



## Article

# AdPSO: Adaptive PSO-Based Task Scheduling Approach for Cloud Computing

Said Nabi <sup>1,2</sup> , Masroor Ahmad <sup>2</sup>, Muhammad Ibrahim <sup>3,4,\*</sup> and Habib Hamam <sup>5,6,7</sup> 

<sup>1</sup> Department of Computer Science, Virtual University of Pakistan, Rawalpindi 46300, Pakistan; said.nabi@vu.edu.pk

<sup>2</sup> Department of Computer Science, Capital University of Science & Technology (CUST), Islamabad 46300, Pakistan; masroor.ahmed@cust.edu.pk

<sup>3</sup> Department of Information Technology, University of Haripur, Haripur 22610, Pakistan

<sup>4</sup> Department of Computer Science and Statistics, Jeju National University, Jeju-si 63243, Korea

<sup>5</sup> Faculty of Engineering, Uni de Moncton, Moncton, NB E1A 3E9, Canada; habib.hamam@gmail.com

<sup>6</sup> Spectrum of Knowledge Production & Skills Development, Sfax 3027, Tunisia

<sup>7</sup> Department of Electrical and Electronic Engineering Science, School of Electrical Engineering, University of Johannesburg, Johannesburg 2006, South Africa

\* Correspondence: ibrahimmayar@uoh.edu.pk

**Abstract:** Cloud computing has emerged as the most favorable computing platform for researchers and industry. The load balanced task scheduling has emerged as an important and challenging research problem in the Cloud computing. Swarm intelligence-based meta-heuristic algorithms are considered more suitable for Cloud scheduling and load balancing. The optimization procedure of swarm intelligence-based meta-heuristics consists of two major components that are the local and global search. These algorithms find the best position through the local and global search. To achieve an optimized mapping strategy for tasks to the resources, a balance between local and global search plays an effective role. The inertia weight is an important control attribute to effectively adjust the local and global search process. There are many inertia weight strategies; however, the existing approaches still require fine-tuning to achieve optimum scheduling. The selection of a suitable inertia weight strategy is also an important factor. This paper contributed an adaptive Particle Swarm Optimisation (PSO) based task scheduling approach that reduces the task execution time, and increases throughput and Average Resource Utilization Ratio (ARUR). Moreover, an adaptive inertia weight strategy namely *Linearly Descending and Adaptive Inertia Weight (LDAIW)* is introduced. The proposed scheduling approach provides a better balance between local and global search leading to an optimized task scheduling. The performance of the proposed approach has been evaluated and compared against five renown PSO based inertia weight strategies concerning makespan and throughput. The experiments are then extended and compared the proposed approach against the other four renowned meta-heuristic scheduling approaches. Analysis of the simulated experimentation reveals that the proposed approach attained up to 10%, 12% and 60% improvement for makespan, throughput and ARUR respectively.

**Keywords:** meta-heuristic; PSO; inertia-weight; cloud; task scheduling; makespan; throughput



**Citation:** Nabi, S.; Ahmad, M.; Ibrahim, M.; Hamam, H. AdPSO: Adaptive PSO-Based Task Scheduling Approach for Cloud Computing. *Sensors* **2022**, *22*, 920. <https://doi.org/10.3390/s22030920>

Academic Editor: Subhas Mukhopadhyay

Received: 29 November 2021

Accepted: 21 January 2022

Published: 25 January 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Cloud Computing has revolutionized computing technology, where computing resources are accessed globally through the Internet [1]. These resources are provided in the form of services that can be easily and dynamically scaled-up and scaled-down by the Cloud users according to their needs [2,3]. The Cloud services are provided on a pay-as-go basis [4] to the users. The Cloud service model consists of Cloud service provider [5–7], Cloud user, and datacenter. A Cloud service provider acquires Cloud resources and provides these resources to the Cloud users according to their requirements. A Cloud

data center represents the computing power of a Cloud which may contain hundreds of thousands of host machines. Each host on the data center may have one or more Virtual Machines (VMs) [8].

VM is considered an essential component of the Cloud environment. It enables the optimal use of the host machines in the Cloud computing environment [9]. VM provides flexibility to the Cloud operator to increase or decrease the number of CPUs, computation power of CPU, memory, and bandwidth according to its need. To select the most suitable resource among the resource pool for user's jobs, task schedulers play a key role. Task scheduling [4,7,10] is a principal component of the Cloud environment and the most challenging issue that needs to be optimized which is termed as an NP-hard problem [11].

Task scheduling approaches are categorized as heuristics, meta-heuristic algorithms, and hybrid of meta-heuristics and others approaches like heuristics and Machine learning among others. Heuristic-based algorithms provide near to optimal solutions for a specific problem. However, the meta-heuristics approaches are specifically designed for generalized optimal solutions that can be applied to multiple domains. Hybrid approaches are the combination of meta-heuristics, heuristics or machine learning based techniques for solving load-balanced task scheduling in Cloud computing [12–14]. This research focuses on meta-heuristics task scheduling algorithms.

The meta-heuristic based task scheduling schemes are divided into categories that are evolutionary-based like Genetic Algorithm (GA) [15], bio-meta-heuristics (swarm Intelligence-based), and non-bio-meta-heuristics like Simulation-Based Optimization (SBO) and Simulated Annealing (SA) [16]. Swarm Intelligence (SI) [17] is a sub-domain of computational intelligence and first used this concept by [18]. SI aims to solve computational problems by modeling self-organized populations of agents that can interact with each other. Agents can share their experiences by exchanging information. The interactions and movements of agents represent the population performance [19].

SI was first used by [18] for robotic intelligence in cellular robotic systems. After that, the definition of SI is expanded by [20] for algorithms and solving distributed problems. Currently SI is used for solving problems in various domains like health care, diagnosing diseases [21], stock analysis [22], academics [5], fraud and intrusion detection, feature selection [23], solving real-world engineering problems [24], pipe and road problems [25], data classification [26], Recommender Systems [12,13], and distributed computing and Cloud computing [27–29].

SI algorithms are broadly categorized into two sub-categories [30] i.e., (1) Sign based SI that is Bee Colony Optimization (BCO) and Ant Colony Optimization (ACO) [31], (2) Imitation based SI algorithms comprise of Cat Swarm Optimization (CSO) [32], Raven Roosting Optimization (RRO) [33], Improved RRO (IRRO) [30], Particle Swarm Optimization (PSO) [27,34,35], and Chicken Swarm optimization (CSO) [36] algorithms among others. The literature study shows that PSO is the most adopted optimization algorithm [19] for Cloud task scheduling.

Genetic Algorithm (GA) [15], SA [16], ACO [31], and PSO are the renowned meta-heuristic techniques use for cloud task scheduling. The meta-heuristic approach like ACO perform better at the early stage of optimization but converge slowly at later stage. PSO based schedulers perform better optimization than GA, has more natural computation background, fast convergence, and easy implementation as compared to GA.

One of the key factors in Cloud task scheduling is their computation time. Being a dynamic computation platform, Cloud schedulers should be fast and adoptable to the real Cloud platform. These scheduling schemes should have fast convergence and provide optimized solution. Therefore, this research focuses on optimizing tasks execution time, ARUR, and throughput [11] using PSO.

PSO can be applied to both discrete and continuous problems and is more efficient for global search in the problem space. PSO converges globally and tries to find a comparatively better fitness value. However, PSO is weak for local search and cannot pay more attention to the search in the local subspace. This increase the chances of trapping to the local optima

which may have a lower convergence in the later stages. To overcome these limitations of PSO algorithms, inertia weights play a significant role. Inertia weight is an important control parameter for effectively adjusting the local and global search capability of the PSO scheduler. A small value of the inertia weight improves the local search while a bigger value of weight facilitates the global search. The literature study shows that there are five prominent and most commonly used inertia weights strategies out of a total of 15 inertia weights.

In this research, we contributed an adaptive meta-heuristic-based task scheduling approach which minimizes tasks execution time, enhances Cloud resource utilization, and throughput. The proposed approach favors the compute-intensive and independent Cloud tasks [37]. Moreover, a novel Inertia weight strategy i.e., *Leaner Descending and Adaptive Inertia Weight* (LDAIW) for PSO-based algorithms have been proposed. The proposed inertia weight strategy improves the performance of the PSO approach concerning the makespan [11], ARUR, and throughput. This is because the proposed technique provides a better integration of local and global search. The performance of the proposed approach has been investigated and compared against five inertia weight strategies. These weights are evaluated for optimizing the makespan, ARUR, and throughput in Cloud task scheduling. The major contribution of this research includes:

- In-depth study and critical analysis of state-of-the-art meta-heuristics based cloud task scheduling schemes to ascertain their application types, scheduling objectives, and limitations of these algorithms.
- Empirical evaluation of the most prominent and state-of-the-art inertia strategies for PSO-based algorithms.
- A adaptive PSO-based meta-heuristic task scheduling approach is proposed that reduces makespan, improves the resource utilization and throughput.
- A novel inertia weight strategy named *Leaner Descending and Adaptive Inertia Weight* (LDAIW) is designed and developed that improves the performance of PSO-based algorithms concerning makespan, throughput and ASRUR.
- A comparative experimental performance evaluation of AdPSO has been performed against their counterparts.

The rest of the paper is structured as follows: Section 2 presents the details concerning the related task scheduling work followed by Section 3 that delineates the details of the proposed approach. This section discuss the SWARM intelligence-based algorithms, PSO model, System model, Inertia weight strategy, proposed Inertia weight strategy, and proposed Task scheduler. The experimental configuration setup, dataset details and results and discussions are presented in Section 4. Section 5 discuss the conclusion and future work.

## 2. Related Work

The discussion concerning the state-of-the-art meta-heuristic task scheduling approaches is presented as follows. In [38] a hybrid load-balancing approach is proposed that combines Teaching Learning Based Optimization (TLBO) algorithms with the Grey Wolves Optimization (GWO) approach. The proposed approach combines the strengths of GWO and TLBO algorithms to effectively balance the load based on the time and related cost. This approach also reduces task waiting time in the tasks queue. However, throughput is not considered.

The authors in [39] have designed and developed a deadline aware task scheduling approach for Cloud Computing. The scheduling scheme has used the GA algorithm to enhance the execution time and cost of resources by considering variation in VM performance and acquisition delay. However, GA faces scalability issues for a large and complex problem.

Look-Ahead Genetic Algorithm (LAGA) is a modified form of the Genetic Algorithm (GA) that has been proposed by [1] for large-scale distributed systems such as Cloud and Grid computing environments. LAGA is considered suitable for run-time based scheduling

of compute-intensive tasks and reliability. This approach identifies task orders based on the completion time of resources in every generation and chooses the resource that has a minimum failure rate during the mutation step. The reliability and task failure rate are the scheduling objectives of this approach. However, this approach does not consider makespan and throughput as scheduling criteria.

