parallel machine. The definition of the fairness factor in ROA-CONS is as follows. This factor was employed as a decision criterion during the ROA-CONS selection phase.

Multiple jobs can be submitted to a machine by any user. The waiting time required for any job was defined in Definition 1. As a result, a user's overall waiting time is equal to the sum of the waiting times for all of his jobs.

Definition 5 (**User Total Waiting Time**)**.** *Assume $user_i$ is a system user (job owner), and $jobs_{user_i}$ is the collection of all jobs this user owns. Consider the waiting time for all jobs in $jobs_{user_i}$ as described in Definition 1. The cumulative waiting time for this user, labelled as twt_{user_i}, is computed as follows:*

$$twt_{user_i} = \sum_{\forall j \in jobs_{user_i}} Jwt_{J_i} \tag{4}$$

It is preferable in common parallel and distributed systems to prioritise less active users over those who have frequent requests and numerous resources [40]. Many real-world distributed systems, such as the Czech National Grid Infrastructure MetaCentrum [41], have implemented this logic. When the user waiting time is normalised, this prioritisation is applied to the users. The normalised waiting time for the user (nwt_{user_i}) is therefore defined as follows.

$$nwt_{user_i} = \frac{twt_{user_i}}{rb_{user_i}} \tag{5}$$

The normalised waiting time for a user is defined by Equation (5). A parallel machine, on the other hand, is used by several users. The average of waiting times (awt) provides a global view of all users' waiting times.

Definition 6 (**Average Normal Waiting Time**)**.** *Let Users represent the collection of all system users. According to the preceding definitions, the average normal waiting time of system users awt is calculated as follows:*

$$awt = \frac{1}{|Users|} \sum_{i=1}^{|Users|} nwt_{user_i} \tag{6}$$

The average waiting time for all users should be reduced through a fair scheduling method. Furthermore, users who request fewer resources should obtain the service faster. The Common Least Squares approach [42] aids in the creation of a single factor that reflects the system's average waiting time as well as the users' normalised waiting time.

Definition 7 (**Fairness Factor**)**.** *Inspiring the common Least Square method [42], the fairness factor FF is defined as:*

$$FF = \sum_{\forall u \in users} (awt - nwt_{user_i})^2 \tag{7}$$

The fairness aspect (FF) contributes to increased user satisfaction. However, from the standpoint of the system, a better order is one with lower waiting and response times. The selection algorithm employs a combination of user fairness and job response and waiting time to achieve an unbiased selection between user satisfaction and resource management. As a result, a selection weight $weight_{selection}$ is defined for the orders, which is utilised as a measure to choose between two orders.

The strategy utilised by multi-objective optimization methods to integrate the criteria ([38,39,43]) was utilised to produce the selection weight for two orders, $order_a$ and $order_b$. As a result, the selection weight has been defined as follows:

$$weight_{selection} = \left(\frac{\frac{1}{|order_a|}\sum_{\forall j \in order_a} Jwt_j - \frac{1}{|order_b|}\sum_{\forall j \in order_b} Jwt_j}{\frac{1}{|order_a|}\sum_{\forall j \in order_a} Jwt_j}\right) + \left(\frac{\frac{1}{|order_a|}\sum_{\forall j \in order_a} Jrt_j - \frac{1}{|order_b|}\sum_{\forall j \in order_b} Jrt_j}{\frac{1}{|order_a|}\sum_{\forall j \in order_a} Jrt_j}\right) + \left(\frac{FF_{order_a} - FF_{order_b}}{FF_{order_a}}\right) \tag{8}$$

In summary, the selection phase's pseudocode is provided in Pseudocode 2.

Pseudocode 2 Order Selection

```
Input:
Output: final order $order_{final}$
 1: procedure SELECTORDER($\Pi$, $H_{coarsest}$, $itr_{coarsening}$)
 2:     $vJwt \leftarrow (\frac{\frac{1}{|order_{fix}|}\sum_{\forall j \in order_{fix}} Jwt_j - \frac{1}{|order_{flexible}|}\sum_{\forall j \in order_{flexible}} Jwt_j}{\frac{1}{|order_{fix}|}\sum_{\forall j \in order_{fix}} Jwt_j});$
 3:     $vJrt \leftarrow (\frac{\frac{1}{|order_{fix}|}\sum_{\forall j \in order_{fix}} Jrt_j - \frac{1}{|order_{flexible}|}\sum_{\forall j \in order_{b}} Jrt_j}{\frac{1}{|order_{fix}|}\sum_{\forall j \in order_{flexible}} Jrt_j});$
 4:     $vFF \leftarrow \frac{FF_{order_{fix}} - FF_{order_flexible}}{FF_{order_{fix}}};$
 5:     $weight_{selection} \leftarrow vJwt + vJrt + vFF;$
 6:     if $weight_{selection} > 0$ then
 7:         $order_{final} \leftarrow order_{flexible};$
 8:     else
 9:         $order_{final} \leftarrow order_{fix};$
10:     end if
11: end procedure
```

Finally, in Pseudocode 3 the overall scheduling process in ROA-CONS has been given.

