2023 لسنة A10 العدد
No10A September 2023
المجلة العراقية للبحوث الانسانية والاجتماعية والعلمية
Iraqi Journal of Humanitarian, Social and Scientific Research
Print ISSN 2710-0952-Electronic ISSN 2790-1254

خوارزميات جدولة مهام الحوسبة السحابية: دراسة مقارنة

راقية جاسم محمد
Osamaahmed8817@gmail.com

الملخص:

يقدم هذا البحث دراسة مقارنة شاملة لمختلف خوارزميات جدولة المهام المستخدمة في بيئات الحوسبة السحابية. يعد التخصيص الفعال للمهام على الأجهزة الافتراضية (VMs) في البنية التحتية السحابية أمرًا بالغ الأهمية لتحسين استخدام الموارد والأداء العام للنظام. في هذا البحث، نركز على ثلاث خوارزميات بارزة: تحسين سرب الجسيمات (PSO)، وتحسين مستعمرة النمل (ACO)، ومجموعة مختارة من الخوارزميات التقليدية. تبدأ الدراسة بتقديم خلفية وأهمية جدولة مهام الحوسبة السحابية، وتسليط الضوء على الطلب المتزايد على الحلول القابلة للتطوير والموفرة للموارد. بعد ذلك، يتم تقديم كل خوارزمية، وهي PSO وACO والخوارزميات التقليدية، مع مبادئها الأساسية وأساليبها لجدولة المهام. تتضمن منهجية البحث تنفيذ الخوارزميات المذكورة أعلاه في بيئة سحابية محاكاة وتقييم أدائها باستخدام مقاييس مختلفة. يتم استخدام آثار عبء العمل في العالم الحقيقي لمحاكاة أنماط وصول المهام الديناميكية، مما يعزز أهمية الدراسة وصلاحيتها. يكشف التحليل المقارن للخوارزميات عن نقاط القوة والضعف ومدى ملاءمتها لأنواع مختلفة من المهام والسيناريوهات السحابية. وتناقش العوامل الرئيسية مثل قابلية التوسع، والقدرة على التكيف مع أعباء العمل المتغيرة، والاستجابة لتقلبات الموارد. في الختام، تساهم هذه الورقة في فهم أفضل لخوارزميات جدولة مهام الحوسبة السحابية وتأثيراتها على أداء البنية التحتية السحابية. يمكن أن تساعد النتائج مقدمي الخدمات السحابية والباحثين في اتخاذ قرارات مستنيرة فيما يتعلق باختيار الخوارزمية بناءً على سيناريوهات نشر السحابة المحددة ومتطلبات التطبيق.

الكلمات المفتاحية :الحوسبة السحابية، خوارزمية الجدولة، خوارزمية،

Cloud computing task scheduling algorithms: Comparative Study

Raqia Jassim Mohammed

Abstract

This paper presents a comprehensive comparative study of various task scheduling algorithms used in cloud computing environments. The efficient allocation of tasks on virtual machines (VMs) in a cloud infrastructure is crucial for optimizing resource utilization and overall system performance. In this research, we focus on three prominent algorithms: Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and a selection of traditional algorithms. The study begins by introducing the background and significance of cloud computing task scheduling, highlighting the growing demand for scalable and resource-efficient solutions. Next, each algorithm, namely PSO, ACO, and traditional algorithms, is presented with its underlying principles and approaches to task scheduling. The research methodology involves implementing the aforementioned algorithms in a simulated cloud environment and evaluating their performance using various metrics. Real-

world workload traces are used to simulate dynamic task arrival patterns, enhancing the relevance and validity of the study. The comparative analysis of the algorithms reveals their strengths, weaknesses, and suitability for different types of tasks and cloud scenarios. Key factors such as scalability, adaptability to changing workloads, and responsiveness to resource fluctuations are discussed. In conclusion, this paper contributes to a better understanding of cloud computing task scheduling algorithms and their implications for cloud infrastructure performance. The findings can aid cloud service providers and researchers in making informed decisions regarding algorithm selection based on specific cloud deployment scenarios and application requirements.