The authors have proposed a Node duplication-based Genetic Algorithm (NGA) is a GA-based algorithm designed for multi-processor heterogeneous systems [8]. The focus of NGA is on communication delay time and application completion time of the resources. The fitness function of this algorithm evolves in two steps. (1) Fitness of tasks that provided information regarding all other tasks to the system is scheduled for the legal order. Here the legal system schedules a pair of independent tasks on a single processor. (2) In this step, NGA looks at the processor fitness that tries to execute the task in minimum time. NGA inherits core issues of scalability for large and complex problems from the genetic algorithm.

The author in [40] has performed a comparison between the GA and PSO algorithms by using several test cases. It has been observed that in the majority of test cases, the PSO algorithm provides a better quality solution in a faster way than GA. Based on their experiments, it is claimed that for distributed systems the performance of the PSO algorithm is better than GA.

Author in [9] has proposed a task scheduling approach that combines Gravitational Emulation Local Search (GELS) and PSO. This approach aims to improve the makespan and tasks meeting their deadlines. However, throughput is not considered

In [41] authors have analyzed the PSO-based task scheduling algorithms in Cloud. The authors have classified the current PSO-based research work on the basis of the no of objectives that need to be optimized. This categorization of PSO algorithms includes a single objective or multiple objectives. To improve the solution quality, most of the researchers have applied basic PSO or updated PSO. The author has concluded that balanced scheduling and meeting Quality of Service (QoS) requirements (i.e., makespan, throughput, and resource utilization, etc.) required more focus and improvement. Inertia weight has not been considered for evaluation and analysis.

Authors in [30] has proposed IRRO and CSO based meta-heuristic algorithms. The proposed technique combines the strengths of CSO and IRRO algorithms that help in balancing the global and local search process. This approach has also proposed a dynamic scheduling framework named IRRO-CSO based Dynamic Scheduling Framework (ICDSF). Response time, premature convergence, makespan, and throughput have been used as evaluation parameters. However, ARUR is not considered as an evaluation parameter.

RTPSO-B a Rang and Tune based PSO with Bat technique has been proposed in [42]. RTPSO-B is an enhanced PSO-based algorithm that improves the efficiency of task scheduling in the Cloud. This algorithm solves the inertia weight issue of the existing PSO by introducing the data locality technique. The small inertia weight assists local search and the large inertia weight assists the global search process. For better optimization, the PSO approach has been combined with the Bat algorithm. Utilization of Cloud resources, makespan, cost are the key evaluation parameters however, throughput is not considered for evaluating scheduling algorithms.

Integer-PSO [43] is a discrete version of PSO based tasks scheduling algorithms in Cloud. Integer-PSO can be used for both single and multiple objectives optimization-based task scheduling in Cloud. The Integer-PSO has considered a bi-objective optimization problem. These objectives include task execution time and computation and management costs. However, the Integer-PSO does not consider throughput as a task scheduling objective and a fixed value is used for inertia weight.

In [44] authors have proposed a honey bee and improvement detection operator based load balancing algorithm named Hyper-heuristic. The proposed approach has the capability to distribute the cloud-based workload among the virtual machines with minimized makespan time. Hyper-heuristic scheduling scheme is evaluated against state-

of-the-art in terms of makespan time, processing time, degree of imbalance. However, the Hyper-heuristic algorithm does not consider throughput and ARUR as scheduling objectives.

Authors have proposed a PSO based Task oriented Load Balancing (TBSLB-PSO) in [45]. TBSLB-PSO improves the load-balancing by migrating tasks from overload VMs to under-loaded VMs using the task migration technique. The proposed scheduling technique reduces the load balancing tasks by migrating tasks without stopping the overloaded VMs.

An adaptive Particle Swarm Optimization (APSO) based scheduling technique for Resource Constrained Project Scheduling Problems (RPSP) has been proposed in [46]. This technique aims to issue invalid particle generation. For this purpose, the authors have proposed a Valid Particle Generator (VPG) operator that is embedded with the PSO algorithm. The VPG convert the invalid particles to the valid one by in-degree and out-degree of activities in the directed acyclic graph. Moreover, the author has also proposed an adaptive inertia weight strategy using parameters like previous inertia weight, current iteration number, fitness value. Performance of the APSO is evaluated in terms of Makespan. However, throughput and ARUR are not used for evaluation.

Author in [28] has proposed a cloud task scheduling framework using a modified PSO (PSO-BOOST) based meta-heuristic algorithm. The proposed approach finds an optimized solution for conflicting objectives. This approach considers time, acceptance ratio, cost, and throughput as evaluation parameters. However, in this approach, the role and selection criteria of inertia weight have not been explicitly discussed. Moreover, a new compromise-optimized solution for conflicting metrics has been proposed using the principle of Pareto Optimal Theory (POT). ARUR is also not considered as an evaluation parameter.

Author in [24] has proposed an adaptive inertia weight approach based enhanced version of PSO. A set of ten (10) well-known test problems for optimization were used to evaluate and test their presented scheduling technique and four (4) other variants of PSO. The reason is that the performance of the PSO-based algorithms mostly depends on the selection of inertia weight strategy and optimal parameter setting. The proposed approach has also been evaluated for real-world engineering problems. This approach has been evaluated in terms of solution accuracy and convergence speed. Makespan and throughput are not considered as evaluation parameters.

Author in [47] has presented a review of different inertia weight strategies used by various researchers in their work. The author has classified these inertia weights into three groups includes time-varying, constant, and adaptive inertia weights strategies. The scheduling objective of this approach is the average makespan. However, throughput and ARUR are not considered.

Author in [48] has evaluated and compared five different inertia weights for the PSO algorithm. Makespan is considered as an objective function for the evaluation of inertia weights. The author has suggested that Linear Descending Inertia Weight (LDIW) performed better than other inertia weight strategies. However, throughput is not considered for evaluation. Table 1 presents a comparison of the existing task scheduling approaching highlighting the application type, strengths and weaknesses of each of the approach.

**Table 1.** Summary of the related work.

Approach	Application Type	Strengths	Weaknesses
GWO-TLBO [38]	11 benchmark functions	Consider Time and cost	Throughput not considered
GA [39]	Independent tasks	considering variation in VM performance and acquisition delay	Scalability issue and Throughput not considered
LAGA [1]	Independent tasks	Reduces the failure rate	Makespan and throughput not considered
NGA [8]	Workflow-based tasks	Support for communication delay and application completion time	scalability issue and Throughput not considered
GA vs PSO [40]	Test cases	Compared the performance of both PSO and GA	Makespan and throughput not considered
GELS-PSO [9]	10 well-known test problems	Improve makespan and maximize meeting task deadline	Throughput and ARUR not considered
PSO [41]	independent and workflow-based tasks	Consider both independent and workflow based workload for load balancing	Inertia weight strategy has not considered for analysis
ICDSF [30]	Independent tasks	Makespan, throughput and response time	ARUR not considered
RTPSO-B [42]	Independent tasks	ARUR, makespan, and cost	Throughput not considered
Integer-PSO [43]	Independent tasks	Support for makespan and cost	Throughput and ARUR is not considered and a constant value is used for inertia weight
PSO-BOOST [28]	independent tasks	considered throughput and conflicting parameters like makespan and cost	Role and selection criteria of inertia weight has not explicitly discussed, ARUR not considered
AIWPSO [24]	10 set of benchmark problems	Accuracy and convergence speed	Makespan and throughput not considered
PSO [47]	Workflow based tasks	Average makespan	Throughput and ARUR not considered
MIPSO [48]	Independent tasks	Makespan	Throughput and ARUR not considered

The in-depth investigation of the related literature shows that majority of the existing task scheduling approaches are evaluated using small datasets which is not enough to prove the scalability of these approaches. This is because scalability is an important factor in scheduling algorithms. Moreover, the inertia weight strategy is an important control parameter for PSO-based algorithms to balance the local and global search of particles. However, most of the existing scheduling techniques either used a constant value for Inertia weight or were not discussed explicitly. Similarly, the majority of the existing approach has not considered a makespan, ARUR, and throughput as scheduling objectives. To overcome these limitations, a novel and adaptive inertia weight strategy for PSO-based task scheduling algorithm has been proposed and compared with five most prominent inertia weight strategies and other PSO-based state-of-the-art task scheduling algorithms. The proposed approach uses four instances of a renowned HCSP based scientific benchmark dataset using makespan and throughput as scheduling objectives.

### 3. Proposed Approach

This Section delineates the methodology of our proposed task scheduling algorithm. The proposed methodology comprises the proposed inertia weight strategy and the proposed task scheduler. This section also describes the background knowledge of swarm based PSO algorithm.

### 3.1. PSO Model

PSO algorithm is a global search-based self-adaptive optimization approach [34]. This approach is population-based scheduling technique that relies on the social behavior of the particles. This is a swarm intelligence based approach that is inspired by the social behavior of the fish school and birds flock. The swarm population consists of generations and particles. Generations show the total number of iterations that need to be performed to get an optimized solution. Each generation has several particles and every particle in a generation shows a single solution. The number of generations and particles varies from case to case and is adjusted to get a more optimal solution. Every particle has a position, velocity, local/personal best ( $pBest$ ), and global best ( $gBest$ ). Personal best shows the most optimal solution of a particle while  $gBest$  shows the best solution among all particles. The velocity and position of all particles are updated in each iteration based on inertia weight,  $pBest$ , and  $gBest$  of the particle. Suppose  $D$  represents the dimension of solution space, where  $X_i$  is a vector that represents positions of particles in a search space i.e.,

$$X_i = (x_{i1}, x_{i2}, x_{i3}, \dots, x_{id}) \quad (1)$$

In every iteration, particles constantly change their position and search for a more suitable solution.

$$Pb_i = (pb_{i1}, pb_{i2}, pb_{i3}, \dots, pb_{id}) \quad (2)$$

The  $Pb_i$  represents the best solution for each particle (as depicted in Equation (2)).  $V_i$  shows the velocity of particles (as shown in Equation (3)).

$$V_i = (v_{i1}, v_{i2}, v_{i3}, \dots, v_{id}) \quad (3)$$

Equation (4) is used by the PSO to the updated velocity of each particle

$$v_{id}^{k+1} = w * v_{id}^k + c_1 r_1 (pb_{id}^k - x_{id}^k) + c_2 r_2 (Gb_d^k - x_{id}^k) \quad (4)$$

where  $i = 1, 2, 3, \dots, n$  (shows the no of particles),  $k = 1, 2, 3, \dots, itrmax$  (max. no of iterations that is 200 iterations in this article),  $D$  shows the number of dimensions or the no of tasks that need to be assigned to the VMs in an optimized manner.  $x_{id}^k$  is the current position and  $v_{id}^k$  is the current velocity of the  $i^{\text{th}}$  particle of  $k^{\text{th}}$  iteration in  $d$  dimensional space. The parameter  $w$  is the inertia weight that balances the global and local search of the particles,  $c_2$  and  $c_1$  are the constant acceleration factors.  $r_1$  and  $r_2$  are random values between 0 and 1. Equation (5) depicts the updated position of particles.

$$x_{id}^{k+1} = (v_{id}^{k+1} + x_{id}^k) \quad (5)$$

The best value of every individual element (personal best) is computed by the fitness function which is based on maximization or minimization problem. The fittest element of all the individuals is termed as global best.

$$FitnessFunction = Minimization(Objective) \quad (6)$$

The  $pBest$  value of each individual element is updated for each particle in all iterations if the new value is better than the current value. The PSO based heuristics computes and records the best value among all individuals (i.e.,  $gBest$  shown in the Equation (7)) in the swarms.

$$gBest = max(pBest_1, pBest_2, pBest_3, \dots, pBest_n) \quad (7)$$

To resolve the task scheduling issue using PSO-based heuristics and to enter a schedule as a search solution, identify suitable maps among PSO particles and problem solution [49]. Every particle of PSO indicates the possible solution for tasks to VM mapping. Tables 2–4 are used to illustrate mapping among PSO particles and problem solutions. Table 4 shows the tasks to VM mapping using PSO-based swarm intelligence. For this mapping, five VMs

with different processing power (in Million Instruction Per Second (MIPS)) are used. VM<sub>1</sub> has computation power of 50 MIPS, 100, 200, 350 and 500 MIPS that represents computation capability of VM<sub>2</sub>, VM<sub>3</sub>, VM<sub>4</sub> and VM<sub>5</sub> respectively (as shown in Table 2).