Pseudocode 3 ROA-CONS Scheduling

```
Input: Application Model Hypergraph $\mathcal{H} = (\mathcal{T}, \mathcal{D}, w_{\mathcal{H}}, c_{\mathcal{H}})$, Topology Model Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w_{\mathcal{G}}, c_{d_{(i,j)}}, c_{b_{(i,j)}})$, Current Schedule $S$, Incoming Job $J$, Set of users $Users$
Output: Updated Schedule $schedule_{updated}$
 1: procedure SELECTORDER($H_{coarsest}$, $itr_{coarsening}$)
 2:     $\Pi \leftarrow$ HATS$(\mathcal{H}, \mathcal{G});$
 3:     $order_{fix} \leftarrow$
    CONSERVATIVEBACKFILLING$(S, J, \mathcal{H}, \mathcal{G}, \Pi);$
 4:     $order_{flexible} \leftarrow$ ROA$(order_{fix}, \mathcal{H}, \mathcal{G}, itr, limit);$
 5:     $schedule_{updated} \leftarrow$
    SELECTORDER$(order_{fix}, order_{flexible}, Users, \mathcal{H}, \mathcal{G});$
 6: end procedure
```

4. Simulation and Evaluation

Simulation is usually a common way for assessing novel job scheduling algorithms. In the literature, there are various high-performance computer (HPC) simulators, such as GridSim [44], BigSim [45], and Performance Prediction Toolkit (PPT) [46]. GridSim is a well-known and commonly utilised simulator that is used by numerous researchers [47–49]. It is a simulator for grids, parallel high-performance clusters, and peer-to-peer networks. This simulator perfectly simulates heterogeneous systems [47]. However, it is difficult to properly configure it to imitate HPC job planning.

The Alea [50] is a simulator that emulates scheduling in parallel HPC clusters based on the newest GridSim (GridSim v5.0) toolkit. It inherits the features of the GridSim simulator, making the job scheduling and mapping challenges on HPCs easier to set and run. Alea is a modular simulator that has several components to imitate real-world HPC computers, centralized schedulers, and task assignment systems. The broad overview of the structure of Alea is shown in Figure 2.

The latest Alea simulator (Alea v4.0) is used in the paper to simulate the scheduling and mapping in heterogeneous parallel architectures. To mimic the scheduling, Alea needs valid datasets. These datasets, known as workloads, are logs of job execution collected from various parallel systems around the world. These logs are in Standard Workload Format (SWF) [51], which contains a great deal of information on the jobs submitted to the parallel machine, such as submission time, waiting time, execution time, and the number of allocated processors.

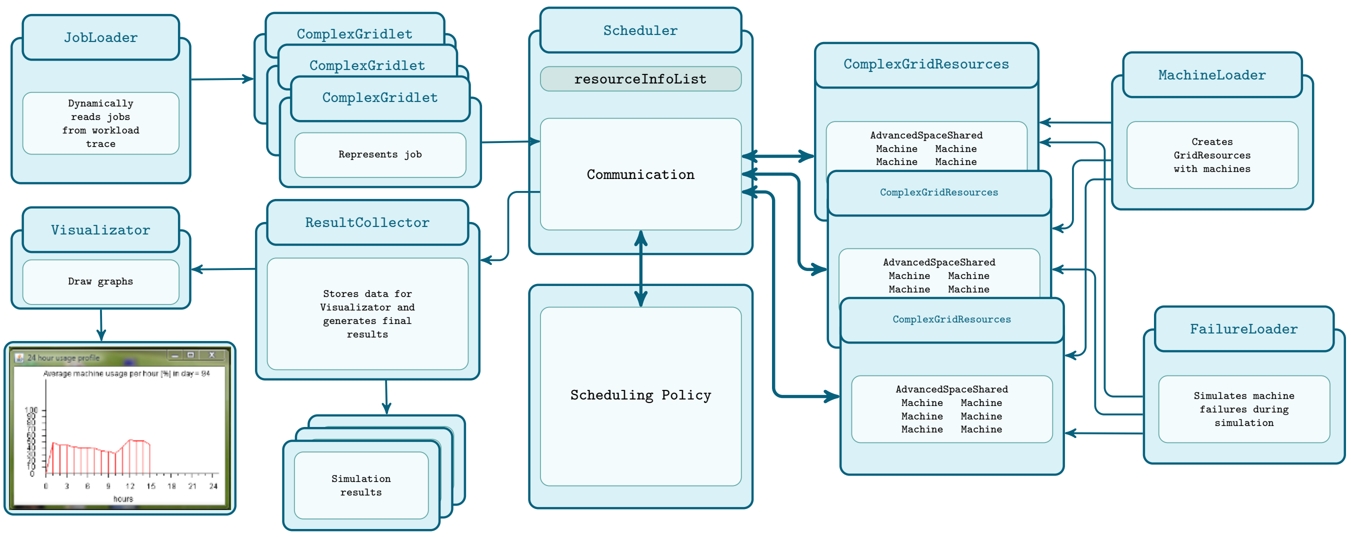
JobLoader
Dynamically reads jobs from workload trace
ComplexGridlet
ComplexGridlet
ComplexGridlet
Represents job
Scheduler
resourceInfoList
Communication
Scheduling Policy
ComplexGridResources
AdvancedSpaceShared Machine Machine Machine Machine
ComplexGridResources
AdvancedSpaceShared Machine Machine Machine Machine
ComplexGridResources
AdvancedSpaceShared Machine Machine Machine Machine Machine Machine
MachineLoader
Creates GridResources with machines
FailureLoader
Simulates machine failures during simulation
Visualizator
Draw graphs
ResultCollector
Stores data for Visualizator and generates final results
Simulation results
24 hour usage profile
Average machine usage per hour [%] in day = 84
hours