Keywords: cloud computing, Scheduling Algorithm , Round Robin

Introduction:

Cloud computing has emerged as a transformative paradigm that offers vast computational resources and scalable services to meet the escalating demands of modern applications [1]. The ability to efficiently allocate tasks on virtual machines (VMs) in a cloud infrastructure is of paramount importance to optimize resource utilization, enhance system performance, and reduce operational costs [2]. As the scale and complexity of cloud infrastructures continue to grow, the need for advanced task scheduling algorithms becomes increasingly critical [3].

Task scheduling in cloud computing involves mapping tasks from applications onto available virtualized resources while considering various constraints and performance objectives [4]. The scheduler must intelligently distribute tasks to VMs to achieve objectives such as minimizing makespan [5], reducing response time [6], balancing the load, and maximizing resource utilization [7]. Moreover, the scheduling algorithms must be able to handle dynamic workloads, fluctuating resource availability, and changing user demands.

In response to these challenges, researchers have developed a variety of task scheduling algorithms, each employing different optimization techniques and strategies. Among the prominent approaches are Particle Swarm Optimization (PSO) [8] and Ant Colony Optimization (ACO) [9], which draw inspiration from nature's collective behaviors to search for optimal solutions. Additionally, traditional task scheduling algorithms, which are widely used in cloud

environments, remain relevant and serve as a valuable reference for comparison [10].

In light of the plethora of task scheduling algorithms available, it becomes crucial to perform a comprehensive comparative study to understand their relative merits and limitations. This research focuses on an in-depth examination of Particle Swarm Optimization, Ant Colony Optimization, and selected traditional algorithms, analyzing their underlying principles, characteristics, and performance.

The primary objective of this study is to explore the strengths and weaknesses of these algorithms, assess their performance under various scenarios, and provide insights into their suitability for different cloud-based applications. By delving into the intricacies of these algorithms and evaluating their performance metrics, this research aims to contribute to a deeper understanding of their capabilities.

The outcome of this comparative study will aid cloud service providers and researchers in making informed decisions in the pursuit of efficient and effective cloud task scheduling strategies. By identifying the most suitable algorithms for specific cloud deployment scenarios and application requirements, cloud providers can enhance resource utilization, optimize energy consumption, and improve the overall quality of service offered to end-users.

In the subsequent sections of this paper, we present the methodologies used for implementing and evaluating the scheduling algorithms in a simulated cloud environment. We then showcase the experimental results and discuss the implications of our findings, offering valuable insights into the design and optimization of task scheduling mechanisms for cloud computing environments.

1. Traditional Scheduling Algorithms:

One of the simplest algorithms for task scheduling is the First Come First Serve (FCFS) algorithm [11]. It is based on real life application of a queue system. It simply schedules the tasks according to their arrival time [12]. However, one can simply imagine why such an algorithm is not normally used in real life scenarios. It neither has the concept of task prioritization nor is a pre-emptive algorithm [13]. This means, once a task schedule has begun, its execution is completed in one sitting. Hence, it is not an effective algorithm and several other algorithms have since developed to increase the performance of task schedulers. These include like Shortest Job First (SJF) [14], Round Robin, Priority Based etc [15].

One of the ways to represent task scheduling is Generalized Activity Normalization Time Table (Gantt chart) [16]. It is a type of bar chart in which tasks completion is shown along with a timeline. For e.g., if FCFS algorithm is being followed and a list of tasks is given as:

Task	Arrival Time	Burst Time
T1	0	1
T2	2	3
T3	2	4
T4	5	2

Table 1. Simple example of list of tasks

Then the Gantt Chart would be as follows:

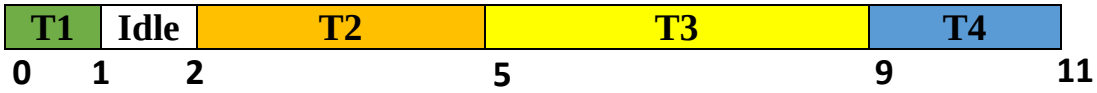

Figure 1. Gantt chart of FCFS algorithm

Shortest Job First (SJF) is a non-preemptive task scheduling algorithm used in operating systems and job scheduling systems. It prioritizes tasks based on their burst time, executing the shortest job first among the available tasks.

