

Resource-and-Latency-Aware PSO — A novel algorithm for intelligent and contextual cloud native resource scheduling

Tomás Dias*, Karima Velasquez[†], Luís Ferreira*, Diogo Fevereiro*,
Luís Rosa*, Luís Cordeiro*

*OneSource, Coimbra, Portugal

Email: {tomas.dias, luis.ferreira, duarte.fevereiro, luis.rosa, luis.cordeiro}@onesource.pt

[†]University of Coimbra, CISUC/LASI, DEI, Portugal

Email: kcastro@dei.uc.pt

Abstract—Swarm Intelligence (SI) has emerged as a prominent approach in resource scheduling for cloud-native applications, offering adaptability and effectiveness in multi-objective optimisation. However, SI-based methods often introduce additional complexity and challenges such as premature convergence and excessive exploration, which can lead to suboptimal performance. To address these challenges, this paper presents a novel Resource-and-Latency-Aware Particle Swarm Optimisation (RALA-PSO) algorithm for intelligent and contextual cloud native resource scheduling. RALA-PSO expands upon Particle Swarm Optimisation (PSO), a classical SI algorithm, by incorporating a greedy initialisation for faster convergence, a circular search space to avoid boundary stagnation, a multi-criteria fitness function tailored to Cloud-Native characteristics, a heuristic-based scoring model to reward suitable solutions, and the integration of adaptive weights and learning factors for exploration–exploitation balance. The RALA-PSO implementation is available as open-source in [14]. RALA-PSO prioritise solutions based on efficient resource allocation and reduction of communication overhead via service co-location. Experimental evaluations using the Alibaba Cluster TraceV2018 dataset show that RALA-PSO reduces network transmission overhead cost by 1.8× to 2.36× compared to state-of-the-art algorithms MOPPSO-CMS and ACO-CMS, while maintaining competitive resource efficiency. Notably, RALA-PSO also achieves runtimes up to 212 times faster than ACO-CMS and 1.5 times faster than MOPPSO-CMS. These results demonstrate that RALA-PSO is a scalable, efficient, and context-aware scheduling solution for cloud-native applications, particularly effective in environments with strict network and resource constraints.

Index Terms—Swarm Intelligence, resource scheduling, cloud-native and edge

I. INTRODUCTION

Recent works in cloud-native scheduling have explored intelligent optimisation techniques, with Swarm Intelligence (SI) showing potential due to its flexibility in handling non-linear, discrete, and dynamic environments, conditions under which traditional algebraic methods often struggle. Unlike Machine Learning (ML), which relies on large datasets and training, SI is well-suited for cloud-native scheduling due to its population-based design and support for multi-objective optimisation without assumptions on the objective function. Nevertheless, SI-based methods often entail increased algorithmic complexity and may suffer from premature convergence and

excessive exploration. To address these limitations, we propose the Resource-and-Latency-Aware Particle Swarm Optimisation (RALA-PSO) algorithm that optimises resource availability and inter-component proximity based on infrastructure status. RALA-PSO enhances traditional Particle Swarm Optimisation (PSO) through five key innovations: a greedy initialisation for faster convergence, a circular search space to avoid boundary stagnation, a multi-criteria fitness function that considers Cloud-Native characteristics, a heuristic-based scoring model to reward suitable solutions, and the integration of adaptive weights and learning factors for a more balanced exploration-exploitation trade-off. This paper presents the design and experimental evaluation of RALA-PSO, comparing it against state-of-the-art approaches. The results demonstrate the benefits of the proposed approach and underscore its relevance to the cloud-native resource management field.

The paper is organised as follows. Section II presents a literature review of SI-based scheduling algorithms. Section III introduces our proposed solution. Section IV shows the experiments conducted and results obtained. Finally, Section V concludes the paper.

