Research on Improved Chicken Swarm Algorithm for Cloud Computing Resource Scheduling (Post-print)

Authors: Chen Xuan, Long Dan

Date: 2018-05-24T00:00:00+00:00

Abstract

To address the issue of low resource scheduling efficiency in cloud computing, an improved chicken swarm algorithm is proposed for scheduling. First, the concept of opposition-based learning is introduced to initialize the chicken swarm population, thereby enhancing global search capability. Second, the concepts of inertia weight and learning factors from the particle swarm optimization algorithm are incorporated into the position update of chicks to improve individual positions within the swarm. Third, the differential evolution algorithm is employed to optimize the overall individual positions in the chicken swarm algorithm. Finally, boundary handling is implemented to comprehensively prevent potential out-of-boundary issues for individual positions in the algorithm. In simulation experiments, the optimized chicken swarm algorithm is compared with the basic chicken swarm algorithm, particle swarm optimization, and ant colony algorithm in terms of completion time, cost, energy consumption, and load balancing, achieving favorable results.

Full Text

Preamble

Title: Research on Improved Chicken Swarm Optimization in Cloud Computing Resource Scheduling

Authors: Chen Xuan[1], Long Dan[2]
Affiliations:
1. Zhejiang Industry Polytechnic College, Shaoxing, Zhejiang 312000, China
2. Zhejiang University, Hangzhou 310058, China

Abstract: To address the low efficiency of resource scheduling in cloud computing, this paper proposes applying an improved chicken swarm optimization algorithm to the scheduling problem. First, the concept of opposition-based

learning is introduced to initialize the chicken swarm population, enhancing global search capability. Second, the position update for chicks incorporates the weight value and learning factor concepts from particle swarm optimization to improve individual positions within the swarm. Third, the differential evolution algorithm is used to optimize the overall individual positions in the chicken swarm algorithm. Finally, boundary processing is applied to prevent potential position boundary violations across the algorithm. Simulation experiments compare the optimized chicken swarm algorithm with the basic chicken swarm algorithm, particle swarm optimization, and ant colony algorithm in terms of completion time, cost, energy consumption, and load balancing, demonstrating superior performance.

Keywords: chicken swarm optimization; opposition-based learning; learning factor; differential algorithm

0 Introduction

Efficient resource scheduling represents a critical focus in current cloud computing research. Numerous scholars have applied various intelligent algorithms—including genetic algorithms, particle swarm optimization, and ant colony algorithms—to resource scheduling with favorable results. Meng proposed the Chicken Swarm Optimization (CSO) algorithm in 2014, a population-based stochastic optimization method characterized by simple implementation but prone to local optima, slow convergence, and low precision. Subsequent research has explored CSO applications in diverse domains. For instance, improved CSO algorithms have been applied to micro-grid index optimization and workshop job scheduling problems, yielding enhanced efficiency. However, these studies often lack comprehensive comparisons with the basic CSO algorithm or classical scheduling algorithms, limiting the persuasiveness of their optimization results. Other work has adapted CSO for multi-classifier coefficient optimization and water resource scheduling, achieving promising outcomes but similarly lacking comparisons with established methods in their respective fields.

Building upon these studies, this paper employs an Improved Chicken Swarm Optimization (ICSO) algorithm for cloud computing resource scheduling. Through enhanced population initialization, chick position updates, differential algorithm-based optimal individual selection, and boundary processing, the algorithm' s local and global search capabilities are improved. Cloud computing simulation results demonstrate that the proposed algorithm offers advantages over several classical intelligent algorithms, effectively enhancing cloud computing resource scheduling efficiency.

1 Cloud Computing Resource Scheduling Description

Task-to-resource allocation in cloud computing constitutes a complete NP-hard problem. This paper addresses resource allocation from both user requirements and resource owner perspectives by introducing constraints on time, cost, and energy consumption to propose a cloud computing resource scheduling model that satisfies user QoS demands while effectively achieving resource load balancing.