Here's a simple example to illustrate the SJF scheduling algorithm, consider the following five process:

Task	Arrival Time	Burst Time
T1	2	6
T2	5	2
T3	1	8
T4	0	3
T5	4	4

Table 2. Simple example of list of tasks

Then the Gantt Chart would be as follows:

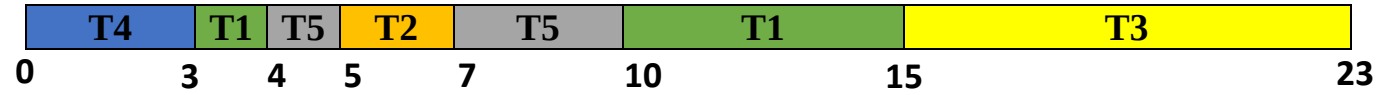

Figure 2. Gantt chart of SJF algorithm

Round-Robin (RR) is a preemptive task scheduling algorithm commonly used in operating systems and time-sharing systems. It assigns a fixed time quantum (also

known as time slice) to each task in the ready queue, and tasks are executed in a circular manner based on their arrival time.

Here's a simple example to illustrate the Round-Robin scheduling algorithm with a time quantum of 3 milliseconds:
Let's consider a set of tasks with their respective burst times in milliseconds:

Task	Arrival Time	Burst Time
T1	0	5
T2	1	6
T3	2	3
T4	3	1
T5	4	5
T6	6	4

Table 3. Simple example of list of tasks

Then the Gantt Chart would be as follows:

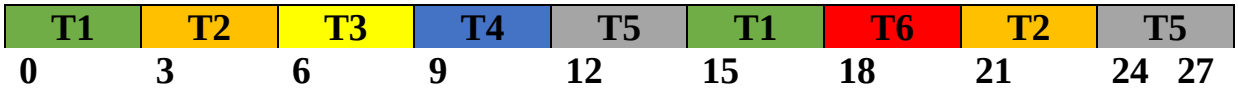

Figure 3. Gantt chart of RR algorithm

As a result by compare the three task scheduling algorithms: First Come First Serve (FCFS), Shortest Job First (SJF), and Round-Robin (RR), based on various criteria:

1) Scheduling Policy:
- FCFS: Non-preemptive. Tasks are executed in the order they arrive.
- SJF: Non-preemptive or Preemptive. Shortest job is executed first.
- RR: Preemptive. Each task is executed for a fixed time quantum and then moved to the back of the queue.

2) Complexity:
- FCFS: Simplest and easy to implement.
- SJF: More complex than FCFS, as it requires knowing the burst times of all tasks beforehand.
- RR: Moderately complex due to the need for time quantum management and preemption.

3) Average Waiting Time:

- FCFS: Can suffer from high average waiting time, especially if longer tasks arrive first (convoy effect).
- SJF: Minimizes average waiting time, as shorter jobs are prioritized.
- RR: Fairly low average waiting time due to time-slicing and equal time allocation.

4) Throughput:

- FCFS: Moderate throughput, but can be suboptimal in certain scenarios.
- SJF: Can achieve good throughput for short and bursty tasks.
- RR: Moderate throughput, may suffer when time quantum is too small or too large.

5) Starvation:

- FCFS: Can lead to starvation for longer tasks if shorter tasks keep arriving.
- SJF: May cause starvation for longer tasks if they continuously arrive with shorter ones.
- RR: Generally avoids starvation as tasks are executed in a cyclical manner.

6) Response Time:

- FCFS: Moderate response time, but can be higher for longer tasks.
- SJF: Short response time for short tasks, but longer tasks may experience higher response time.
- RR: Provides reasonably low response time, especially with a small time quantum.