Figure 2. An overview of the structure of the Alea simulator

In addition to workloads, this simulator requires a description of the target HPC on which the specified workload will run. Another configuration file is used to specify the target machine's structure, which includes HPC details such as the number of clusters, their names, the number of nodes per cluster, the number of CPUs per node and their speeds, and RAM per node. This configuration is a text-based file with one line for each of the parallel distributed machine's clusters. Pseudocode 4 contains a sample description of an HPC with two clusters.

Pseudocode 4 Machine Configuration File in Alea [41]

```
//cluster_id   name     no_nodes   CPUs_per_node   CPU_speed   RAM_per_node_(KB)
    0          zewura      20           80            3.2          529426432
    1          zegox       48           12            2.4          94035968
```

The proposed Raccoon Optimization Job Scheduling approach (ROA-CONS) has been assessed using Alea simulation. This section summarises the findings of this evaluation. Alea has also simulated three other common schedulers, scheduling the identical datasets. The ROA-CONS findings were compared to those of other schedulers using two primary metrics: job waiting time and job response time.

4.1. Benchmarks Datasets and Algorithms

Alea needs a proper workload in SWF format with the description of the target HPC, as indicated in the preceding section. Two alternative workloads, Wagap and Zewura, are used in this simulation. These datasets are real workloads from the Czech National Grid Organization's MetaCentrum parallel structure [41], and they are available via the Parallel Workload Archive [52].

The Wagap data set comprises 17,900 jobs on a high-performance computer with two clusters. For this dataset, the target machine is shown in Table 2.

Table 2. HPC Configurations for Wagap Workload.

#	Nodes	CPUS Per Node	CPU Speed	RAM Per Node (kb)
1	20	80	3.2 GHz	529,426,432
2	48	12	2.4 GHz	94,035,968

Zewura, the second dataset, comprises 17,257 jobs. This workload has been applied to one HPC cluster. In Table 3 the configurations for this HPC are provided.

Table 3. HPC Configurations for Zewura Workload.

#	Nodes	CPUS Per Node	CPU Speed	RAM Per Node (kb)
1	20	4	3.2 GHz	536,870,912

The simulation is conducted with a ROA-CONS scheduling algorithm in the same environment together with three more scheduling techniques. Conservative backfilling (CONS), Easy backfilling (BF-EASY) and Conservative backfilling Tabu Search (CONS+Tabu) are the benchmark algorithms that are utilised for evaluation of ROA-CONS.

4.2. Results and Discussion

The simulation results are reported in this section. The proposed scheduling approach (ROA-CONS) is tested on Alea with the three previously stated algorithms. The outcomes recorded two key metrics: job waiting time and job response time. The charts below summarise the average outcomes of ten simulations for each scheduling algorithm. Figure 3 depicts the average job waiting time, whereas Figure 4 depicts the average response time.