II. LITERATURE REVIEW

This section reviews SI approaches for resource scheduling. SI refers to a class of biologically inspired optimisation algorithms that utilise stochastic search processes and iterative cooperative heuristics to explore and exploit complex solution spaces collectively. Examples include PSO, based on avian flocking behaviour [3], and Ant Colony Optimisation (ACO), based on pheromone-driven foraging in ants [4]. Both have been used for application scheduling in edge and cloud environments, where the problem is typically formulated as a multi-objective optimisation task. For instance, Lin et al. [4] proposed the ACO-MCMS (Ant Colony Optimisation for Multi-objective Container-based Microservice Scheduling) algorithm with three key objectives: reducing network transmission overhead between microservices, balancing cluster load, and enhancing the reliability of cluster services. Nevertheless, ACO-MCMS exhibits a time complexity of $O(N_{\max} \times K \times$

$n^2 \times \sum_{i=1}^m \text{Scale}_i$), where $N_{\max}$ is the maximum number of iterations, K is the size of the ant colony, n is the number of nodes in the cluster, and $\sum_{i=1}^m \text{Scale}_i$ represents the total sum of microservice replicas. This level of complexity can be prohibitive for real-time or dynamic scheduling scenarios, as performance degrades quadratically with the growth in microservices, nodes, or request volume. To overcome the limitations of ACO-MCMS, X. Chen and S. Xiao [3] proposed a new approach, MOPPSO-CMS, which combines the fast convergence and parallelism of PSO with Pareto dominance. Although MOPPSO-CMS demonstrates improved performance over ACO-MCMS, its increased algorithmic complexity also poses constraints on overall efficiency. Hence, despite their potential, state-of-the-art SI algorithms continue to face challenges related to scalability, parameter sensitivity, and the balance between exploration and exploitation, which collectively hinder their efficiency in large-scale, dynamic environments [2], [5], [6].

Our approach extends the original PSO [7], considering the cloud-native context (cf. Section III). To contextualise our proposal, we begin by summarising essential PSO concepts. A particle represents an individual candidate solution within the swarm, with its position corresponding to a point in the search space. Each particle maintains two key values: the personal best $p_{kbest,i}$, denoting the best solution it has discovered so far, and the global best $g_{best,i}$, denoting the best solution identified by any particle in the entire swarm. The particle's position is updated by adding its velocity to its current position, as described in Eq. (1).

$$X_k(i+1) = X_k(i) + V_k(i+1) \quad (1)$$

Where $X_k(i)$ represents the current position of particle k at iteration i , $X_k(i+1)$ denotes the updated position at iteration $i+1$, and $V_k(i+1)$ corresponds to the velocity of particle k at iteration $i+1$.

Velocity governs how a particle moves through the search space to find optimal solutions. The velocity accounts for the particle's current position $X_k(i)$, its personal best position $p_{kbest,i}$ and the global best position $g_{best,i}$ with coefficients c_1 and c_2 controlling the influence of the personal and global bests (Eq. (2), from Eberhart et al. [8]).

$$V_k(i+1) = wV_k(i) + c_1r_1\Delta p_k(i) + c_2r_2\Delta g_k(i) \quad (2)$$

Where r_1 and r_2 are random numbers in $[0, 1]$, $\Delta p_k(i) = p_{b,i} - X_k(i)$ denote the displacement from personal best and $\Delta g_k(i) = g_{b,i} - X_k(i)$ the displacement from the global best.

The population comprises all particles, each representing a candidate solution that is iteratively updated through changes in velocity and position. The algorithm begins by randomly initialising these positions and velocities. During each iteration, velocities and positions are updated based on the personal and global best values. The fitness of each particle is then evaluated through an objective function, with personal and global bests updated whenever improved solutions are found.

This process continues until convergence or termination, with the final population representing the scheduling decision.

III. RESOURCE-AND-LATENCY-AWARE PSO (RALA-PSO)

Motivated by previously discussed limitations, we propose a novel algorithm: Resource-and-Latency-Aware Particle Swarm Optimisation (RALA-PSO) as an enhanced PSO variant designed for the constraints of dynamic, cloud-native environments. RALA-PSO maintains the simplicity and lightweight nature of traditional PSO, while introducing five novel enhancements (see Section III-B), enabling intelligent, context-aware scheduling. The following sections describe the design of RALA-PSO, covering the system model, particle initialisation, evolution and fitness function. The RALA-PSO implementation is available as open-source in [14].