7) Context Switching Overhead:

- FCFS: Low context switching overhead, as tasks run to completion.
- SJF: Low context switching overhead in non-preemptive mode. Preemptive mode may incur more overhead.
- RR: Moderate to high context switching overhead, especially with smaller time quantum.

8) Predictability:

- FCFS: Highly predictable, as tasks run in the order they arrive.

- SJF: Less predictable due to variations in burst times of tasks.
- RR: Reasonably predictable due to equal time slicing for tasks.

9) Suitability:
- FCFS: Suitable for non-preemptive scenarios or when tasks arrive in an orderly manner.
- SJF: Best suited when burst times are known or can be accurately estimated.
- RR: Suitable for time-sharing systems and scenarios where fairness and responsiveness are essential.

Overall, the choice of the most appropriate scheduling algorithm depends on the specific characteristics of the tasks, system requirements, and performance objectives. Each algorithm has its strengths and weaknesses, and selecting the right one involves considering the trade-offs between fairness, average waiting time, throughput, and response time.

2. Evolutionary Algorithms:

Evolutionary Algorithms (EAs) are a family of optimization algorithms inspired by the principles of natural selection and genetics [17]. These algorithms mimic the process of evolution to iteratively search for optimal solutions to complex problems. EAs are particularly useful for solving optimization problems where traditional methods may struggle due to a large search space or non-linear relationships between variables [18].

The core idea behind evolutionary algorithms is to maintain a population of potential solutions (often referred to as individuals or chromosomes) and use selection, crossover, and mutation operations to create new generations of solutions [19]. The individuals that represent better solutions (according to a defined fitness function) have a higher chance of being selected for reproduction and producing offspring with desirable characteristics.

Here are the key components and steps involved in most evolutionary algorithms:

1) Initialization: Create an initial population of individuals randomly or using domain-specific knowledge.
2) Fitness Evaluation: Evaluate the fitness of each individual in the population based on a fitness function, which quantifies how well a solution performs in the given problem domain [20].

3) Selection: Choose individuals from the population to form the parent pool for the next generation. The probability of selection is usually proportional to the fitness of each individual (higher fitness leads to higher chances of being selected).
4) Recombination (Crossover): Perform crossover (recombination) on pairs of selected individuals to create new offspring. Crossover involves exchanging genetic information between parents to generate diverse solutions.
5) Mutation: Apply mutation to some of the newly created offspring. Mutation introduces random changes to individuals, helping explore new regions of the search space.
6) Replacement: Replace the old population with the new generation of individuals (including parents and offspring).
7) Termination: Repeat the selection, crossover, mutation, and replacement steps for a predefined number of generations or until a termination criterion is met (e.g., a satisfactory solution is found).

2.1 Particle Swarm Optimization:

Particle Swarm Optimization is an optimization technique which applies the concept of population-based search [21]. Here, position of particle is the solution while swarm of particles act as searching agent. PSO find the minimum value for the function.

The PSO particle learns from other particles (social learning) as well as from its own experience (cognitive learning). Due to this, there are two solutions [22]:
gbest and pbest. Velocity V and acceleration play a role here as well since there are changes in position with respect to time (or iteration) plus there is an implementation of random weighted acceleration.
The following figure 4 illustrates the concept of swarm intelligence algorithm:

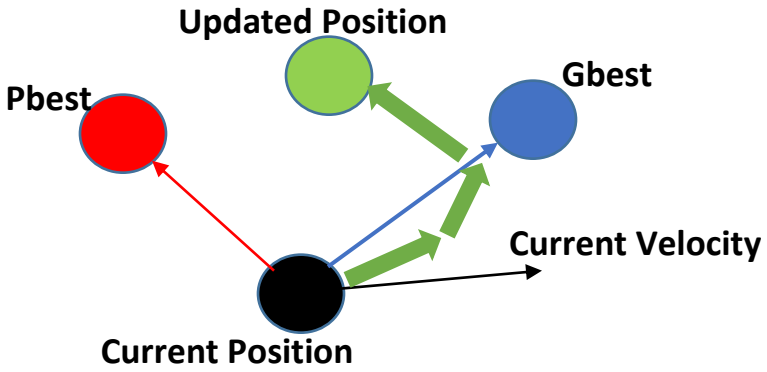

Figure 4. PSO Algorithm

The position X at t+1 can be formulated as:

$$X(t+1) = X(t) + V(t+1)$$

And subsequently velocity V at t+1 can be formulated as:

$$V(t+1) = wV(t) + c_1 r_1 (X_{Pbest} - X(t)) + c_2 r_2 (X_{Gbest} - X(t))$$

Where:

w: Initial weight.

c_1, c_2: Accelerating factor.

r_1, r_2: Uniformly distributed random number ∈ [0,1].

2.2 Ant Colony Optimization:

Ant Colony Optimization is an optimization technique which applies the concept of population-based meta heuristic [23]. It involves probabilistic techniques inspired from the behavior of real ant colonies. For this algorithm, problems must be considered in the form of path finding with a weighted graph. Hence, ACO finds the shortest path.

It starts with a group of ants leaving their nests and searching randomly for food. Whenever a particular ant finds food, they deposit pheromones on their way to the nest. The more the pheromones concentrated on the path, the more becomes the probability of it being followed. Since pheromones evaporate with time, longer paths are automatically discarded. Hence, different ants keep giving different path routes until we reach the shortest path where the pheromone trails have the strongest concentration. Almost all of the ants will then follow this path, reinforcing it further and hence all of them reach the food source.

4. Methodology:

We used CloudSim software to perform the simulation and extract the results. CloudSim is an open-source framework used for modelling and simulation of cloud computing infrastructures and services [24]. It is developed by the CLOUDS Lab organization and is written entirely in Java. It can successfully model and simulate infrastructures such as:

- Large-scale cloud computing datacenters
- Virtualized server hosts
- Application containers
- Energy-aware computational resources
- Datacenter network topologies

The major modal classes used in CloudSim simulation are:

- Cloudlet: Used to define a task to be scheduled in an environment.

- Vm: Used to define a virtual machine which will run the tasks in the simulation.
- Host: Used to define management of virtual machines.
- Datacenter: Used to define datacenter in the simulation.
- DatacenterBroker: Used to define as an entity which performs user actions.

The following figure 5 shows the main parts involved in CloudSim:

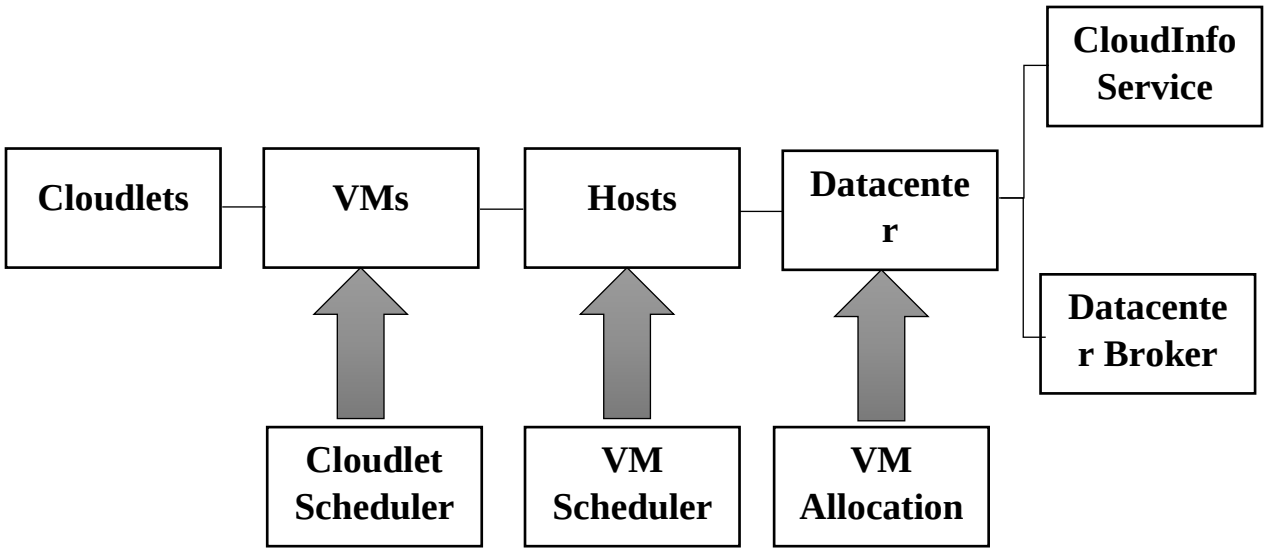

Figure 5. the main parts involved in CloudSim

The following table 4 shows meaning of the constants which used for cloud simulation and its used values in our simulation.

Parameters	Description	Value
POPULATION_SIZE	Number of particles for PSO Scheduler	25
NO_OF_ANTS	Number of ants for ACO Scheduler	4
NO_OF_GENERATIONS	Number of generations for ACO Scheduler	50
DATACENTER PARAMETERS		
NO_OF_DATACENTERS	Number of Datacenters	5
ARCHITECTURE	System Architecture	x86
OS	Operating System	Linux
TIME_ZONE	Time zone of the Datacenter	5.5
COST_PROCESS	Cost of using processing in the Datacenter	3
COST_MEMORY	Cost of using memory in the Datacenter	0.05

COST_STORAGE	Cost of using storage in the Datacenter	0.001
COST_BANDWIDTH	Cost of using bandwidth in the Datacenter	0.1
HOST PARAMETERS		
STORAGE	Storage size (in MB)	1000000 MB
HOST_MIPS	MIPS score for Host's Processing Element	1000
HOST_RAM	RAM size (in MB)	2048 MB
HOST_BANDWIDTH	Bandwidth of Host (in Gbps)	10000 Gbps
VIRTUAL MACHINE PARAMETERS		
NO_OF_VMS	Number of VMs	5
VM_IMAGE_SIZE	VM Image size (in MB)	10000 MB
VM_RAM	RAM size (in MB)	512 MB
VM_MIPS	MIPS capacity for VM's Processing Element	250
VM_BANDWIDTH	Bandwidth of VM (in Gbps)	1000 Gbps
VM_PES	Number of Processing Elements	1
CLOUDLET PARAMETERS		
FILE_SIZE	Input file size (in Bytes) before execution	3000 B
OUTPUT_SIZE	Output file size (in Bytes) after execution	3000 B
TASK_PES	Number of PEs required to execute	1

Table 4. Meaning of the constants which used for cloud simulation and its used values in our simulation

In every scheduler, these steps are followed for Scheduling and Comparison:

- Initialize CloudSim simulation with 3 parameters: number of cloud users (num_user), calendar instance with current date and time (calendar) and flag for tracing events (trace_flag).
- Create Datacenter(s) with the given parameters. Note that we have assigned only one Host and PE per Datacenter.

- Create a Datacenter Broker according to the scheduling algorithm.
- Create Virtual Machine(s) with the given parameters.
- Create Cloudlet(s) with the given parameters and length.
- Submit the VM and Cloudlet list to the Datacenter Broker.
- Start the CloudSim simulation.
- Once there are no more events to execute, stop the CloudSim simulation.
- From the broker, fetch the Cloudlet list after execution.
- This list contains details related to the execution of Cloudlets like Execution Start Time and Finish Time.
- The total finish time is calculated.