**Table 2.** Computation power of VMs.

VMs	VM <sub>1</sub>	VM <sub>2</sub>	VM <sub>3</sub>	VM <sub>4</sub>	VM <sub>5</sub>
VMs Computation power(in MIPS)	50	100	200	350	500

Table 3 shows 15 tasks with different computation requirements in term of Million Instructions (MIs) that need to be mapped on five VMs (Table 2).

**Table 3.** Computation requirements of Tasks.

Task	Tsk <sub>1</sub>	Tsk <sub>2</sub>	Tsk <sub>3</sub>	Tsk <sub>4</sub>	Tsk <sub>5</sub>	Tsk <sub>6</sub>	Tsk <sub>7</sub>	Tsk <sub>8</sub>	Tsk <sub>9</sub>	Tsk <sub>10</sub>	Tsk <sub>11</sub>	Tsk <sub>12</sub>	Tsk <sub>13</sub>	Tsk <sub>14</sub>	Tsk <sub>15</sub>
MIs	50	100	150	200	300	450	500	600	700	900	1200	1500	2000	3000	4000

Table 4 depicts the mapping of 15 tasks to 5 VMs, P11 (particle 1, iteration 1) represents the first possible solution, where Tsk<sub>5</sub>, Tsk<sub>10</sub>, Tsk<sub>15</sub> are allocated to VM<sub>1</sub>; Tsk<sub>4</sub>, Tsk<sub>9</sub>, Tsk<sub>14</sub> to VM<sub>2</sub>, Tsk<sub>6</sub>; Tsk<sub>13</sub> to VM<sub>3</sub>, Tsk<sub>3</sub>, Tsk<sub>7</sub>, and Tsk<sub>12</sub> to VM<sub>4</sub>; Tsk<sub>1</sub>, Tsk<sub>2</sub>, Tsk<sub>8</sub>, Tsk<sub>11</sub> are assigned to VM<sub>5</sub> respectively.

**Table 4.** Tasks to VM mapping.

Task	Tsk <sub>1</sub>	Tsk <sub>2</sub>	Tsk <sub>3</sub>	Tsk <sub>4</sub>	Tsk <sub>5</sub>	Tsk <sub>6</sub>	Tsk <sub>7</sub>	Tsk <sub>8</sub>	Tsk <sub>9</sub>	Tsk <sub>10</sub>	Tsk <sub>11</sub>	Tsk <sub>12</sub>	Tsk <sub>13</sub>	Tsk <sub>14</sub>	Tsk <sub>15</sub>
P11	VM <sub>5</sub>	VM <sub>5</sub>	VM <sub>4</sub>	VM <sub>2</sub>	VM <sub>1</sub>	VM <sub>3</sub>	VM <sub>4</sub>	VM <sub>5</sub>	VM <sub>2</sub>	VM <sub>1</sub>	VM <sub>5</sub>	VM <sub>4</sub>	VM <sub>3</sub>	VM <sub>2</sub>	VM <sub>1</sub>
P12	VM <sub>1</sub>	VM <sub>5</sub>	VM <sub>3</sub>	VM <sub>2</sub>	VM <sub>1</sub>	VM <sub>5</sub>	VM <sub>3</sub>	VM <sub>4</sub>	VM <sub>4</sub>	VM <sub>5</sub>	VM <sub>1</sub>	VM <sub>3</sub>	VM <sub>2</sub>	VM <sub>5</sub>	VM <sub>4</sub>
P13	VM <sub>2</sub>	VM <sub>1</sub>	VM <sub>3</sub>	VM <sub>3</sub>	VM <sub>5</sub>	VM <sub>4</sub>	VM <sub>5</sub>	VM <sub>3</sub>	VM <sub>1</sub>	VM <sub>4</sub>	VM <sub>2</sub>	VM <sub>5</sub>	VM <sub>4</sub>	VM <sub>2</sub>	VM <sub>3</sub>
P14	VM <sub>3</sub>	VM <sub>2</sub>	VM <sub>5</sub>	VM <sub>4</sub>	VM <sub>4</sub>	VM <sub>1</sub>	VM <sub>1</sub>	VM <sub>5</sub>	VM <sub>2</sub>	VM <sub>3</sub>	VM <sub>5</sub>	VM <sub>1</sub>	VM <sub>2</sub>	VM <sub>4</sub>	VM <sub>2</sub>
—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
P44	VM <sub>2</sub>	VM <sub>4</sub>	VM <sub>5</sub>	VM <sub>1</sub>	VM <sub>3</sub>	VM <sub>1</sub>	VM <sub>2</sub>	VM <sub>4</sub>	VM <sub>5</sub>	VM <sub>2</sub>	VM <sub>3</sub>	VM <sub>1</sub>	VM <sub>4</sub>	VM <sub>3</sub>	VM <sub>5</sub>

### 3.2. System Model

The objective of the task scheduling scheme is to choose the optimized task mapping strategy on the Cloud resources that can reduce the task execution time and improve ARUR, and the Cloud throughput. Considering user's applications that consists of a set of  $D$  independent and compute-intensive tasks i.e.,  $Tsk_s = Tsk_1, Tsk_2, Tsk_3, \dots, Tsk_d$  that need to be scheduled on a set of  $M$  VMs i.e.,  $T_s = T_1, T_2, T_3, \dots, T_d$  that need to be scheduled on a set of  $M$  VMs, i.e.,  $VMs = V_1, V_2, V_3, \dots, V_m$  where  $D \gg M$ . For instance, Table 2 shows VMs ( $M = 5$ ) with their computation power in MIPS that needs to execute 15 tasks (i.e.,  $D = 15$ ) with their computation requirements in MI. When task  $T_D$  are mapped to  $VM_j$ , the Expected Completion Time (ETC) of the assigned task is calculated using Equation (8):

$$ETC_{dj} = RT_j + EET_{dj} \quad (8)$$

where  $RT_j$  is the ready time of  $VM_j$ , i.e., the time needed for VM to complete already assigned workload.  $EET_{dj}$  is the Expected Execution Time (EET) of task  $Tsk_D$  on  $VM_j$  which is obtained by dividing task computation requirements (in MI) by computation power of VMs (in MIPS) and is described by formula given in Equation (9).

$$EET_{Dj} = \frac{T_D \text{ size}}{VM_j \text{ computation power}} \quad (9)$$



Equation (10) computes the total time taken by VM<sub>*j*</sub> to execute all of the assigned tasks denoted by CT<sub>*j*</sub>.

$$CT_j = \sum_{D=1}^{S_j} (ETC_{Dj}) \quad (10)$$

where  $S_j$  is the number of scheduled (assigned) tasks to VM<sub>*j*</sub>.

Makespan is the maximum completion time among all the VMs and is computed as shown in Equation (11). Minimized makespan shows better performance in terms of early execution of the workload.

$$\text{Makespan} = \max(CT_1, CT_2, CT_3, \dots, CT_m) \quad (11)$$

where  $m$  is the number of VMs.

Throughput is another important evaluation metric in Cloud computing. Throughput is the ratio between the total number of tasks executed per unit time [11]. In our case, throughput of the whole Cloud datacenter is defined as the ratio between total number of tasks executed on a datacenter and makespan (shown in Equation (12)). Higher throughput represents better performance i.e., executing more tasks in a unit time.

$$\text{Throughput} = \frac{D}{\text{Makespan}} \quad (12)$$

where  $D$  represents the total number of tasks executed in the Cloud data center.

The objective function of the proposed approach is based on the maximization problem and is computed using Equation (13).

$$\text{ObjectiveFunction} = \max(\text{throughput} + (1/\text{makespan})) \quad (13)$$

where makespan [11] represents the maximum execution time of the data center i.e., the execution time of VM that completes execution of the assigned task at the last of all other VMs and throughput in our case is the ratio between the total number of tasks executed by the Cloud and makespan. Higher throughput and reduced makespan give maximum value for our objective function.

### 3.3. Inertia Weight Strategies for PSO Model

The optimization procedure of all the swarm intelligence-based meta-heuristics consists of two major phases. These phases include local and global searches. The balance of global and local search has a key role in finding optimal solutions. For the ideal situation, at the start of the search procedure, the espousal of global search space should be more than a local search space [47]. It allows population-based meta-heuristics to explore more search space at the beginning and then finding the global optimal position with more care. Inertia weight is the strongest control factor to maintain local and global search in a balanced way [48].

The literature study show that various researchers have worked on the inertia weight selection strategy to balance the local and global search of particles. However, there is still need to improve this balance. Researchers have proposed several inertia weight strategies, however, some of these inertia weights are popular among the research community. These inertia weights strategies includes Simple Random Inertia Weight (SRIW) [50], Chaotic Inertia Weight (CIW) [47,51], Chaotic Random Inertia Weight (CRIW) [51], Linear Descending Inertia Weight (LDIW) [47,52], and Adaptive Inertia Weight (AIW) [53].

SRIW inertia weight was proposed by [50] represented by the formula (shown in Equation (14)).

$$W_{\text{SRIW}} = 0.5 + (0.5 * \text{rand}()) \quad (14)$$

where  $\text{rand}()$  represents random values between 0 and 1.

The formula (shown in Equation (15)) shows CRIW that is proposed in [51]

$$W_{\text{CRIW}} = (0.5 * z) + (0.5 * \text{rand}()) \quad (15)$$

where  $z$  represents any value between 0 and 1,  $\text{rand}()$  represents random value between 0 and 1.