A. System Model

We defined the optimal placement by two equally important criteria: (1) selecting clusters with more available resources (i.e., lower current usage), and (2) the proximity of interdependent components, by placing them close to each other. For simplicity, the system model assumes single-node clusters, and all replicas of a given component are co-located on the same node. Although this may introduce a single point of failure and reduce load balancing efficiency, it can be beneficial when replicas are used primarily for horizontal scalability rather than fault tolerance. The model takes as input a tuple (Eq. (3)), consisting of res_spec , the set of available clusters, and app_spec , the set of applications, where res_spec is itself another tuple representing resource attributes.

$$\langle res_spec \mid app_spec \rangle \quad (3)$$

$$res_spec \rightarrow \langle c_id \mid c_spec \rangle \quad (4)$$

$$c_spec \rightarrow \langle available_x \mid total_x \rangle \quad (5)$$

$$x \rightarrow \langle cpu; dsk; ram \rangle \quad (6)$$

The app_spec (Eq. (7)) includes app_id as the application identifier, app_req detailing the application's resource requirements, and $links$ representing network requirements and interdependencies.

$$app_spec \rightarrow \langle app_id \mid app_req \mid links \rangle \quad (7)$$

$$app_req \rightarrow \langle m_id \mid req_x \mid CONS \mid rep_set \rangle \quad (8)$$

$$links \rightarrow \langle t_req \mid req_thr \mid d_trans \mid t_time \mid dist \rangle \quad (9)$$

app_req (Eq. (8)) defines resource requirements req_x per component and replica count rep_set . The $links$ set (Eq. (9)) includes total requests t_req (as tuple $\langle m_id \mid m_id \rangle$), request threshold req_thr , data volume d_trans , transmission time t_time , and component distance $dist$.

RALA-PSO defines a particle (Eq. (10)) with pos as the assigned cluster, v as the velocity controlling search direction

and speed, *fitness* as the current solution quality, and b_{pos} and $b_{fitness}$ as the particle's best position and fitness to date.

$$\langle pos \mid v \mid b_{pos} \mid fitness \mid b_{fitness} \rangle \quad (10)$$

B. Design Principles and Algorithm

RALA-PSO retains PSO's advantages, including the ability to handle constrained and multi-objective optimisation problems and the simplicity afforded by its minimal parameter requirements. Furthermore, RALA-PSO introduces five key enhancements to the standard PSO algorithm:

- The incorporation of a greedy approach combined with probabilistic reasoning (cf. Section III-C), employed during population initialisation to ensure particles start in feasible (or at least sub-optimal) positions.
- A circular search space used to prevent particles from becoming trapped at the minimum or maximum boundaries (cf. Section III-D).
- A multi-objective fitness function to balance multiple criteria of resource scheduling (cf. Section III-E).
- A heuristic method for classifying cluster suitability.
- Integration of adaptive weights (Eq. (11)) and learning factors (Eqs. (12) and (13)) adapted from [12].

RALA-PSO (Algorithm 1) begins by initialising the best fitness value ($b_{fitness}$) and the best solution. To reduce the inherent stochastic variability of PSO and improve the chances of finding a high-quality global solution, it performs multiple independent executions ($N_{executions}$, line 3). For each execution, the algorithm starts by initialising the particle (see Section III-C). This involves, for each component, identifying feasible clusters with available resources, calculating allocation scores, assigning initial positions probabilistically based on these scores, and updating the remaining cluster resource availability accordingly (lines 5–8). During each iteration ($max_{iterations}$), the algorithm updates each particle (cf. Section III-D), which includes adjusting the inertia weight (w) and learning factors (c_1, c_2) to balance exploration and exploitation, evaluating the feasibility of the current solution, and updating the particle's velocity and position (lines 11–14). After movement, the particle's fitness is evaluated (cf. Section III-E), and if an improved position is found, the corresponding best values are updated (line 16). At the end of each execution, the algorithm evaluates the fitness of the current solution, i.e., the fitness of all particles. If this fitness is greater than or equal to the best fitness recorded so far, it updates the global best fitness and the corresponding solution (lines 21–22). After completing all executions, the algorithm returns the best solution found along with its fitness value (line 25).