Consider a scenario with n tasks sharing m resources. Each task S_i consists of a_i parallel and interdependent subtasks with equal computational loads. Each computing resource R_j has a fixed price P_j and energy consumption M_j. When multiple subtasks are assigned to R_j, they proportionally share its computing capacity.

The core resource allocation problem involves solving an $n \times m$ matrix A, where rows represent tasks and columns represent resources. Element a_{ij} indicates the number of subtasks from task S_i allocated to resource R_j, with the constraint $\sum_{j=1}^{m} a_{ij} = a_i$. Similarly, we derive three matrices for time, cost, and energy consumption, denoted as T, E, and U respectively.

Matrix element t_{ij} represents the time resource R_j spends completing the a_{ij} subtasks allocated from task S_i. Since subtasks execute in parallel, task S_i' s completion time is $t_i = \max_{j\in[1,m]}\{t_{ij}\}$. Matrix element e_{ij} represents the cost for resource R_j to complete its allocated subtasks from task S_i, giving task S_i' s total cost as $e_i = \sum_{j=1}^{m} e_{ij}$. Matrix element u_{ij} represents the energy consumption for resource R_j to process its allocated subtasks from task S_i, yielding task S_i' s energy consumption as $u_i = \sum_{j=1}^{m} u_{ij}$.

To maintain a balanced relationship among these three metrics, weight values ω_t, ω_e, and ω_u are assigned with $\omega_t + \omega_e + \omega_u = 1$. The total cost for task S_i is given by:

$$C_i = \omega_t \cdot t_i + \omega_e \cdot e_i + \omega_u \cdot u_i$$

Each task aims to maximize its utility by minimizing cost, time, and energy consumption. The total utility for task S_i is:

$$U_i = \frac{1}{C_i} = \frac{1}{\omega_t \cdot t_i + \omega_e \cdot e_i + \omega_u \cdot u_i}$$

The optimal value for cloud computing task-resource scheduling utility is therefore the maximum total utility across all tasks:

$$U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{1}{\max_{j\in[1,m]}\{t_{ij}\} \cdot \omega_t + \sum_{j=1}^{m} e_{ij} \cdot \omega_e + \sum_{j=1}^{m} u_{ij} \cdot \omega_u}$$

2 Chicken Swarm Algorithm

The Chicken Swarm Optimization algorithm models optimization problems as a chicken flock searching for food, simulating the hierarchical classification and foraging behaviors observed in chicken flocks. The algorithm divides the entire flock into multiple subgroups, each comprising one rooster, several hens, and several chicks. Due to different hierarchies, competition exists among subgroups to obtain optimal positions. The specific process is as follows:

a) The flock contains multiple subgroups, each with one rooster, several hens, and several chicks.

b) In CSO, individuals with the best fitness values are designated as roosters distributed across different subgroups, while those with the worst fitness become chicks. The remaining individuals serve as hens, which can freely choose subgroups and their corresponding chicks. The mother-chick relationships are randomly assigned.

c) Hierarchical classification and dominance relationships only change during updates. Once established, the hen-chick relationship remains unchanged until the next update.

d) All subgroup members search for food centered around their rooster while preventing individuals from other subgroups from encroaching. Chicks can 抢夺 food from other subgroups and search alongside their mother hens. Certain individuals possess priority in food location, gaining competitive advantages.

All chickens move according to their own patterns until finding optimal positions. Thus, individual chicken positions correspond to solutions of the optimization problem, with the best-positioned individual representing the optimal solution. Throughout the CSO algorithm, the total number of chickens is N, with each chicken' s position represented by $x_{i,j}^t$, indicating the position of the i-th chicken individual in the j-th dimension during the t-th iteration. Different chicken types have distinct position update mechanisms.