Equation (16) depicts the CDIW strategy that has proposed by [47,51,51]

$$W_{\text{CDIW}} = (w_1 - w_2) * \left( \frac{\text{MAXitr} - \text{Itr}}{\text{MAXitr}} \right) + (w_2 * z) \quad (16)$$

where  $w_2$  and  $w_1$  represents initial and final inertia weight and  $z$  is a random value between 0 and 1.  $\text{MAXitr}$  represents maximum iterations and  $\text{Itr}$  shows current iteration.

LDIW has been proposed by [47,52] which is represented by formula (depicted by Equation (17))

$$W_{\text{LDIW}} = (w_1 - w_2) * \left( \frac{\text{MAXitr} - \text{Itr}}{\text{MAXitr}} \right) + (w_2) \quad (17)$$

where  $w_1$  and  $w_2$  are the initial and final values of inertia weight.

Author has proposed Adaptive Inertia Weight (AIW) strategies in [53] (shown in Equation (18)). AIW approach adjusts the weight value after each iteration using feedback received from the previous iteration. The feedback provides the success rate (i.e.,  $P_s$ ) of particles in their previous iterations. The success rate of the particle shows that how many times a particular particle improves local best values as compared to the previous one and the inertia weight is computed in Equation (18).

$$W_{\text{AIW}} = ((w_1 - w_2) * P_s) + w_2 \quad (18)$$

where  $W_{\text{AIW}}$  is the inertia weight calculated using AIW strategy,  $w_1$  shows initial value,  $w_2$  denotes the final value and  $P_s$  is the particles success rate from previous iterations and is computed using Equation (20).

### 3.4. Proposed Inertia Weight Strategy

The performance of the PSO algorithm depends on optimal parameters setting and inertia weight. Searching for an optimal solution within the search region, the inertia weights bring a balance between exploration and exploitation [24]. In this paper, a novel inertia weight strategy “Linearly Decreasing Adaptive Inertia Weight (LDAIW)” has been proposed (shown in Equation (19)).

$$W_{\text{PA}} = \left( \frac{(w_1 - w_2)}{P_s} + \left( \frac{(\text{MAXitr} - \text{Itr})}{\text{MAXitr}} \right) * \left( \frac{(w_1 - (w_1 - w_2))}{P_s} \right) \right) \quad (19)$$

The proposed approach, exploits strengths of both LDIW [47,52] and AIW [53] strategies.

In LDIW, the weight value is set to the maximum in the start based on the assumption that global search is favored in the initial stage to explore more search space. The weight value of LDIW is decreasing gradually to decrease the search space gradually in local search at the end which leads to better performance than other state-of-the-art inertia weight strategies [48]. This is a widely used inertia weight strategy due to its simplicity and fast convergence; however, the state of the environment is not checked for adopting the inertia weight.

The AIW technique monitors the search space and adjusts weight value based on the feedback from one or more parameters. This method uses the success rate of the particles as a feedback parameter and adjusts the inertia weight strategy according to the state of the environment. It has been observed that AIW works well for smaller datasets as compared to larger one.

Based on these observations, a novel inertia weight strategy has been proposed that combines the strength of LDIW and AIW inertia weight strategies. The proposed strategy LDIAIW has characteristic of linear decreasing behavior from LDIW (not in AIW) and adjust weight values based on feedback behavior from AIW (which not exist in LDIW). Being linearly decreasing and adaptive, LDIAIW out perform in terms of Makespan, throughput, and ARUR than other state-of-art approaches.

In Equation (19), the value of  $P_s$  is calculated using Equation (20). The value of  $P_s$  can also approach to zero when there is no improvement in particle position (Pos) as given in Algorithm 1 line 3. If the value of  $P_s$  becomes zero then it is assigned a default value of 1 as given in Algorithm 1 line no 35–37. In Equation (19),  $w_1$  is the maximum value and  $w_2$  is the minimum value. MAXitr represents the total number of iterations which is set to 200 in this research after fine-tuning, Itr shows the current iteration, and  $P_s$  represents article success rate which is used as feedback for adjusting inertia weight as shown in Equation (20).

$$P_s = \left( \frac{\sum_{i=1}^n (SS_i)}{N} \right) \quad (20)$$

Particle success rate ( $P_s$ ) is computed using Equation (20), where N represents the total number of particles, and  $SS_i$  is the success status of particles and is defined using Equation (21), and  $n$  represent index of particle i.e.,  $i = 1$  to  $n$ . The proposed weight strategy provides a better balance between global and local searches.

$$SS_i = \begin{cases} 1 & pBest_i > pBest_{i-1} \\ 0 & \text{otherwise} \end{cases} \quad (21)$$

### 3.5. Proposed Scheduler

This section presents and discuss the proposed scheduling algorithm. A set of tasks with their computation requirements in Million Instructions (MI) and a set of VMs with their computation power in Million Instructions Per Second (MIPS) has been used as the input parameters. The output of the proposed approach includes the tasks to VM mapping.

At Line 1–2 (Algorithm 1), VM list (*vmList*) and task list (*taskList*) is obtained. The total number of tasks (*taskCount*) and total number VMs (*vmCount*) are computed (Line 3–4, Algorithm 1). Line 5–10 (Algorithm 1) presents the necessary initialization of the proposed approach. The initialization step of the particles has been performed and their results have been stored in *pbMap* (Line 11, Algorithm 1). The while loop (Line 12–39, Algorithm 1) executes according to a fixed number of times i.e., *MaxItr* which is 200 in our case. The Inertia weight has been computed (Line 13, Algorithm 1) based on Equations (19)–(21). The for loop (Line 14–37, Algorithm 1) repeats according to the number of particles which is 20 in this work. Each particle represents a complete mapping of tasks to VMs. The nested for loop (Line 15–28, Algorithm 1) iterates for each task to be mapped to the VM, where  $r1$  and  $r2$  are two random numbers between 0 and 1 (Line 17, Algorithm 1). The  $c_1$  and  $c_2$  are constant acceleration factors whose values are initialized at 2 and 1.49455 respectively [48].

**Algorithm 1:** Proposed PSO scheduler.

---

```

Input : taskMap: Set of tasks with their length in MI and
         vmMap: Set of VMs with their processing capacity(MIPS)
Output: gbFMap: global best based final mapping of tasks to VMs
vmList = getVmList(vmMap)
taskList = gettaskList(taskMap)
taskCount = taskList.size()
vmCount= vmlist.size()
pos = 0, v = Randnbr(0, 1), w = 0.0, SS = 0
noParticles = 20, itr = 1, MaxItr = 200
c1 = 2, c2 = 1.49455, w1 = 0.9, w2 = 0.4
pbMap<Integer, Double> = Null
vRTMap <Integer, Double> = Null
pMap<task, Vm> = Null
prtsMap<Integer, Map<task, Vm>> = Null
pbMap = initializeParticles(taskCount, vmCount, pbMap, pMap, gbFMap,
noParticles)
while (itr <= MaxItr) do
    w = ((w1 - w2)/Ps) + ((MAXitr - Itr)/MAXitr)*(w1 - (w1 - w2))/Ps
    for (p = 1 to noParticles) do
        for (c = 1 to taskCount) do
            r1 = Randnbr(0, 1), r2 = Randnbr(0, 1)
            v = (w * v) + (c1 * r1 * (pbMap.get(p) - pos)) + (c2 * r2 * (gbValue - pos))
            pos = pos + V
            if pos >= taskCount || pos < 0 then
                | pos = RandPos(0, taskCount-1)
            end
            Vm vm = getVm(pMap);
            execTime = getCltExecTime(taskList, c, vm)
            execTime += getVmReadyTime(vRTMap, vm, c)
            updateVRTMap(vRTMap, vm, c, execTime)
            updatepMap(vRTMap, pMap, pos, c);
        end
        pBestValue = getpbMap(vRTMap)
        if (pBestValue > pbMap.get(p)) then
            | pbMap.put(p, pBestValue)
            | SS++
            if pBestValue > gbValue then
                | gbValue = pBestValue
                | gbFMap.put(0, pMap)
            end
        end
    end
    Ps = SS/noParticles
    if (Ps ≤ 0) then
        | Ps = 1
    end
    itr++
end

```

---

The *velocity* and *position* are updated (Line 18–19, Algorithm 1) using Equation (4) and (5) respectively. The *if* condition (Line 20–22, Algorithm 1) restricts the particle positions within the lower and upper bound of the search space. In case the particle position value exceeds the total count of tasks or less than zero (Line 20, Algorithm 1), a new random

position is assigned to the particles (Line 21, Algorithm 1). The VM is identified in Line 23 of Algorithm 1 and the execution time of the task is computed (Line 24, Algorithm 1) using Equation (9). The completion time of the task is computed by adding the execution time and ready time of the VM (that is stored in  $vRTMap$ ) to start task execution as represented in Equation (10) (Line 25, Algorithm 1). The VM ready time is updated in the  $vRTMap$  (Line 26, Algorithm 1) and particle is added to the map (Line 27, Algorithm 1). The Personal best value ( $pBestValue$ ) is computed (Line 29, Algorithm 1) and current  $pBestValue$  is compared with the previous  $pBestValue$  (Line 30, Algorithm 1).

In case the current best value is greater than the previous one then the previous value is replaced with new  $pBestValue$  and stored in  $pbMap$  (Line 31, Algorithm 1). At line 32 (Algorithm 1),  $pBestValue$  is compared with the global best value ( $gBValue$ ). When the *if* condition (Line 32, Algorithm 1) becomes true then  $gBValue$  is updated and mapping based on global best is updated in the global best based final mapping ( $gbFMap$ ) (Line 33–34, Algorithm 1). This process continues until all tasks of single-particle are mapped to the VMs. In each iteration, the  $Itr$  is incremented by 1 (Line 38, Algorithm 1) which is used as a condition in the while loop at (Line 12, Algorithm 1).

#### 4. Experimental Setup and Simulation Results

To evaluate and compare the proposed approach, we have implemented PSO algorithm with five different inertia weights proposed in [50–53]. These inertia weights strategies includes SRIW [50], CRIW [51], CIW [51], LDIW [52], and AIW [53]. The experiments simulate the optimized solution using makespan, ARUR, and throughput. The results of the proposed approach have been validated by comparing results generated using PSO with the other five prominent inertia weight strategies.