C. Particle Swarm Initialisation

Unlike the random cluster selection in the original PSO, we introduce a greedy and probabilistic method for particle initialisation. The greedy aspect ensures particles are initialised within feasible regions of the search space, meaning components are assigned only to clusters with sufficient resources to meet their requirements. Complementing this, the probabilistic

Algorithm 1 RALA-PSO Algorithm

Require: $\langle res_spec \mid app_spec \rangle$, global parameters

Ensure: Optimal component location (by maximising fitness function)

```

1:  $BestFitness \leftarrow 0$ 
2:  $solution \leftarrow \text{None}$ 
3: for each execution  $e = 1$  to  $N_{executions}$  do
4:   for each component do
5:     Identify feasible clusters (with sufficient resources)
6:     Compute allocation scores considering constraints
7:     Assign initial position based on larger allocation score
8:     Update clusters' resources availability considering the initial position
9:   end for
10:  for each iteration  $i = 1$  to  $max_{iterations}$  do
11:    Update inertia weight  $w$  and learning factors  $c_1, c_2$ 
12:    Check feasibility of current particle allocation
13:    for each particle do
14:      Update velocity ( $v$ ) and position ( $p$ )
15:      Evaluate fitness particle
16:      Update local and global bests if improved
17:    end for
18:  end for
19:  Evaluate fitness of current solution
20:  if current fitness  $\geq BestFitness$  then
21:     $BestFitness \leftarrow$  current fitness
22:     $solution \leftarrow$  current particle positions
23:  end if
24: end for
25: return  $Solution, BestFitness$ 

```

approach computes allocation scores, enabling components to be placed in higher-scoring clusters with greater probability rather than always selecting the single best option deterministically. It is relevant to note that this strategy reduces initial diversity compared to random sampling, as choices are biased toward higher scores, limiting exploration in some instances. Additionally, computing feasible clusters and allocation scores for each particle introduces extra computational overhead during initialisation compared to random sampling. Nevertheless, in Cloud-Native environments, prioritising feasibility and resource-aware preferences is generally more beneficial than complete randomness, as it accelerates convergence toward valid and high-quality solutions. Hence, combining greedy and probabilistic initialisation reduces the risk of infeasible or poor-quality initial placements while still maintaining enough diversity to prevent premature convergence. The allocation score is calculated as a weighted sum of the Resource-Aware score and the Constraint-Aware score, each contributing 50%. The Resource-Aware score measures a cluster's attractiveness based on its available resources, favouring those with greater remaining capacity. The Constraint-Aware score incentivises clusters that have already been initialised and host components on which the current particle depends, encouraging co-location and reducing potential inter-component latency. To translate these scores into selection probabilities, they are normalised using a softmax function, ensuring clusters with higher scores have proportionally greater chances of being chosen, while still allowing exploration of other feasible options.

D. Particle Evolution

In RALA-PSO, particle evolution is enhanced through the introduction of adaptive inertia weights (w) and learning factors (c_1, c_2). Specifically, RALA-PSO adopts the adaptive weight mechanism from Eq. (11) and the dynamic learning factors defined in Eq. (12) and Eq. (13), initially proposed by Liu et al. [12]. This novel integration enables the algorithm to dynamically balance exploration and exploitation during the search, preventing premature convergence and improving the ability to escape local optima. As a result, RALA-PSO achieves faster convergence and higher-quality scheduling solutions.

$$\omega(t+1) = \omega_{\min} + (\omega_{\max} - \omega_{\min}) \exp \left[-\frac{\theta \cdot t^2}{2 \cdot t_{\max}} \right] \quad (11)$$

The inertia weight ω is calculated at each iteration t using the initial and termination inertia weights, $\omega_{\max}$ and $\omega_{\min}$, respectively, along with a new parameter θ , which controls the smoothness of the weight decay curve. The term $t_{\max}$ represents the maximum number of iterations of the algorithm.