Roster Position Update:
Roosters with the best fitness values in each subgroup search broadly for food. The position update for roosters is:

$$x_{i,j}^{t+1} = x_{i,j}^t \cdot (1 + \text{Randn}(0, \sigma^2))$$

where $\text{Randn}(0, \sigma^2)$ is a Gaussian distribution with mean 0 and variance σ^2, calculated as:

$$\sigma^2 = \begin{cases} 1, & \text{if } f_i \leq f_k \\ \exp\left(\frac{f_k - f_i}{|f_i| + \varepsilon}\right), & \text{otherwise} \end{cases}$$

Here, k ($k \in [1, N]$, $k \neq i$) represents another rooster individual, and ε is a small constant.

Hen Position Update:
The position update for hens is:

$$x_{i,j}^{t+1} = x_{i,j}^t + S1 \cdot \text{rand} \cdot (x_{r1,j}^t - x_{i,j}^t) + S2 \cdot \text{rand} \cdot (x_{r2,j}^t - x_{i,j}^t)$$

where rand is a uniform random number in $[0, 1]$, $r1$ is the rooster in hen i' s subgroup, and $r2$ is a randomly selected individual from the entire flock (rooster or hen) with $r1 \neq r2$. The parameters $S1$ and $S2$ are calculated as:

$$S1 = \exp\left(\frac{f_i - f_{r1}}{|f_i| + \varepsilon}\right), \quad S2 = \exp(f_{r2} - f_i)$$

Chick Position Update:
The position update for chicks is:

$$x_{i,j}^{t+1} = x_{i,j}^t + F \cdot (x_{m,j}^t - x_{i,j}^t)$$

where $x_{m,j}^t$ represents the mother hen corresponding to chick i, and F is a following coefficient indicating how the chick follows its mother to find food.

---

3 Improved Chicken Swarm Algorithm for Cloud Computing Scheduling Design

3.1 Initialization Selection

The original CSO algorithm lacks population initialization, preventing optimal solutions from being uniformly distributed in the search space and limiting algorithm efficiency and performance. Opposition-based learning is a machine learning optimization strategy that generates opposite solutions for current solutions during each iteration and selects evolutionarily beneficial solutions from both sets, reducing algorithmic blindness. This paper employs an optimized opposition-based learning strategy to expand the initial search range, further enhancing global search capability and avoiding local optima.

The process is as follows:
a) Randomly select N individuals to form the initial population $P = \{x_1(t), x_2(t), \ldots, x_N(t)\}$, where $x_i(t) = (x_{i,1}(t), x_{i,2}(t), \ldots, x_{i,D}(t))$ for $i \in [1, N]$.

b) Generate the opposite population $P_{opp} = \{x_1^{opp}(t), x_2^{opp}(t), \ldots, x_N^{opp}(t)\}$ corresponding to P, calculating each opposite individual $x_i^{opp}(t)$ using the method from reference [16].
c) Select the N individuals with best fitness values from the combined set $\{P \cup P_{opp}\}$ as the final initial population.

This opposition-based approach enables the algorithm to search for optimal values in a larger space, guiding individuals toward evolutionary directions and improving overall convergence speed.

3.2 Chick Position Update

In the original CSO algorithm, chick position updates only follow the mother hen' s movement without considering the rooster' s optimal position, potentially causing individuals to fall into local optima and reducing overall algorithm efficiency. CSO chicks share similarities with PSO particles—PSO particles obtain local and global optimal positions, analogous to chicks obtaining local positions near their mother and global optimal positions guided by the rooster. Therefore, this paper incorporates PSO' s learning factor concept to update equation (8):

$$x_{i,j}^{t+1} = \omega \cdot x_{i,j}^{t} + \lambda_1 \cdot (x_{m,j}^{t} - x_{i,j}^{t}) + \lambda_2 \cdot (x_{r,j}^{t} - x_{i,j}^{t})$$

where $x_{m,j}^{t}$ represents the mother hen corresponding to chick i, $x_{r,j}^{t}$ is the rooster in chick i' s subgroup, λ_1 and λ_2 are learning factors indicating the degree to which the chick learns from its mother and rooster respectively, and ω is the weight value.