##### 4.1. Dataset Analysis

Experiments have been performed using four different instances of HCSP benchmark dataset [11,54–56]. Based on our supposition that in real Cloud environment  $D \gg M$ , each of the HCSP instances used has 8132 heterogeneous tasks and 16 heterogeneous VMs. Heterogeneous means that VMs in the VMs set are heterogeneous in terms of their processing capability (in MIPS) and tasks in the task set are heterogeneous in terms of tasks computation requirement (in MI). These instances include *i\_hilo*, *c\_hilo*, *i\_lohi*, and *c\_lohi* that represent different heterogeneity and consistency level in terms of VM computation capability (in MIPS) and tasks computation requirement (in Million Instructions(MI)) [54,55]. By consistent means, the variations in VMs MIPS and tasks MIs are uniform. However, inconsistent behavior shows that the variations in VM MIPS and tasks MI are not uniform. The *c* and *i* are used for the consistent and inconsistent nature of tasks and VMs respectively. Similarly, *lo* represents low and *hi* represents a high level of heterogeneity. The *lohi* shows a low level of heterogeneity for VMs and a high level of heterogeneity of tasks. High heterogeneity means that there are more variations in the sizes of tasks (i.e., there is a big difference between the smallest and largest task). Similarly, low heterogeneity represents a smaller difference between the largest and smallest tasks. The sizes of tasks in different HCSP instances are different like i.e., the sizes of tasks in *i\_hilo* and *c\_hilo* are smaller than that of *i\_lohi* and *c\_lohi* instances of the HCSP dataset and is considered as a bit lighter datasets than others.

##### 4.2. Parameter Initialization

At the start of the search process, particles are initialized to some random positions as shown in the table Table 5. For the task to VM mapping, tasks are randomly assigned to VMs at this stage.

**Table 5.** Initialization parameters.

Parameters	Values
Total <sub>iterations</sub>	200
Total <sub>particles</sub>	20
Min <sub>position</sub>	0
Max <sub>position</sub>	No of VMs-1
w <sub>1</sub>	0.4 [47,48]
w <sub>2</sub>	0.9 [28,47,48]
Acceleration <sub>factor c<sub>1</sub></sub>	2 [48]
Acceleration <sub>factor c<sub>2</sub></sub>	1.49455 [48]
Stopping Criteria	MaxItr

The maximum number of iterations and particles is fixed (200 iterations and 20 particles) after comprehensive fine-tuning by setting different values for maximum iterations and particle size. The lower bound and upper bound are set to zero and no of VMs-1 (VMCount-1) respectively. The values of  $w_1$  and  $w_2$  are set to 0.4 and 0.9, respectively [47,48]. The value of Acceleration factors  $c_1$  and  $c_2$  are initialized to 2 and 1.49455, respectively [48]. The stopping criteria are the maximum number of iterations (MaxItr), which equals 200.

#### 4.3. Simulation Environment

This section presents the computational environment used for simulation. To perform experiments and evaluate the proposed scheduling technique, a simulation environment was used. This is because, in the simulation environment, any number and types of resources with various heterogeneity levels can be used for performing experiments. Moreover, experiments can be performed as many times as needed without any restriction of time and execution cost. The experimental setup for performing experiments consists of a PC equipped with a CPU (Intel core i5 T8500 3.0 GHz, Memory (20.00 GB)) HD 2 TB, implemented in Java-based Eclipse IDE 3.0 and Cloudsim 3.0.3 [11,57] simulation tool. Table 6 summarizes the experimental setup used for experimentation.

**Table 6.** Summary of simulation environment configuration.

Parameters	Values
Simulator	Cloudsim version 3.0.3
processor	Intel cor i5-8500 3.00 GHz
RAM	20 GB
Hard drive	2 TB
Total host machines	10
Host machines Power	15,000 MBs each
VMs	16
Total tasks	8132

#### 4.4. Experimental Execution and Results

Being stochastic optimization approaches, meta-heuristic-based algorithms need to be run multiple times to achieve meaningful and more realistic statistical evaluation [47]. In this research, every approach is executed 5 times for each instance of the HCSP dataset, and the average of these runs is computed and presented for the comparison.

Makespan is the key performance evaluation parameter and ultimate demand of the cloud user. The results concerning the makespan for the AdPSO and available state-of-the-art approaches are plotted in Figure 1 for *i\_hilo*, *c\_hilo*, *c\_lohi*, and *i\_lohi* instances of the HCSP benchmark dataset. These results shows that SRIW, CRIW approaches has shown poor makespan performance for all the four instances of HCSP dataset. The CDIW, and LDIW approaches have shown improved behavior for *c\_hilo* and *i\_hilo* instances and slightly poor performance for *c\_lohi* and *i\_lohi* instances of HCSP dataset. The AIW approach has better performed for *c\_hilo* and *i\_hilo* instances as compared to *c\_lohi* and *i\_lohi* instances of HCSP dataset. This is because, the sizes of tasks in the *c\_hilo* and *i\_hilo* is smaller as compared to the tasks sizes in *c\_lohi* and *i\_lohi* HCSP instances. Moreover, LDIW strategy has shown consistent performance for all instance of the HCSP dataset due to its linearly decreasing mode. The AdPSO technique is capable to lower the makespan for all instances of HCSP dataset and improved by 1–7 % on *i\_hilo*, 2–11% on *c\_hilo*, 1–5% on *i\_lohi*, and 1–4% on *c\_lohi* instance of HCSP benchmark dataset as compared to AIW, SRIW, CRIW, CDIW, and LDIW state-of-the-art inertia weight strategies. This shows that the proposed approach maintain better balance between local and global search process.

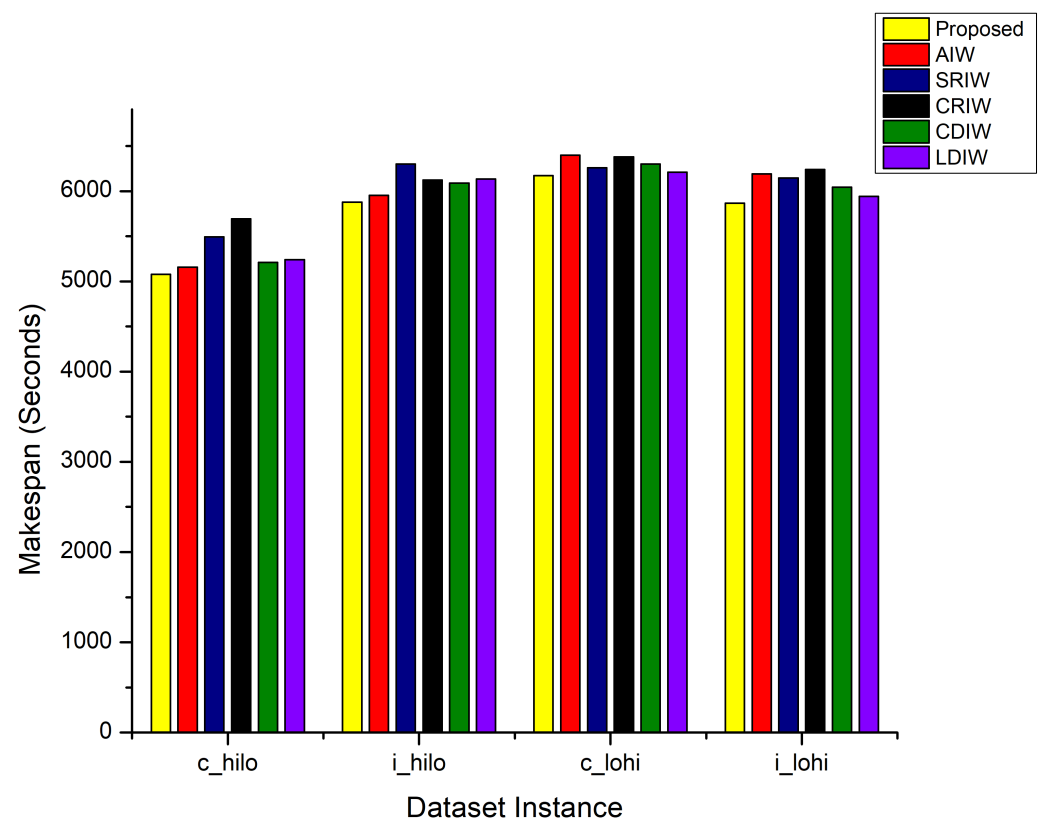
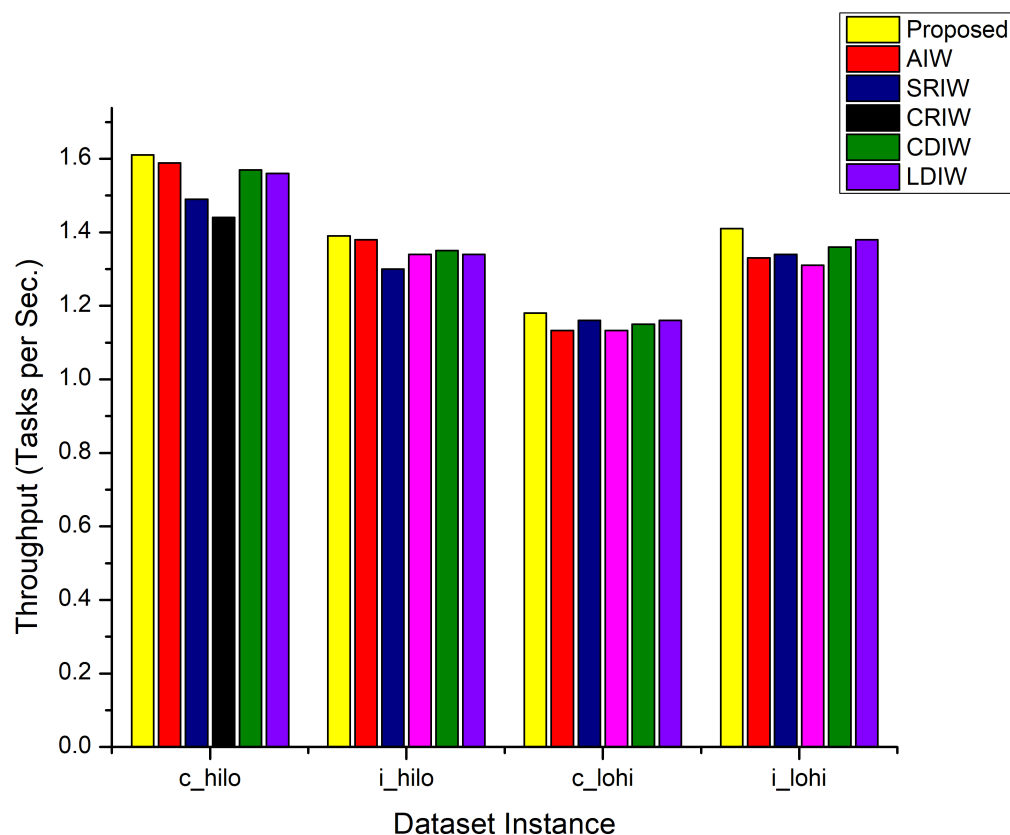


Figure 1. Makespan Comparison.

Another key parameter to compare the performance of the AdPSO is throughput. Figure 2 shows the throughput results for the AdPSO technique and compared contemporary approaches considering the *c\_hilo*, *i\_hilo*, *c\_lohi*, and *i\_lohi* instance of HCSP dataset. Likewise the makespan results, similar behavior is observed for the throughput results for all of the compared approaches. Figure 2 reveals that the proposed scheduler has achieved higher throughput up to 1–7% for the execution of *c\_hilo*, 2–12% for *i\_hilo*, 1–6% for *c\_lohi*,

and 1–4% for the execution of *i\_lohi* instance of HCSP dataset as compared to PSO with AIW, SRIW, CRIW, CDIW, and LDIW inertia weight strategies.