The learning factors are updated at each iteration t , as shown in Eq. (12) and Eq. (13).

$$c_1 = c_{1-\max} - \frac{c_{1-\max} - c_{1-\min}}{t_{\max}^2} \cdot t^2 \quad (12)$$

$$c_2 = c_{2-\min} - \frac{c_{2-\max} - c_{2-\min}}{t_{\max}^2} \cdot t^2 \quad (13)$$

The adaptive particle evolution approach initially uses higher weights and learning factors to encourage exploration. These values gradually decrease, promoting convergence towards the best global solution. The position (Eq. (1)) is updated using the nearest value of the sum of the current position and velocity. To prevent particles from getting trapped at boundaries, a circular search space is used. Thus, when a particle moves beyond one boundary, it re-enters the search space from the opposite limit, effectively enabling continuous position updates.

E. Fitness Function

We introduced a fitness function (Eq. (14)) that evaluates the suitability of allocating a component to a cluster by balancing local resource utilisation and latency reduction. The parameters max_cpu and max_ram were set to 95% and 90%, respectively, meaning that the available CPU on a cluster must be at least 5% and the available RAM must be at least 10%. Values below these thresholds are considered invalid, rendering the solutions infeasible. These parameter values are based on empirical experience and reflect commonly accepted industry standards [12].

$$f(m_j, c_i) = \begin{cases} -\infty & \text{if infeasible} \\ 1 + w_1 \cdot R + w_2 \cdot L + w_p \cdot P & \text{otherwise} \end{cases} \quad (14)$$

Where m_j represents component j , c_i represents cluster i , and w_1, w_2 , and w_p are weighting coefficients, and R measures the equilibrium between different resource utilisations (RAM, CPU, and storage) (Eq. (15)). The resource usage proportions are calculated using Eq. (17), where x and y represent $\langle ram, cpu, disk \rangle$. This term encourages a balanced use of resources, achieving a maximum score of 1 when the usage of all resource types is equal.

$$R = 1 - \frac{\Delta_{ram,cpu} + \Delta_{ram,disk} + \Delta_{cpu,disk}}{3} \quad (15)$$

$$\Delta_{x,y} = |r_x - r_y| \quad (16)$$

$$r_x = \frac{m_j.replicas \times m_j.x}{c_i.total_x} \quad (17)$$

Additionally, terms L and P are used to guide optimisation. In Eq. (18), L measures the satisfaction of component co-location constraints by encouraging interdependent components to be placed within the same cluster.

$$L = \sum_{m \in C_j} (x_m = k_i) \quad (18)$$

Where C_j denotes the set of components co-located with component c_j , and x_m representing the cluster assigned to component m , this term counts the number of satisfied co-location constraints for a given placement. Whereas P (Eq. (19)) serves as a penalty term that discourages placements resulting in insufficient available resources within a cluster, thereby ensuring the allocation remains feasible concerning resource capacity.

$$P = - \left(\frac{ram_penalty + cpu_penalty}{2} \right) \quad (19)$$

Resource penalties $ram_penalty$ and $cpu_penalty$ apply when available RAM or CPU drops below thresholds ram_thr and cpu_thr , set by 90% and 95%. Penalties are normalised relative to total resource capacity, providing a quantifiable measure of resource scarcity.

IV. EVALUATION

RALA-PSO was evaluated against two state-of-the-art SI algorithms, MOPPSO-CMS [3] and ACO-CMS [4], as well as a non-AI greedy approach, Spread. The evaluation employed these algorithms to optimise the placement of a reference application across clusters and assess its impact in terms of (1) resource efficiency, (2) network transmission overhead, and (3) algorithm computation time. All algorithms were tested under identical conditions, replicating data and metrics from prior studies [3], [4]. The RALA-PSO algorithm settings are summarised in Table I. The reference application, comprising 17 microservices, was sourced from the public Alibaba Cluster TraceV2018 dataset [13], which provides normalised data on resource usage and communication patterns for each component. Workloads, the number of application components

submitted to the algorithm, were generated by scaling the number of application instances, from 1.0 to 5.0 in increments of 0.5. To evaluate algorithm robustness under varying infrastructure conditions, each workload configuration was executed 30 times using different search spaces, i.e., distinct sets of candidate clusters available for component placement. Each search space consisted of 100 clusters, with resource capacities drawn from predefined reference sets based on the Alibaba Cluster TraceV2018 dataset, $total_cpu$ and $total_disk \in \{100, 200, 400\}$, and $total_ram \in \{60, 120, 180\}$. Each experiment run was repeated 30 additional times to ensure statistical robustness. The experiments were conducted on commodity hardware, equipped with an Intel Core i7-10750H processor and 16 GB of RAM.