3.2.1 Weight Value Setting Weight value setting affects population diversity in later algorithm stages. This paper adopts a nonlinear dynamic inertia weight expression:

$$\omega = \omega_{\max} - (\omega_{\max} - \omega_{\min}) \cdot \left(\frac{t}{T_{\max}}\right)^{k} \cdot \exp\left(1 - \frac{t}{T_{\max}}\right)$$

where $\omega_{\max}$ and $\omega_{\min}$ are the maximum and minimum inertia weights, $T_{\max}$ is the maximum iteration number, t is the current iteration, and k is a control factor for weight adjustment. This ensures global search in early stages to improve precision and strengthens local search in later stages to enhance solution quality.

3.2.2 Learning Factor Setting Learning factor values determine the extent to which chicks learn from hens and roosters. Small λ_2 values allow chicks to find better solutions within their current group, while large values provide broader search space. However, improper λ_2 settings can cause premature convergence or divergence. Therefore, λ_1 and λ_2 require limitation:

For λ_1:

$$\lambda_1 = \begin{cases} \lambda_{\min} \times 2, & t > \frac{T_{\max}}{2} \\ \lambda_{\min}, & t \leq \frac{T_{\max}}{2} \end{cases}$$

For λ_2:

$$\lambda_2 = \begin{cases} \lambda_{\max} + \lambda_{\max} \times 2, & t < \frac{T_{\max}}{2} \\ \lambda_{\max}, & t \geq \frac{T_{\max}}{2} \end{cases}$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimum and maximum learning factor values, $t_{current}$ is the current iteration, and $t_{\max}$ and $t_{\min}$ are the defined maximum and minimum iteration counts. This configuration enables chicks to better learn from both roosters and hens.

3.3 Introducing Differential Evolution for Optimal Individual Selection

For optimal position selection in CSO, this paper employs differential evolution—a heuristic random search algorithm that operates through mutation, crossover, and selection.

a) Mutation Operation:
Let $x_{i,j}^t$ represent the j-dimensional individual at generation t. The mutation operation follows:

$$V_{i,j}^{t+1} = x_{r1,j}^t + F \cdot (x_{r2,j}^t - x_{r3,j}^t)$$

where $V_{i,j}^{t+1}$ is the mutated individual, and $F \in [0, 1]$ is a random scaling factor controlling the differential vector' s magnitude.

b) Crossover Operation:
With probability $\kappa \in [0, 1]$, crossover between the j-dimensional individual $x_{i,j}^t$ and mutated individual $V_{i,j}^{t+1}$ produces a new individual; otherwise, the original is retained:

$$\kappa_{i,j}^{t+1} = \begin{cases} V_{i,j}^{t+1}, & \text{if rand} \leq \kappa \\ x_{i,j}^t, & \text{otherwise} \end{cases}$$

c) Selection Operation:
Using a greedy strategy, the fitness values of two individuals are compared. The better individual undergoes mutation and crossover to generate a new individual $\kappa_{i,j}^{t+1}$, which is compared with the previous generation $x_{i,j}^t$. If the former is superior, it proceeds to the next generation; otherwise, $x_{i,j}^t$ is retained.

3.4 Boundary Processing

Literature [18] notes that chicken positions in CSO can exceed boundaries. While chicks follow their mother hens, boundary violations can occur when hens move near search space edges, potentially causing the algorithm to fall into local optima. Therefore, boundary mutation transfers out-of-bound chick positions to feasible regions. Setting random number $r \in (0.2, 1.0)$, if a chick exceeds the upper boundary U, its position is adjusted using equation (18); if it exceeds the lower boundary L, equation (19) applies:

$$x_{i,j} = U - r \cdot (U - L)$$

$$x_{i,j} = L + r \cdot (U - L)$$

3.5 Algorithm Steps

The complete ICSO algorithm proceeds as follows:

a) Map ICSO individuals to cloud computing resource scheduling solutions.

b) Initialize: set maximum iteration count, CSO algorithm parameters, $\omega_{\max}$ and $\omega_{\min}$ as 0.9 and 0.1, $\lambda_{\max}$ and $\lambda_{\min}$ as 0.8 and 0.1, and F as 0.5.

c) Apply opposition-based learning using equations (9)-(11) to optimize the initial population.

d) Update chick positions using equations (13)-(15) and handle potential boundary violations with equations (18)-(19) to improve individual position quality.

e) Select optimal individual positions using equations (15)-(17).

f) If iteration count is less than maximum, return to step d); otherwise, proceed to step g).

g) The best-positioned individual represents the optimal cloud computing resource scheduling solution.