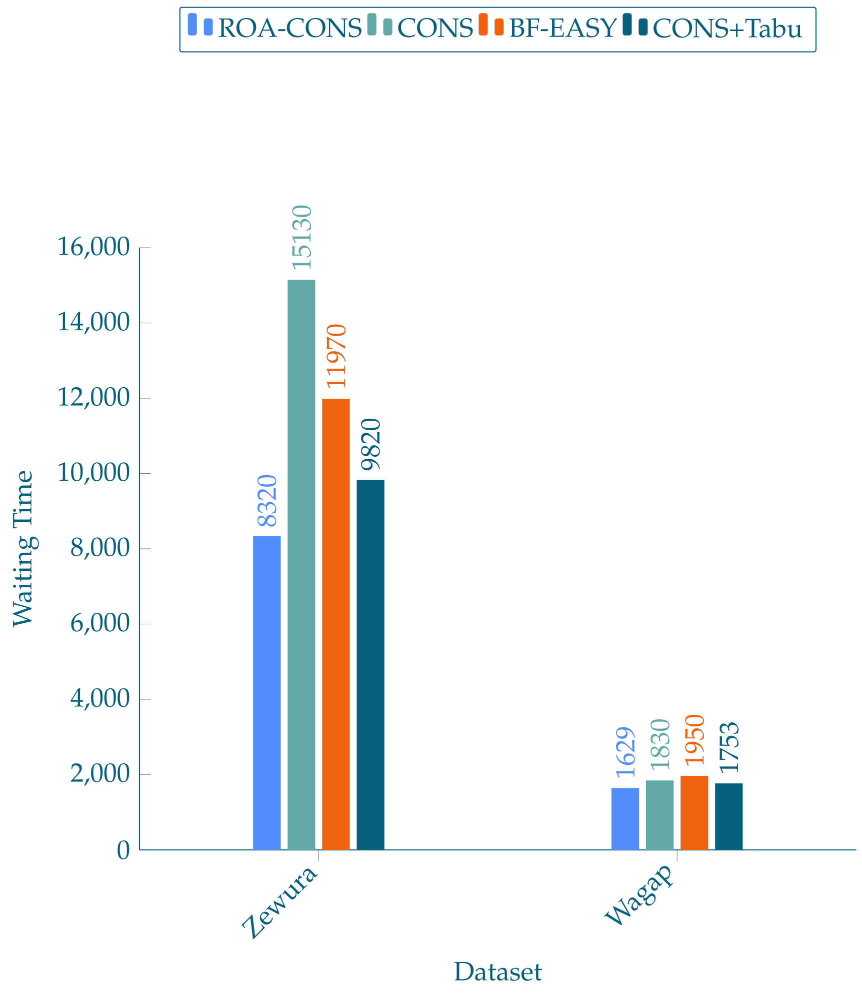

Figure 3. Waiting time for datasets.