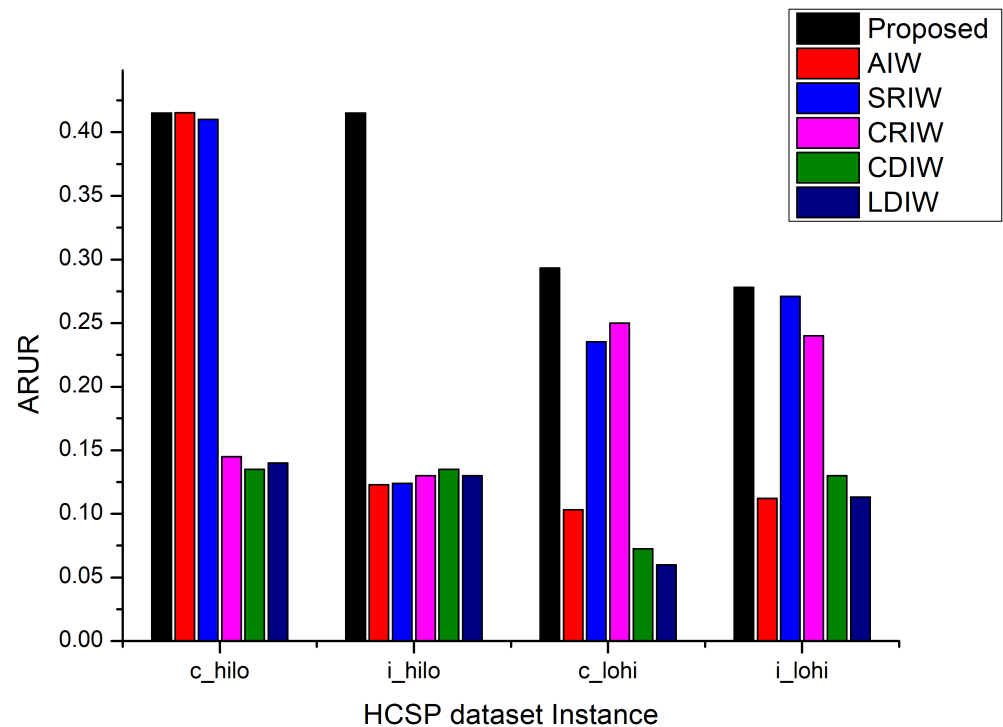


**Figure 2.** Throughput Comparison.

To perform evaluation of AdPSO, the ARUR results are obtained and compared against the AIW, SRIW, CRIW, CDIW, and LDIW approaches as shown in Figure 3. For the *c\_hilo* instance, the AIW, SRIW and proposed approach have shown almost similar ARUR performance. The rest of the three approaches resulted in 60% less ARUR as compared to the AIW, SRIW, and proposed approaches. For the *c\_lohi* and *i\_lohi* datasets, the proposed approach dominated the rest of the contemporary approaches followed by SRIW and CRIW approaches. The AIW approach has shown poor ARUR performance for the *i\_lohi* dataset instance whereas the LDIW approach has shown the lowest ARUR performance for the *c\_lohi* dataset. However, for the *i\_hilo* dataset, the proposed approach dominated all the approaches by almost 2 times against the other approaches.

The comparison results reveal that the proposed approach outperforms AIW, SRIW, CRIW, CDIW, and LDIW approaches for all four instances (*i\_hilo*, *c\_hilo*, *i\_lohi*, and *c\_lohi*) of the HCSP benchmark dataset. The reason is that the inertia weight strategy of the proposed approach is more effective in keeping a balance between local and global search.





**Figure 3.** ARUR Comparison.

The evaluation is then extended and proposed PSO approach is compared against PSO-Boost, APSO, PSO, and Hyper-heuristic approaches using same HCSP benchmark datasets. The best and worst case results concerning the makespan for the compared approaches are shown in Figure 4. For the *c\_hilo* dataset, the AdPSO has better performed the other three approaches by an improvement of 14–27%. The PSO-BOOST and Hyper-heuristic approaches have shown similar behavior for the *i\_hilo* dataset instance while negligible degradation is observed for the *c\_lohi* and *i\_lohi* instances. The PSO approach has shown poor makespan performance whereas the APSO approach shown steady behavior. All these results advocates the effectiveness of the proposed approach.

The results shown in Figure 5 show the throughput achieved for all the compared approaches. The empirical results (depicted in Figure 5) reveal that the proposed scheduling algorithm has improved the throughput by 6–22.32% against the compared approaches. Again the PSO-BOOST and Hyper-heuristic approaches attained second best throughput results and dominated the other compared approaches.

Another important metric to evaluate the efficiency of the task scheduling approaches is the Average Resource Utilization Ratio (ARUR). To evaluate the AdPSO technique against their counterparts, the experiments are performed using the same HCSP benchmark dataset instances. The experimental results shown in Figure 6 show that the proposed technique has gained 3.23%, 8.63%, 10.66% and 25% higher ARUR than PSO-Boost, APSO, Hyper-heuristic and PSO respectively for the *c\_hilo* instance of the HCSP dataset. When the *i\_hilo* dataset is used for the performance evaluation, almost same behavior is revealed for the approaches similar to that for *c\_hilo* instance. similarly, the proposed approach attained 7% and 25% improved ARUR than APSO and PSO using *c\_lohi* instance and 2.6%, 5.2%, 3.34% and 18.3% higher ARUR as compared to PSO-Boost, APSO, Hyper-heuristic and PSO for the *i\_lohi* instance of HCSP benchmark dataset.

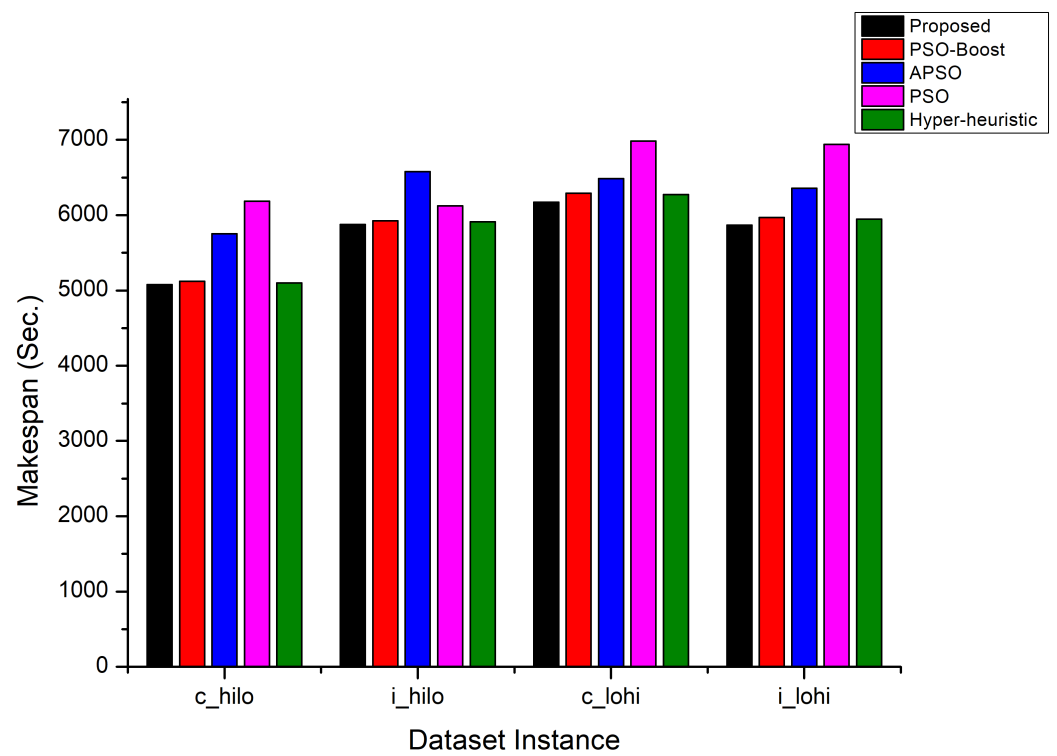


Figure 4. Makespan Comparison on HCSP dataset instances.