---

4 Simulation Experiments

To demonstrate ICSO' s effectiveness in improving cloud computing resource scheduling efficiency, we compare it with several classical intelligent algorithms: GA, IPSO [19], CSO, and ACO. The hardware environment uses an Intel Core i3-8100 processor, 4 GB DDR4 memory, and 500 GB hard drive, with Windows

7 operating system. Experiments are conducted on the CloudSim platform. Algorithm parameters are listed in Table 1 . We evaluate ICSO from two perspectives: (1) comparing time, cost, and energy consumption for small-scale and large-scale tasks, and (2) comparing load balancing performance.

Table 1: Algorithm Parameters

Parameter	Value
Information 素 evaporation coefficient	0.5
Path selection probability	0.8
PSO learning factor c_1	1.5
PSO learning factor c_2	1.5
Maximum weight value	0.9
Minimum weight value	0.1
ICSO learning factor maximum	0.8
ICSO learning factor minimum	0.1
Boundary random factor	rand

4.1 Task Scale Comparison

4.1.1 Small-Scale Task Performance Comparison Figures 1 [Figure 1: see original paper]-3 show performance comparisons of the four algorithms for small-scale tasks regarding completion time, cost, and energy consumption. Figure 1 demonstrates that with small-scale tasks, competition for user resources and conflicts are minimal, resulting in similar completion times across algorithms, though ICSO and CSO outperform IPSO and ACO. Figure 2 [Figure 2: see original paper] reveals that ICSO incurs lower costs than the other three algorithms, indicating that ICSO improves performance and reduces expenditure. Figure 3 [Figure 3: see original paper] shows energy consumption comparisons: ICSO exhibits higher initial consumption due to opposition-based learning initialization, but as task numbers increase, its curve fluctuates less compared to the other three algorithms, demonstrating better stability in energy consumption.

4.1.2 Large-Scale Task Performance Comparison Figures 4 [Figure 4: see original paper]-6 show performance comparisons for large-scale tasks. Figure 4 illustrates that ICSO has clear advantages in completion time; as task numbers increase, all four algorithms show rising curves, but ICSO' s increase is the slowest, indicating excellent performance. Figure 5 [Figure 5: see original paper] demonstrates ICSO' s cost advantages; algorithmic improvements reduce cost consumption and improve performance. Figure 6 [Figure 6: see original paper] compares energy consumption: as tasks increase, ICSO' s curve remains relatively flat with small fluctuations, while CSO, IPSO, and ACO show steep increases, indicating higher energy consumption. Therefore, ICSO is better suited for large-scale task scheduling.

4.2 Load Balancing Comparison

Setting 10,000 tasks distributed across five resource nodes $\{s_1, s_2, s_3, s_4, s_5\}$ with processing capacities $\{100, 200, 300, 400, 500\}$, Figure 7 [Figure 7: see original paper] shows the comparison results. Due to varying processing capacities, resource loads differ across nodes. ICSO demonstrates more balanced load distribution, significantly improving cloud computing resource allocation effectiveness. In contrast, CSO, IPSO, and ACO produce uneven load values, causing high-capacity nodes to receive fewer tasks while low-capacity nodes become overloaded.

---

5 Conclusion

This paper addresses cloud computing resource allocation by introducing the CSO algorithm and improving it in four aspects. The improved algorithm is applied to cloud computing resource scheduling, with simulation experiments demonstrating that ICSO adapts well to cloud environments and achieves favorable results. However, this study focuses on several common metrics while overlooking the role of virtual machines in task scheduling, which will be investigated in future research.