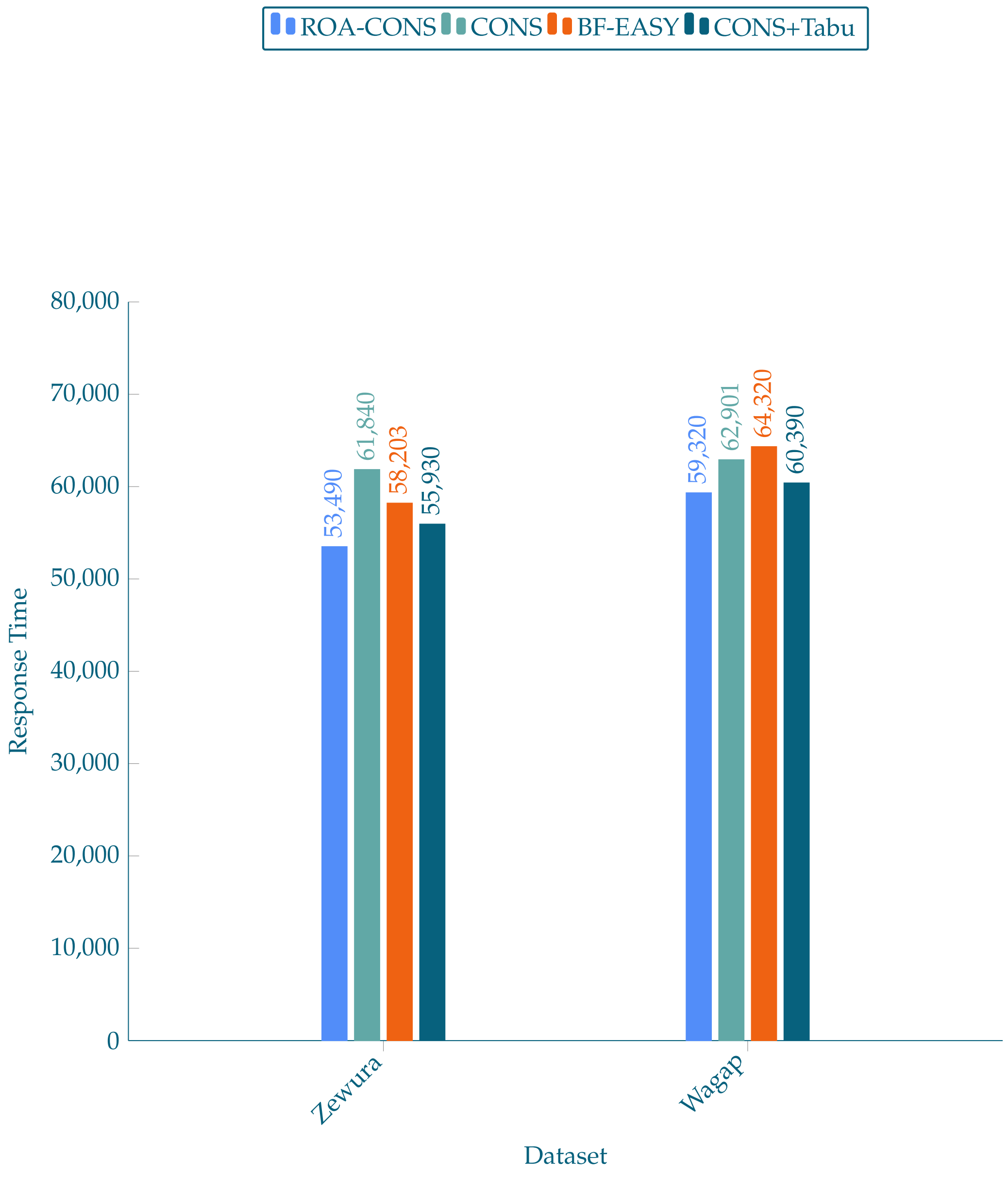

Figure 4. Response time for datasets.

According to the simulation results, ROA-CONS aided in achieving a shorter waiting time and response time. Previous research [18,22–24], as well as our findings, demonstrates that conservative backfilling has a lesser efficiency in terms of optimising job waiting time. This approach is a fixed technique, and as explained in Section 3, employing solely a fixed technique does not reduce waiting time [25].

Furthermore, the CONS reservation approach is less efficient than the aggressive and meta-heuristic methods employed in the BF-EASY and CONS-Tabu approaches, respectively [20,28,29]. Furthermore, BF-EASY employs a locking system, which has a significant impact on reserving jobs and reduces the options for backfilling [19,20].

CONS-Tabu provides relatively better outcomes in terms of job waiting time than the other two benchmark scheduling algorithms. This technique employs the Tabu optimization technique and improves on the BF-EASY approach, resulting in a better schedule than BF-EASY.

ROA-CONS employs both fixed and flexible methods. It generates a fixed order initially, then optimises it via ROA, and finally chooses the best order between them. Using ROA allows the scheduler to re-arrange jobs in a fixed order to save waiting time. Furthermore, the selection algorithm evaluates the orders based on their fairness factor as well as their average waiting and response time. All of this contributes to the selection of an order with the smallest amount of waiting time for all users.

ROA-CONS benefits from the Model of Exascale Message-Passing Programs on Heterogeneous Architectures (MEMPHA), which was introduced in [32]. MEMPHA assists ROA-CONS in delivering excellent details about the application and topology, allowing them to make more informed judgments. Although the process of scheduling jobs has a significant impact on resource management, the execution pattern of each job, which consists of several tasks, is critical. The ROA-CONS technique assigns job tasks to processors using the partitioning structure supplied by Heterogeneity Aware Task Scheduling (HATS) [33].

Experiments on HATS indicated that it can minimise the execution time of a parallel programme by using topology-aware partitioning. As a result, it contributes to faster job execution in ROA-CONS. As stated in Definition 3, a job's response time is the sum of its waiting time and execution time. As a result, by shortening the execution time, HATS reduces the job's response time. It is also one of the reasons behind the faster response time gained by using ROA-CONS.

Faster execution and response times imply that the parallel machine must run for a shorter length of time to execute the parallel application. It indicates that the parallel application will consume fewer resources, such as energy, on the HPC. In this way, ROA-CONS aids in parallel machine resource management and optimises resource consumption.

5. Conclusions

This paper introduces a job scheduling approach based on the Raccoon Optimization Algorithm (ROA-CONS). This scheduling method is a high-level combination of fixed and flexible techniques. This method employs two scheduling strategies. The use of Conservative Backfilling first offers a fixed order. The Raccoon Optimization algorithm then reorders the fixed order to get a flexible order. Finally, a selection algorithm chooses the best order among the fixed and flexible alternatives.

ROA-CONS optimises orders from the perspective of the parallel machine by utilising job waiting and job response time. Furthermore, it employs the fairness factor to determine an order that is preferable in the eyes of the users (job owners). As a result, ROA-CONS develop balanced schedules that are optimised for both the machine and the user. It provides a symmetrical allocation that achieves a balance between job waiting and response time reduction and user fairness.