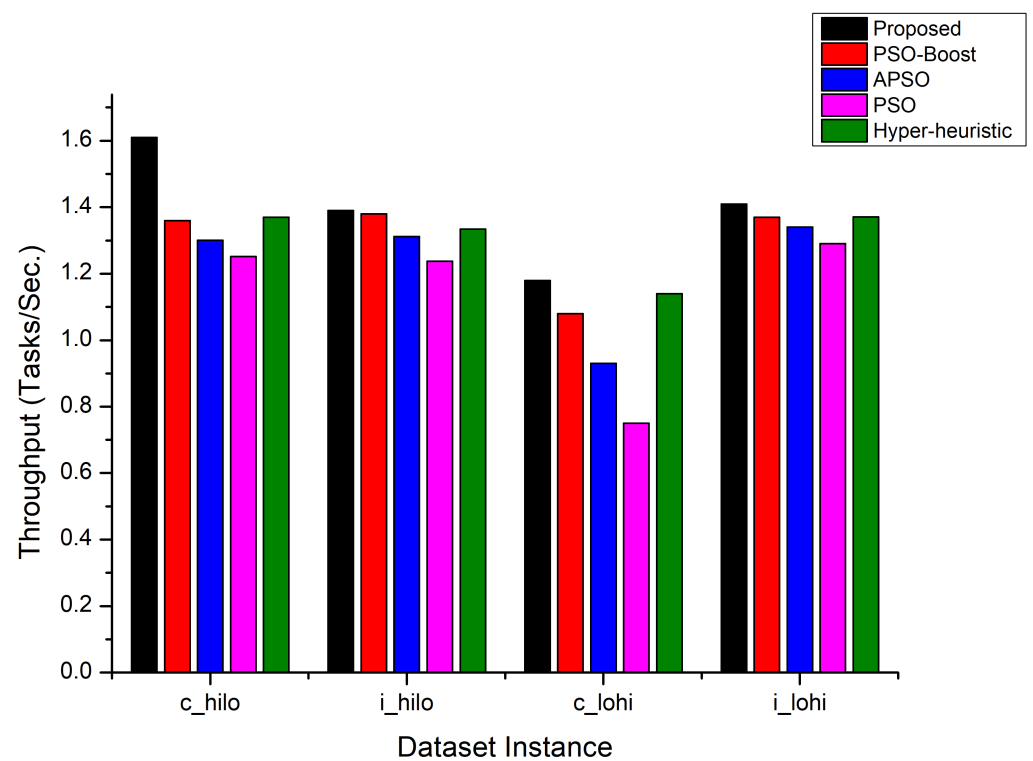
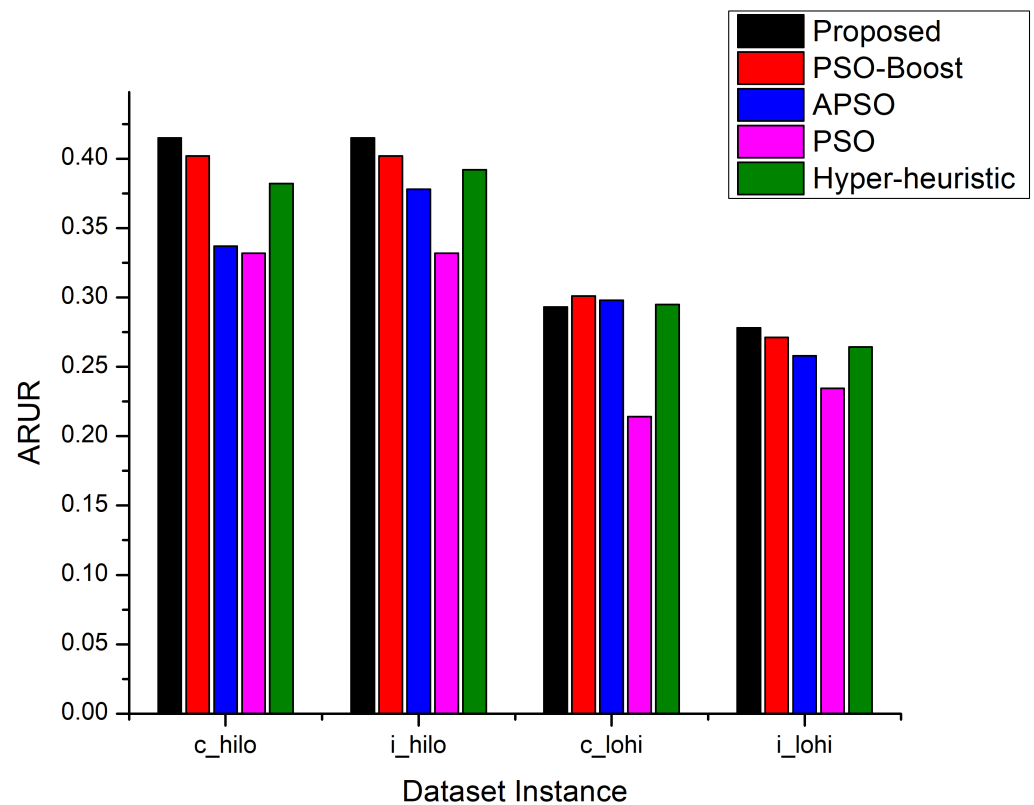


Figure 5. Throughput Comparison on HCSP dataset instances.



**Figure 6.** ARUR Comparison on HCSP dataset instances.

Experimental results show that AIW has acquired minimum overall execution time (makespan) and higher throughput as compared to SRIW, CRIW, CDIW, and LDIW for the execution of *i\_hilo* and *c\_hilo* instances of the HCSP dataset. This reveals that the performance of AIW is better in terms of makespan and throughput for comparatively smaller datasets than the larger one. This is because the size of *i\_hilo* and *c\_hilo* instances based workload are smaller in terms of total MI as compared to workload of *i\_lohi* and *c\_lohi* instances of HCSP dataset.

Experimental results show that the proposed approach has achieved 5.20 and 3.53% reduced makespan than AIW for *i\_lohi* and *c\_lohi* instances of the HCSP dataset respectively. These results also reveal that the proposed scheduling scheme has gained 5.35 and 4.09% higher throughput as compared to AIW for the execution of *i\_lohi* and *c\_lohi* instances of the HCSP dataset respectively. For the execution of *i\_hilo* and *c\_hilo* dataset, the proposed approach attained 1.58 and 1.25% reduced makespan and 1.59, 1.28% improved throughput as compared to AIW.

Our experimental results exhibit that LDIW results in reduced makespan and improved throughput as compared to AIW, SRIW, CRIW, and CDIW for the execution of *i\_lohi* and *c\_lohi* instances of HCSP benchmark datasets respectively. This shows that LDIW performs better in terms of makespan and throughput for larger datasets than smaller size datasets.

Experimental results show that the proposed scheduling scheme has achieved 4.19 and 3.15% reduced makespan than LDIW for the execution of *i\_hilo* and *c\_hilo* benchmark datasets respectively. These results also exhibit that the proposed scheduling approach has gained 4.34 and 3.28% improved throughput as compared to LDIW for *i\_hilo* and *c\_hilo* datasets respectively. Our experimental results show that the proposed approach achieved 1.27% minimized makespan than LDIW for the *i\_lohi* dataset and 0.62% better makespan for the *c\_lohi* dataset. These results show that the proposed approach has gained 1.38% higher throughput as compared to LDIW for *i\_lohi* and 1.25% for *c\_lohi* dataset.

The experimental results reveal that the proposed scheduling algorithm showed up to 26.84% lower makespan, 22.32% higher throughput, and 30% improved ARUR than PSO for the c\_hilo, c\_lohi, and i\_lohi instances, respectively. Moreover, the proposed scheduling approach has achieved up to 20.12, 11.69, and 8.64% higher throughput as compared to the APSO algorithm. Similarly, the proposed scheduling algorithm has gained up to 14.97% lower makespan, 5.5% higher throughput, and 3.23% higher ARUR than the PSO-Boost scheduling algorithm. This is because of the proposed approach use of a novel inertia weight strategy to keep a balance between the local and global search. The proposed approach also exhibits stable performance for heterogeneous dataset. This is because the proposed approach inherent strengths of both AIW (better performance for smaller dataset) and LDIW (better performance for larger dataset) inertia weight strategies. The overall results reveal that the proposed approach outperforms concerning the makespan, throughput, and ARUR and is more stable and scalable than existing approaches. This research mainly focuses on the selection of inertia weight strategy to balance the global and local search using the makespan and throughput.

## 5. Conclusions and Future Work

PSO task scheduling approach is considered a more suitable approach for a load balanced scheduling of tasks due to its fast convergence and easy to implement nature. However, the PSO approach suffers from a pre-mature convergence issue. The inertia weight is a key attribute to keep a balance between global and local search space. In this paper, five inertia weight strategies have been investigated comprehensively using a PSO-based scheduler. This work contributed an adaptive inertia weight approach for Cloud-based task scheduling. The performance of the proposed approach has been evaluated and compared against five renowned PSO based inertia weight strategies concerning makespan, throughput and ARUR. The results evaluation reveal that the proposed approach attained up to 10%, 12%, and 30% improvement concerning throughput and ARUR respectively against the compare approaches.

Most IoT applications [58,59] require real-time responses for accurate decision-making. As a future task it is intended to optimize the response time and employ scheduling to provide real-time or near to real-time response for delay-sensitive applications.

**Author Contributions:** Formal analysis, S.N.; Funding acquisition, H.H.; Methodology, S.N.; Supervision, M.A.; Visualization, M.I. and H.H.; Writing—original draft, S.N.; Writing—review & editing, M.A. and M.I. All authors have read and agreed to the published version of the manuscript.

**Funding:** The authors also thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the New Brunswick Innovation Foundation (NBIF) for the financial support of the global project. These granting agencies did not contribute to the design of the study and collection, analysis, and interpretation of data.

**Institutional Review Board Statement:** Exclude as not required.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not required as dataset is provided in the manuscript.

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

## References

1. Wang, X.; Yeo, C.S.; Buyya, R.; Su, J. Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. *Future Gener. Comput. Syst.* **2011**, *27*, 1124–1134. [[CrossRef](#)]
2. Nabi, S.; Khan, M.N.A. An Analysis of Application Level Security in Service Oriented Architecture. *Int. J. Mod. Educ. Comput. Sci.* **2014**, *6*, 27. [[CrossRef](#)]
3. Ibrahim, M.; Imran, M.; Jamil, F.; Lee, Y.J.; Kim, D.H. EAMA: Efficient adaptive migration algorithm for cloud data centers (CDCs). *Symmetry* **2021**, *13*, 690. [[CrossRef](#)]