References

[1] Lin Weiwei, Qi Deyu. Survey of Resource Scheduling in Cloud Computing [J]. Computer Science, 2012, 39(10): 1-6.

[2] Shahdi S. New approach based on group technology for the consolidation problem in cloud computing: mathematical model and genetic algorithm [J]. Computational and Applied Mathematics, 2016, DOI: 10.1007//s40314-.

[3] Kèo D. Cloud computing-based parallel genetic algorithm for gene selection in cancer classification [J]. Neural Comput & Applic, 2016, DOI//10.1007//s00521-016-2780-z.

[4] Masdan A. A survey of pso-based scheduling algorithms in cloud computing [J]. Journal of Network and Systems Management, 2017, 25(1): 122-158.

[5] Kumar N. Resource management using ANN-PSO techniques in cloud environment [C]// Proc of the International Congress on Information and Communication Technology. 2016: 419-428.

[6] Kushwah V S. A basic simulation of ACO algorithm under cloud computing for fault tolerant [C]// Proc of International Conference on Data Engineering and Communication Technology, 2017: 465-472.

[7] Ragmani A. A performed load balancing algorithm for public Cloud computing using ant colony optimization [C]// Proc of the 2nd International Conference on Cloud Computing Technologies and Applications. 2016.

[8] Kong Fei, Wu Dinghui. An improved chicken swarm optimization algorithm [J]. Journal of Jiangnan University: Natural Science Edition, 2015, 14(6): 681-688.

[9] Hu Hanmei, Li Jingya, Huang Jingguang. Economic operation optimization of micro-grid based on chicken swarm optimization algorithm [J]. High Voltage Apparatus, 2017, 53(1): 119-125.

[10] Xu Shipeng, Wu Dinghui, Kong Fei. Solving flexible Job-Shop scheduling problem by improved chicken swarm optimization algorithm [J]. Journal of System Simulation, 2017, 29(7): 1497-1505.

[11] Wu Dinghui, Xu Shipeng. Solving multi-objective flexible Job Shop scheduling problem by the chicken swarm optimization algorithm based on Pareto entropy [J]. Journal of Chinese Computer Systems, 2017, 38(12): 2683-2688.

[12] Hong Yang, Yu Fengqin. Improved chicken swarm optimization and its application in coefficients optimization of multi-classifier [J]. Computer Engineering and Applications, 2017, 53(9): 158-161.

[13] Wei Yuemei, Chi Liming. Application of dissipation chicken swarm optimization in reservoir optimal operation [J]. Water Power, 2017, 43(3): 111-114.

[14] Xu Yixun, Li Wang, Li Dongdong, et al. Disaggregation for non-invasive domestic appliances based on the improved chicken swarm optimization algorithm [J]. Power System Protection and Control, 2016, 44(13): 27-32.

[15] Tizhoosh H R. Opposition-based learning: a new scheme for machine intelligence [C]// Proc of International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelliget Agents, Web Technologies and Internet Commerce. Washington DC: IEEE Computer Society, 2005: 695-701.

[16] Wang Hui, Wu Zhijian, Rahnamayan S, et al. Enhancing particle swarm optimization using generalized opposition-based learning [J]. Information Sciences, 2011, 181(20): 4699-3714.

[17] Bai Yanmin. Particle swarm optimization and its application [D]. Lanzhou: Lanzhou Jiaotong University, 2013: 8-9.

[18] Yang Juting, Zhang Damin, Zhang Muxue. Hybrid improved for chicken swarm optimization algorithm [J/OL]. 2018, 35(11). [2017-11-10]. http://www.arocmag.com/article/02-2018-11-004.html.

[19] Wang Yue, Liu Yaqiu, Guo Jifeng. QoS scheduling algorithm in cloud computing based on discrete particle swarm optimization [J]. Computer Engineering, 2017, 43(6): 111-117.