The proposed scheduling algorithm's performance is evaluated using simulation. The Alea v4.0 Job Scheduling simulator simulates ROA-CONS as well as other well-known scheduling approaches. The input benchmark data is real-world workloads, and the results are compared based on job waiting and response times.

In simulations, two workloads are used: Zewura and Wagap. In terms of waiting time, the results showed that, in the best situation, ROA-CONS managed to enhance performance by 45.01 percent when compared to the CONS scheduling algorithm on Zewura workload. In the worst-case scenario, it improved performance by 15.28 percent over the CONS-TABU scheduling technique. Furthermore, on Wagap workload, ROA-CONS enhanced schedules by 16.47 percent in the best scenario (compared to BF-EASY) and 7.08 percent in the worst scenario (compared to the CONS-TABU method).

The second parameter measured was the job response time. In this parameter, ROA-CONS enhanced the outcomes in Zewura workload by 13.51 percent in the best case (compared to CONS) and 4.37 percent in the worst case (compared to CONS-TABU). ROA-CONS, on the other hand, scheduled Wagap workload with a 7.78 percent improvement in the best scenario (relative to BF-EASY) and a 1.78 percent improvement over CONS-TABU (worst case).

In general, ROA-CONS produced better results than alternative scheduling approaches, resulting in less waiting time and response time for jobs sent to the parallel computer. This enhancement will reduce user waiting periods and aid to lower the parallel machine's energy consumption by shortening its execution time.

A simulation method is used to evaluate the proposed method in this paper. The authors intend to implement and test the approach on genuine parallel and distributed machines in the future. Extending this technique to cover cloud-based and grid-based systems is another area for development that the authors will investigate in the future.

Author Contributions: Conceptualization, S.Z.K.; data curation, S.Z.K.; formal analysis, S.Z.K.; investigation, S.Z.K.; methodology, S.Z.K.; software, S.Z.K.; validation, M.O. and G.I.; writing—original draft, S.Z.K.; writing—review and editing, S.Z.K.; funding acquisition, N.A.W.A.H.; project administration, N.A.W.A.H.; resources, N.A.W.A.H.; supervision, N.A.W.A.H. and M.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Geran Putra (GP/2020/9693400) funded by Universiti Putra Malaysia (UPM).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

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

References

1. Deveci, M.; Kaya, K.; Uçar, B.; Çatalyürek, Ü.V. Fast and high quality topology-aware task mapping. In Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium, Hyderabad, India, 25–29 May 2015; pp. 197–206.
2. Amaral, M.; Polo, J.; Carrera, D.; Seelam, S.; Steinder, M. Topology-aware gpu scheduling for learning workloads in cloud environments. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, CO, USA, 12–17 November 2017; pp. 1–12.
3. Tuncer, O.; Leung, V.J.; Coskun, A.K. Pacmap: Topology mapping of unstructured communication patterns onto non-contiguous allocations. In Proceedings of the 29th ACM on International Conference on Supercomputing, Newport Beach, CA, USA, 8–11 June 2015; pp. 37–46.
4. Georgiou, Y.; Jeannot, E.; Mercier, G.; Villiermet, A. Topology-aware job mapping. *Int. J. High Perform. Comput. Appl.* **2018**, *32*, 14–27. [CrossRef]
5. Wang, W.; Dey, T.; Mars, J.; Tang, L.; Davidson, J.W.; Soffa, M.L. Performance analysis of thread mappings with a holistic view of the hardware resources. In Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems & Software, New Brunswick, NJ, USA, 1–3 April 2012; pp. 156–167.
6. Hamid, N.A.W.A.; Coddington, P. Comparison of MPI benchmark programs on shared memory and distributed memory machines (point-to-point communication). *Int. J. High Perform. Comput. Appl.* **2010**, *24*, 469–483. [CrossRef]
7. Henderson, R.L. Job scheduling under the portable batch system. In *Workshop on Job Scheduling Strategies for Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 1995; pp. 279–294.
8. Azmi, Z.R.M.; Bakar, K.A.; Abdullah, A.H.; Shamsir, M.S.; Manan, W.N.W. Performance comparison of priority rule scheduling algorithms using different inter arrival time jobs in grid environment. *Int. J. Grid Distrib. Comput.* **2011**, *4*, 61–70.
9. Hovestadt, M.; Kao, O.; Keller, A.; Streit, A. Scheduling in HPC resource management systems: Queuing vs. planning. In *Workshop on Job Scheduling Strategies for Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 1–20.
10. Somasundaram, K.; Radhakrishnan, S. Task resource allocation in grid using swift scheduler. *Int. J. Comput. Commun. Control* **2009**, *4*, 158–166. [CrossRef]
11. Mondal, R.K.; Nandi, E.; Sarddar, D. Load balancing scheduling with shortest load first. *Int. J. Grid Distrib. Comput.* **2015**, *8*, 171–178. [CrossRef]
12. Oskooei, A.R.; Mirza-Aghatabar, M.; Khorsandi, S. Introduction of novel rule based algorithms for scheduling in grid computing systems. In Proceedings of the 2008 Second Asia International Conference on Modelling & Simulation (AMS), Kuala Lumpur, Malaysia, 13–15 May 2008; pp. 138–143.
13. Lifka, D.A. *An Extensible Job Scheduling System for Massively Paralell Processor Architectures*; Illinois Institute of Technology: Chicago, IL, USA, 1998.
14. Singla, M.K.; Scholar, M. Task Scheduling Algorithms for Grid Computing with Static Jobs: A Review. *Int. J. Comput. Sci. Eng.* **2013**, *2*, 218.
15. Vijayvargiya, P. A Comparative Study of CPU Scheduling Algorithms. *arXiv* **2019**, arXiv:1307.4165.
16. Tong, C.; Wong, S. A schedule-based time-dependent trip assignment model for transit networks. *J. Adv. Transp.* **1999**, *33*, 371–388. [CrossRef]
17. Nuzzolo, A. Schedule-based transit assignment models. In *Advanced Modeling for Transit Operations and Service Planning*; Elsevier Science New York: New York, NY, USA, 2002; pp. 125–163.
18. Mu'alem, A.W.; Feitelson, D.G. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. *IEEE Trans. Parallel Distrib. Syst.* **2001**, *12*, 529–543. [CrossRef]