4. Ibrahim, M.; Nabi, S.; Hussain, R.; Raza, M.S.; Imran, M.; Kazmi, S.A.; Hussain, F. A comparative analysis of task scheduling approaches in cloud computing. In Proceedings of the 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melbourne, Australia, 11–14 May 2020; pp. 681–684.
5. Ibrahim, M.; Iqbal, M.A.; Aleem, M.; Islam, M.A. SIM-cumulus: An academic cloud for the provisioning of network-simulation-as-a-service (NSaaS). *IEEE Access* **2018**, *6*, 27313–27323. [[CrossRef](#)]
6. Ibrahim, M.; Iqbal, M.A.; Aleem, M.; Islam, M.A.; Vo, N.S. MAHA: Migration-based adaptive heuristic algorithm for large-scale network simulations. *Clust. Comput.* **2020**, *23*, 1251–1266. [[CrossRef](#)]
7. Ibrahim, M.; Nabi, S.; Baz, A.; Naveed, N.; Alhakami, H. Towards a task and resource aware task scheduling in cloud computing: An experimental comparative evaluation. *Int. J. Networked Distrib. Comput.* **2020**, *8*, 131–138. [[CrossRef](#)]
8. Singh, J.; Singh, H. Efficient Tasks scheduling for heterogeneous multiprocessor using Genetic algorithm with Node duplication. *Indian J. Comput. Sci. Eng.* **2011**, *2*, 402–410
9. Pooranian, Z.; Shojafar, M.; Tavoli, R.; Singhal, M.; Abraham, A. Hybrid metaheuristic algorithm for job scheduling on computational grids. *Informatika* **2013**, *37*, 157–164.
10. Ibrahim, M.; Jamil, F.; Lee, Y.; Kim, D. Blockchain Based Secured Load Balanced Task Scheduling Approach for Fitness Service. *CMC-Comput. Mater. Contin.* **2022**, *71*, 2599–2616. [[CrossRef](#)]
11. Ibrahim, M.; Nabi, S.; Baz, A.; Alhakami, H.; Raza, M.S.; Hussain, A.; Djemame, K. An in-depth empirical investigation of state-of-the-art scheduling approaches for cloud computing. *IEEE Access* **2020**, *8*, 128282–128294. [[CrossRef](#)]
12. Logesh, R.; Subramaniaswamy, V.; Malathi, D.; Sivaramkrishnan, N.; Vijayakumar, V. Enhancing recommendation stability of collaborative filtering recommender system through bio-inspired clustering ensemble method. *Neural Comput. Appl.* **2020**, *32*, 2141–2164. [[CrossRef](#)]
13. Logesh, R.; Subramaniaswamy, V.; Vijayakumar, V.; Gao, X.Z.; Indragandhi, V. A hybrid quantum-induced swarm intelligence clustering for the urban trip recommendation in smart city. *Future Gener. Comput. Syst.* **2018**, *83*, 653–673. [[CrossRef](#)]
14. Mubeen, A.; Ibrahim, M.; Bibi, N.; Baz, M.; Hamam, H.; Cheikhrouhou, O. Alts: An Adaptive Load Balanced Task Scheduling Approach for Cloud Computing. *Processes* **2021**, *9*, 1514. [[CrossRef](#)]
15. Hamad, S.A.; Omara, F.A. Genetic-based task scheduling algorithm in cloud computing environment. *Int. J. Adv. Comput. Sci. Appl.* **2016**, *7*, 550–556.
16. China, K.Y.; China, Q.Y. A task scheduling based on simulated annealing algorithm in cloud computing. *Int. J. Hybrid Inf. Technol.* **2016**, *9*, 403–412.
17. Pacini, E.; Mateos, C.; Garino, C.G. Distributed job scheduling based on Swarm Intelligence: A survey. *Comput. Electr. Eng.* **2014**, *40*, 252–269. [[CrossRef](#)]
18. Beni, G.; Wang, J. Swarm intelligence in cellular robotic systems. In *Robots and Biological Systems: Towards a New Bionics*; Springer: Berlin/Heidelberg, Germany, 1993; Volume 102, pp. 703–712.
19. Ertenlice, O.; Kalayci, C.B. A survey of swarm intelligence for portfolio optimization: Algorithms and applications. *Swarm Evol. Comput.* **2018**, *39*, 36–52. [[CrossRef](#)]
20. Bonabeau, E.; Dorigo, M.; Theraulaz, G. *Swarm Intelligence: From Natural to Artificial Systems*, 1st ed.; Oxford University Press: New York, NY, USA, 1999.
21. Kaur, P.; Sharma, M. Diagnosis of human psychological disorders using supervised learning and nature-inspired computing techniques: A meta-analysis. *J. Med. Syst.* **2019**, *43*, 1–30. [[CrossRef](#)]
22. Hu, Y.; Liu, K.; Zhang, X.; Su, L.; Ngai, E.W.T.; Liu, M. Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review. *Appl. Soft Comput.* **2015**, *36*, 534–551. [[CrossRef](#)]
23. Sharma, M.; Kaur, P.A. Comprehensive Analysis of Nature-Inspired Meta-Heuristic Techniques for Feature Selection Problem. *Arch. Comput. Methods Eng.* **2021**, *28*, 1103–1127. [[CrossRef](#)]
24. Agrawal, A.; Tripathi, S. Particle swarm optimization with adaptive inertia weight based on cumulative binomial probability. *Evol. Intell.* **2021**, *14*, 305–313. [[CrossRef](#)]
25. Ghamisi, P.; Benediktsson, J.A. Feature selection based on hybridization of genetic algorithm and particle swarm optimization. *IEEE Geosci. Remote Sens. Lett.* **2015**, *12*, 309–313 [[CrossRef](#)]
26. Satapathy, S.C.; Chittineni, S.; Krishna, S.M.; Murthy, J.V.R.; Reddy, P.P. Kalman particle swarm optimized polynomials for data classification. *Appl. Math. Model* **2012**, *36*, 115–126 [[CrossRef](#)]
27. Nabi, S.; Ahmed, M. PSO-RDAL: Particle swarm optimization-based resource-and deadline-aware dynamic load balancer for deadline constrained cloud tasks. *J. Supercomput.* **2021**, 1–31. [[CrossRef](#)]
28. Kumar, M.; Sharma, S.C. PSO-based novel resource scheduling technique to improve QoS parameters in cloud computing. *Neural Comput. Appl.* **2019**, *32*, 12103–12126. [[CrossRef](#)]
29. Nabi, S.; Rehman, S.U.; Fong, S.; Aziz, K. A model for implementing security at application level in service oriented architecture. *J. Emerg. Technol. Web Intell.* **2014**, *6*, 157–163. [[CrossRef](#)]
30. Torabi, S.; Safi-Esfahani, F. A dynamic task scheduling framework based on chicken swarm and improved raven roosting optimization methods in cloud computing. *J. Supercomput.* **2018**, *74*, 2581–2626. [[CrossRef](#)]
31. Tawfeek, M.A.; El-Sisi, A.; Keshk, A.E.; Torkey, F.A. Cloud task scheduling based on ant colony optimization. In Proceedings of the 2013 8th International Conference on Computer Engineering & Systems (ICCES), Colombo, Sri Lanka, 26–28 April 2013; pp. 64–66.

32. Chu, S.C.; Tsai, P.W.; Pan, J.S. Cat swarm optimization. In *Pacific Rim International Conference on Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 854–858.
33. Brabazon, A.; Cui, W.; O'Neill, M. The raven roosting optimisation algorithm. *Soft Comput.* **2016**, *20*, 525–545. [[CrossRef](#)]
34. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95-International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948.
35. Xu, L.; Wang, K.; Ouyang, Z.; Qi, X. An improved binary PSO-based task scheduling algorithm in green cloud computing. In Proceedings of the 9th International Conference on Communications and Networking in China, Maoming, China, 14–16 August 2014; pp. 126–131.
36. Meng, X.; Liu, Y.; Gao, X.; Zhang, H. A new bio-inspired algorithm: chicken swarm optimization. In *International Conference in Swarm Intelligence*; Springer: Cham, Switzerland, 2014; pp. 86–94.
37. Yannibelli, V.; Pacini, E.; Monge, D.; Mateos, C.; Rodriguez, G. Endowing the MIA Cloud Autoscaler with Adaptive Evolutionary and Particle Swarm Multi-Objective Optimization Algorithms. In *Advances in Computational Intelligence*; Batyrshin, I., Gelbukh, A., Sidorov, G., Eds.; MICAI 2021; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2021; Volume 13067. [[CrossRef](#)]
38. Mousavi, S.; Mosavi, A.; Varkonyi-Koczy, A.R. A load balancing algorithm for resource allocation in cloud computing. In *International Conference on Global Research and Education*; Springer: Cham, Switzerland, 2017; pp. 289–296.
39. Meena, J.; Kumar, M.; Vardhan, M. Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint. *IEEE Access* **2016**, *4*, 5065–5082. [[CrossRef](#)]
40. Zhang, L.; Chen, Y.; Sun, R.; Jing, S.; Yang, B. A task scheduling algorithm based on PSO for grid computing. *Int. J. Comput. Intell. Res.* **2008**, *4*, 37–43. [[CrossRef](#)]
41. Alkayal, E.S.; Jennings, N.R.; Abulkhair, M.F. Survey of task scheduling in cloud computing based on particle swarm optimization. In Proceedings of the 2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, United Arab Emirates, 21–23 November 2017; pp. 1–6.
42. Valarmathi, R.; Sheela, T. Ranging and tuning based particle swarm optimization with bat algorithm for task scheduling in cloud computing. *Clust. Comput.* **2019**, *22*, 11975–11988. [[CrossRef](#)]
43. Beegom, A.A.; Rajasree, M.S. Integer-PSO: A discrete PSO algorithm for task scheduling in cloud computing systems. *Evol. Intell.* **2019**, *12*, 227–239. [[CrossRef](#)]
44. Gupta, A.; Bhaduria, H.S.; Singh, A. Load balancing based hyper heuristic algorithm for cloud task scheduling. *J. Ambient. Intell. Humaniz. Comput.* **2021**, *12*, 5845–5852. [[CrossRef](#)]
45. Ramezani, F.; Lu, J.; Hussain, F.K. Task-based, system, load, balancing, in, cloud, computing, using, particle, swarm optimization. *Int. J. Parallel Program.* **2014**, *42*, 739–754. [[CrossRef](#)]
46. Kumar, N.; Vidyarthi, D.P. A model for resource-constrained project scheduling using adaptive PSO. *Soft Comput.* **2016**, *20*, 1565–1580. [[CrossRef](#)]
47. Huang, X.; Li, C.; Chen, H.; An D. Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies. *Clust. Comput.* **2019**, *23*, 1137–1147. [[CrossRef](#)]
48. Khalili, A.; Babamir, S.M. Makespan improvement of PSO-based dynamic scheduling in cloud environment. In Proceedings of the 2015 23rd Iranian Conference on Electrical Engineering, Tehran, Iran, 10–14 May 2015; pp. 613–618.
49. Al-Maamari, A.; Omara, F.A. Task scheduling using PSO algorithm in cloud computing environments. *Int. J. Grid Distrib. Comput.* **2015**, *8*, 245–256. [[CrossRef](#)]
50. Wu, Z.; Ni, Z.; Gu, L.; Liu, X. A revised discrete particle swarm optimization for cloud workflow scheduling. In Proceedings of the 2010 International Conference on Computational Intelligence and Security, Nanning, China, 11–14 December 2010; pp. 184–188.
51. Feng, Y.; Yao, Y.M.; Wang, A.X. Comparing with chaotic inertia weights in particle swarm optimization. In Proceedings of the 2007 International Conference on Machine Learning and Cybernetics, Hong Kong, China, 19–22 August 2007; Volume 1, pp. 329–333.
52. Eberhart, R.C.; Shi, Y. Comparing inertia weights and constriction factors in particle swarm optimization. In Proceedings of the 2000 Congress on Evolutionary Computation, CEC00, Las Vegas, NV, USA, 10–12 July 2000; Volume 1, pp. 84–88.
53. Nickabadi, A.; Ebadzadeh, M.M.; Safabakhsh, R. A novel particle swarm optimization algorithm with adaptive inertia weight. *Appl. Soft Comput.* **2011**, *11*, 3658–3670. [[CrossRef](#)]
54. Nabi, S.; Ibrahim, M.; Jimenez, J.M. DRALBA: Dynamic and Resource Aware Load Balanced Scheduling Approach for Cloud Computing. *IEEE Access* **2021**, *9*, 61283–61297. [[CrossRef](#)]
55. Nabi, S.; Ahmed, M. OG-RADL: Overall performance-based resource-aware dynamic load-balancer for deadline constrained cloud tasks. *J. Supercomput.* **2021**, *77*, 7476–7508. [[CrossRef](#)]
56. Heterogeneous Computing Scheduling Problem (HCSP) Instances. Available online: [https://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/HCSP\\_inst.htm](https://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/HCSP_inst.htm) (accessed on 10 October 2021).
57. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.; 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](#)]
58. Ghaffar, Z.; Alshahrani, A.; Fayaz, M.; Alghamdi, A.M.; Gwak, J. A Topical Review on Machine Learning, Software Defined Networking, Internet of Things Applications: Research Limitations and Challenges. *Electronics* **2021**, *10*, 880.
59. Cheikhrouhou, O.; Mahmud, R.; Zouari, R.; Ibrahim, M.; Zaguia, A.; Gia, T.N. One-dimensional CNN approach for ECG arrhythmia analysis in fog-cloud environments. *IEEE Access* **2021**, *9*, 103513–103523. [[CrossRef](#)]