TABLE I
RALA-PSO PARAMETERS CONFIGURATION

Parameter	Value	Parameter	Value
max_cpu	0.95	c1_min	0.01
max_ram	0.90	c1_max	2.5
weights	[0.4, 0.5, 0.1]	c2_min	0.01
theta	3.4	c2_max	2.5
max_itr	120	w_max	0.7
t_max	100	w_min	0.5

Resource efficiency (Eq. (20)) was used to measure the balance of resource utilisation across cluster nodes, considering CPU, RAM, and storage. It was computed as the average of normalised pairwise differences in resource usage. Whereas, the network transmission overhead (Eq. (21)) quantifies the communication cost incurred by the placement of components, reflecting the additional overhead introduced when dependent components are placed further apart across clusters.

$$local_lb = \frac{\sum_{j=1}^n (\Delta_{ram,cpu,j} + \Delta_{ram,disk,j} + \Delta_{cpu,disk,j})}{3n} \quad (20)$$

Where $\Delta_{x,y,j}$ denotes the resource usage difference between resources x and y on node j , with n being the total number of cluster nodes.

$$t_req(cont_i, cont_k^{type}) \cdot d_trans(cont_i, cont_k^{type}) \cdot Dist(cont_i, cont_k^{type}) \cdot PassTime(cont_i, cont_k^{type}) \quad (21)$$

Eq. (21) considers t_{req} and d_{tran} (from Eq. (9)) to calculate the total volume of data transmitted. The communication cost factors in the distance (d_{trans}) between clusters and the transmission time ($dist$). It is assumed that during internal communication, these values are approximately equal to 1, thus having minimal impact on communications. In contrast, during external communication, these values are significant and are set to 4. Finally, the algorithm computation time measures the duration taken to complete the placement decision for all components, starting from the moment the request is submitted.

A. Results and Discussion

The experimental results are summarised in Figures 1, 2, and 3, referring to network transmission overhead, resource efficiency, and algorithm computation time, respectively.

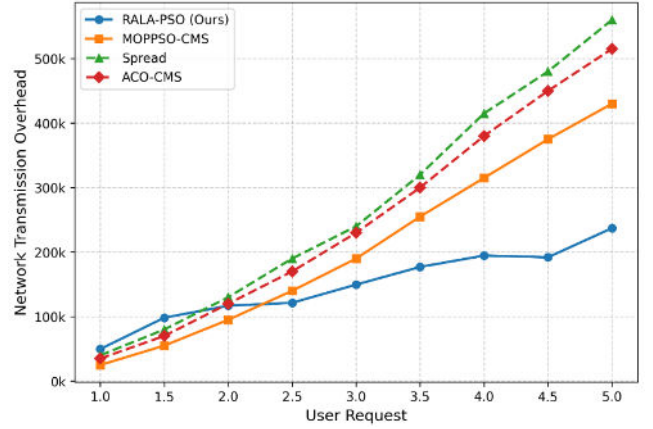


Fig. 1. Normalised network transmission overhead cost incurred when dependent components are placed further apart across clusters. Lower values indicate better performance.

At lower user request levels (1.0x to 2.0x), RALA-PSO yields network overhead costs of approximately 49K and 117K (normalised values), comparable to MOPPSO-CMS's 25K and 95K. Nevertheless, as the workload increases, RALA-PSO outperforms the remaining. Notably, at the peak workload (5x), MOPPSO-CMS's network transmission cost rises to 430K, approximately 1.8 times worse than RALA-PSO's 237K. ACO-CMS and Spread exhibit even poorer scalability, both exceeding 500K under high load conditions. This overall lower network overhead highlights RALA-PSO's enhanced ability to account for inter-component communication dependencies, which is critical for performance and scalability in cloud-native environments. RALA-PSO's superior results arise from its component co-location strategy, which integrates dependencies during both particle initialisation and evolution, promoting the deployment of dependent components in proximity. The initial network overhead occurs because RALA-PSO equally weights inter-component dependencies and resource usage. At low request volumes, fewer active particles lessen dependency impact, causing resource optimisation to dominate, resulting in more dispersed placements and increased cross-node communication.