19. Feitelson, D.G. Experimental analysis of the root causes of performance evaluation results: A backfilling case study. *IEEE Trans. Parallel Distrib. Syst.* **2005**, *16*, 175–182. [CrossRef]
20. Srinivasan, S.; Kettimuthu, R.; Subramani, V.; Sadayappan, P. Selective reservation strategies for backfill job scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 55–71.
21. Li, B.; Zhao, D. Performance impact of advance reservations from the grid on backfill algorithms. In Proceedings of the Sixth International Conference on Grid and Cooperative Computing (GCC 2007), Xinjiang, China, 16–18 August 2007; pp. 456–461.
22. Gómez-Martín, C.; Vega-Rodríguez, M.A.; González-Sánchez, J.L. Fattened backfilling: An improved strategy for job scheduling in parallel systems. *J. Parallel Distrib. Comput.* **2016**, *97*, 69–77. [CrossRef]
23. Keleher, P.J.; Zotkin, D.; Perkovic, D. Attacking the bottlenecks of backfilling schedulers. *Clust. Comput.* **2000**, *3*, 245–254. [CrossRef]
24. Lifka, D.A. The anl/ibm sp scheduling system. In *Workshop on Job Scheduling Strategies for Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 1995; pp. 295–303.
25. Feitelson, D.G.; Rudolph, L.; Schwiegelshohn, U.; Sevcik, K.C.; Wong, P. Theory and practice in parallel job scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 1997; pp. 1–34.
26. Groves, T.; Knockel, J.; Schulte, E. *Bfs vs. Cfs Scheduler Comparison*; The University of New Mexico: Albuquerque, NM, USA, 2009; Volume 11.
27. Klusácek, D.; Rudová, H. Performance and fairness for users in parallel job scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 235–252.
28. Srinivasan, S.; Kettimuthu, R.; Subramani, V.; Sadayappan, P. Characterization of backfilling strategies for parallel job scheduling. In Proceedings of the International Conference on Parallel Processing Workshop, Vancouver, BC, Canada, 21 August 2002; pp. 514–519.
29. Ngubiri, J. Techniques and Evaluation of Processor Co-Allocation in Multi-Cluster Systems. Ph.D. Thesis, Department of Computer Science, Faculty of Computing and Information Technology, Makerere University, Kampala, Uganda, 2008.
30. Frachtenberg, E.; Feitelson, D.G. Pitfalls in parallel job scheduling evaluation. In *Workshop on Job Scheduling Strategies for Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 257–282.
31. Xhafa, F.; Kolodziej, J.; Barolli, L.; Fundo, A. A ga+ ts hybrid algorithm for independent batch scheduling in computational grids. In Proceedings of the 2011 14th International Conference on Network-Based Information Systems, Tirana, Albania, 7–9 September 2011; pp. 229–235.
32. Koohi, S.Z.; Hamid, N.A.W.A.; Othman, M.; Ibragimov, G. MEMPHA: Model of Exascale Message-Passing Programs on Heterogeneous Architectures. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 2570–2581. [CrossRef]
33. Koohi, S.Z.; Hamid, N.A.W.A.; Othman, M.; Ibragimov, G. HATS: Heterogeneity-Aware Task Scheduling. *IEEE Trans. Cloud Comput.* **2021**. Unpublished.
34. Koohi, S.Z.; Hamid, N.A.W.A.; Othman, M.; Ibragimov, G. Raccoon optimization algorithm. *IEEE Access* **2018**, *7*, 5383–5399. [CrossRef]
35. Rauf, M.; Guan, Z.; Yue, L.; Guo, Z.; Mumtaz, J.; Ullah, S. Integrated planning and scheduling of multiple manufacturing projects under resource constraints using raccoon family optimization algorithm. *IEEE Access* **2020**, *8*, 151279–151295. [CrossRef]
36. Balamurugan, A.; Priya, M.D.; Malar, A.C.J.; Janakiraman, S. Raccoon optimization algorithm-based accurate positioning scheme for reliable emergency data dissemination under NLOS situations in VANETs. *J. Ambient Intell. Humaniz. Comput.* **2021**, *12*, 10405–10424. [CrossRef]
37. Tzanetos, A.; Dounias, G. A comprehensive survey on the applications of swarm intelligence and bio-inspired evolutionary strategies. *Mach. Learn. Paradig.* **2020**, *18*, 337–378.
38. Klusáček, D.; Rudová, H. Efficient grid scheduling through the incremental schedule-based approach. *Comput. Intell.* **2011**, *27*, 4–22. [CrossRef]
39. Klusáček, D.; Rudová, H.; Baraglia, R.; Pasquali, M.; Capannini, G. Comparison of multi-criteria scheduling techniques. In *Grid Computing*; Springer: Boston, MA, USA, 2008; pp. 173–184.
40. Kleban, S.D.; Clearwater, S.H. Fair share on high performance computing systems: What does fair really mean? In Proceedings of the CCGrid 2003, 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings, Tokyo, Japan, 12–15 May 2003; pp. 146–153.
41. MetaCentrum National Grid Infrastructure. 2019. Available online: https://www.metacentrum.cz (accessed on 1 October 2021).
42. Wolberg, J. *Data Analysis Using the Method of Least Squares: Extracting the Most Information from Experiments*; Springer Science & Business Media, Berlin/Heidelberg, Germany: 2006.
43. Xhafa, F.; Abraham, A. Computational models and heuristic methods for Grid scheduling problems. *Future Gener. Comput. Syst.* **2010**, *26*, 608–621. [CrossRef]
44. Buyya, R.; Murshed, M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurr. Comput. Pract. Exp.* **2002**, *14*, 1175–1220. [CrossRef]
45. Zheng, G.; Kakulapati, G.; Kalé, L.V. Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. In Proceedings of the 18th International Parallel and Distributed Processing Symposium, 2004, Proceedings, Santa Fe, NM, USA, 26–30 April 2004; p. 78.