MOPPSO-CMS consistently demonstrates superior performance in balancing resource utilisation across clusters, achieving the lowest resource utilisation values across all workloads, from 0.018 at 1.0x to 0.105 at 5.0x. RALA-PSO and ACO-CMS follow a similar trend, with RALA-PSO reaching 0.126 and ACO-CMS 0.129 at the peak workload (5.0x), while Spread performs the worst, peaking at 0.138. The slightly reduced resource efficiency performance of RALA-PSO reflects an intentional design trade-off that balances network cost minimisation with efficient resource utilisation. This trade-off also highlights RALA-PSO's suitability for multi-objective

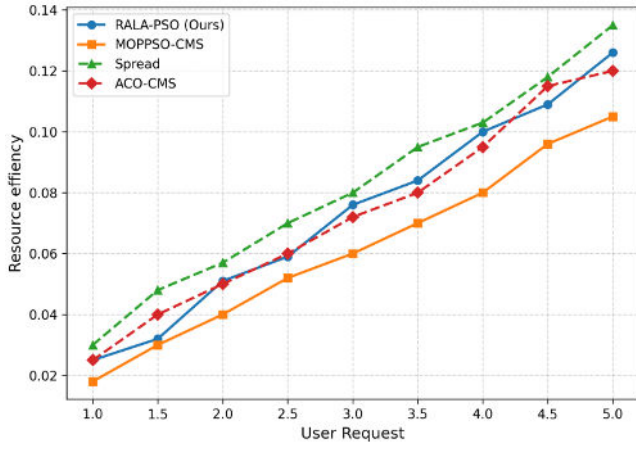


Fig. 2. Normalised resource utilisation balance across cluster nodes (CPU, RAM, storage). Lower values indicate better performance.

optimisation scenarios, where competing goals must be jointly addressed.

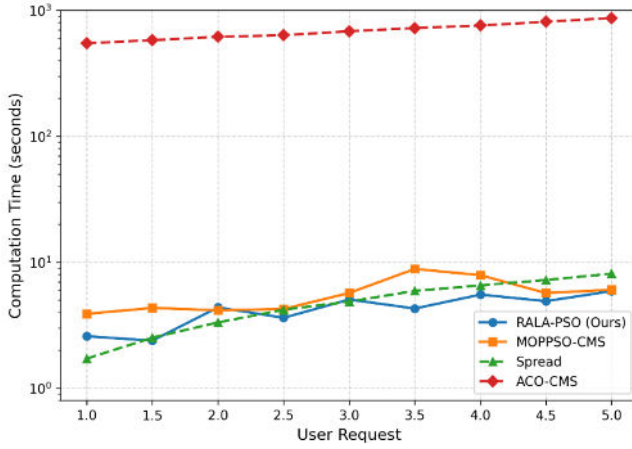


Fig. 3. Algorithm runtime (in seconds) to compute placement decisions. Lower values indicate better performance.

Algorithm computation time reflects both efficiency and practical applicability, with lower values indicating better performance. RALA-PSO overall delivered faster execution times, rising gradually from 2.59 to 5.91 seconds as the workload increased. Its lightweight design and efficient use of NumPy (e.g by using vectorised operations) translate to lower computational overhead, enabling rapid and scalable scheduling, key for dynamic cloud-native environments.