46. Obaida, M.A.; Liu, J. Simulation of HPC job scheduling and large-scale parallel workloads. In Proceedings of the 2017 Winter Simulation Conference (WSC), Las Vegas, NV, USA, 3–6 December 2017, pp. 920–931.
47. Sulistio, A.; Buyya, R. A grid simulation infrastructure supporting advance reservation. In Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004), The Canterbury Hotel, San Francisco, CA, USA, 15–17 September 2004; Volume 11, pp. 9–11.
48. Qureshi, K.; Rehman, A.; Manuel, P. Enhanced GridSim architecture with load balancing. *J. Supercomput.* **2011**, *57*, 265–275. [CrossRef]
49. Chelladurai, S.R. Gridsim: A Flexible Simulator for Grid Integration Study. Master's Thesis, Universidad of Northern British Colombia, Prince George, BC, USA, 2017.
50. Klusáček, D.; Tóth, Š.; Podolníková, G. Complex job scheduling simulations with Alea 4. In Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques. ICST (Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), Prague, Czech Republic, 22–23 August 2016; pp. 124–129.
51. Chapin, S.J.; Cirne, W.; Feitelson, D.G.; Jones, J.P.; Leutenegger, S.T.; Schwiegelshohn, U.; Smith, W.; Talby, D. Benchmarks and standards for the evaluation of parallel job schedulers. In *Workshop on Job Scheduling Strategies for Parallel Processing*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 67–90.
52. Feitelson, D.G.; Tsafrir, D.; Krakov, D. Experience with using the parallel workloads archive. *J. Parallel Distrib. Comput.* **2014**, *74*, 2967–2982. [CrossRef]