V. CONCLUSIONS

This paper presented RALA-PSO, a novel SI-based scheduling algorithm designed for the demands of dynamic, cloud-native environments. RALA-PSO extends the widely known and well-established PSO algorithm, which is valued for its simplicity, flexibility, and effectiveness in solving multi-objective optimisation problems. To adapt PSO for microservice scheduling, RALA-PSO introduces several key modifications. RALA-PSO was evaluated using the Alibaba Cluster

TraceV2018 dataset and benchmarked against state-of-the-art scheduling methods, including MOPPSO-CMS and ACO-CMS, along with the non-AI baseline Spread. Experimental results demonstrate that RALA-PSO reduces network transmission overhead under increasing workloads, improves runtime efficiency, and maintains competitive resource utilisation efficiency. These results underscore the effectiveness of RALA-PSO's PSO-based enhancements, which enable context-aware, resource-efficient scheduling under cloud-native constraints. By modelling communication patterns and feasibility directly in the optimisation process, RALA-PSO achieves a balanced trade-off between performance, feasibility, and execution time.

ACKNOWLEDGMENT

This work is funded by national funds from FCT - Fundação para a Ciência e a Tecnologia, I.P., within the framework of the Projects UIDB/00326/2025 and UIDP/00326/2025. This work was also supported in part by the 6G-PATH project (Grant No. 101139172) and by the COP-PILOT project (Grant No. 101189819).

REFERENCES

- [1] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine learning-based orchestration of containers: A taxonomy and future directions," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 217:1–217:35, Sep. 2022, doi: 10.1145/3510415.
- [2] T. Dias, L. Ferreira, D. Fevreiro, L. Rosa, L. Cordeiro, and J. Fernandes, "Cloud-native scheduling and resource orchestration: A deep dive into AI-driven approaches," in *Artificial Intelligence Applications and Innovations. AIAI 2025 IFIP WG 12.5 Int. Workshops*, Cham: Springer Nature Switzerland, 2025, pp. 101–114.
- [3] X. Chen and S. Xiao, "Multi-objective and parallel particle swarm optimization algorithm for container-based microservice scheduling," *Sensors*, vol. 21, no. 18, p. 6212, 2021.
- [4] M. Lin, J. Xi, W. Bai, and J. Wu, "Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud," *IEEE Access*, vol. 7, pp. 83088–83100, 2019, doi: 10.1109/ACCESS.2019.2924414.
- [5] X.-S. Yang, S. Deb, Y.-X. Zhao, S. Fong, and X. He, "Swarm intelligence: Past, present and future," *Soft Comput.*, vol. 22, pp. 5923–5933, 2018.
- [6] H. Ahmed and J. Glasgow, "Swarm intelligence: Concepts, models and applications," Tech. Rep., Sch. Comput., Queen's Univ., 2012.
- [7] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. ICNN'95 - Int. Conf. Neural Netw.*, vol. 4, 1995, pp. 1942–1948.
- [8] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. 1998 IEEE Int. Conf. Evol. Comput. (IEEE World Congr. Comput. Intell.)*, Anchorage, AK, USA, 1998, pp. 69–73.
- [9] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, 2006. [Online]. Available: <https://doi.org/10.1109/MCI.2006.329691>. Accessed: Jun. 26, 2025.
- [10] M. Jain, V. Saihpal, N. Singh, and S. B. Singh, "An overview of variants and advancements of PSO algorithm," *Appl. Sci.*, vol. 12, no. 17, Art. no. 8392, 2022. [Online]. Available: <https://www.mdpi.com/2076-3417/12/17/8392>. Accessed: Jun. 28, 2025.
- [11] N. Arora and R. K. Banyal, "Hybrid scheduling algorithms in cloud computing: a review," *International Journal of Electrical & Computer Engineering*, vol. 12, no. 1, 2022. [Online]. Available: <https://doi.org/10.11591/ijeece.v12i1.pp1-12>. Accessed: Jun. 28, 2025.
- [12] B. Liu, J. Li, W. Lin, W. Bai, P. Li, and Q. Gao, "K-PSO: An improved PSO-based container scheduling algorithm for big data applications," *Int. J. Netw. Manag.*, vol. 31, no. 2, p. e2092, 2021. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2092>. Accessed: Jun. 29, 2025.
- [13] Alibaba Corp., "Alibaba Cluster Trace V2018," 2018. [Online]. Available: <https://github.com/alibaba/clusterdata>. Accessed: Jun. 29, 2025.
- [14] OneSource Consult, "RALA-PSO" 2025. [Online]. Available: <https://github.com/OneSourceConsult/rala-psy>. Accessed: Jul. 30, 2025.