5. Results and discussion:

Simulations will be performed using CloudSim software using the parameters listed in Table 4. Five VMs will be used and 6 simulation scenarios will be applied. During the first scenario, we will adjust the number of tasks to 10, twice the number of virtual machines, and then the load on the cloud data center will be increased by increasing the tasks from five to 30 tasks, or six times the number of virtual machines, and this case represents a high load case on the cloud data center. The response time taken by each of the above algorithms to perform all the tasks will be calculated and then a comparison between these algorithms will be made.
The following table 5 shows the response time results obtained for each algorithm during each simulation scenario.

Algorithm / Task NO.	PSO	Ant Colony	SJF	RR	FCFS
10	3253.94	3924.32	5693.38	7947.02	3932.60
15	2674.78	4141.76	6978.94	8407.90	8289.36
20	4456.82	5946.68	6229.10	12367.06	8038.96
25	7076.98	8115.64	13061.30	12933.66	8757.92
30	4730.78	9702.24	10015.42	10699.82	10644.04

Table 5. The response time results obtained for each algorithm during each simulation scenario

The following figure 6 shows the comparison of the previous results:

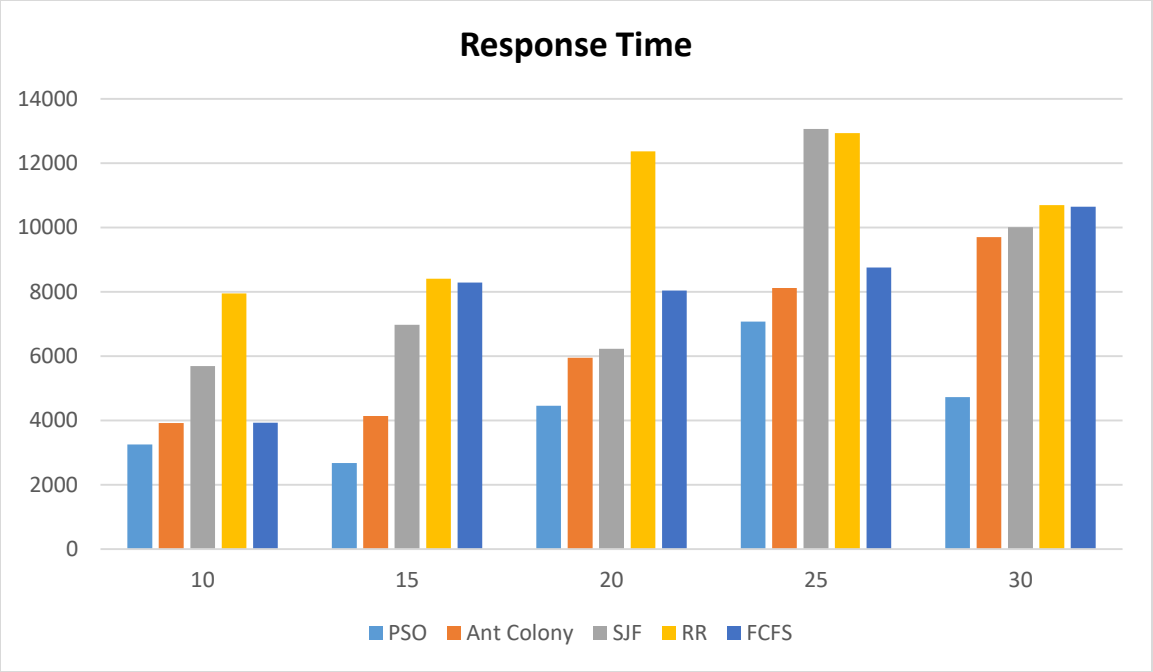

Figure 6. Comparison between algorithms

Let's analyze the response times for the different task scheduling algorithms in each scenario [25]:

First Scenario:

In this scenario, the PSO algorithm achieved the lowest response time, followed by the Ant colony algorithm. SJF and FCFS also performed reasonably well, while the RR algorithm had the highest response time among all the algorithms.

Second Scenario:

Similar to the first scenario, the PSO algorithm outperformed the others with the lowest response time. Ant colony and SJF algorithms followed, while RR and FCFS had relatively higher response times.

Third Scenario:

In this scenario, the PSO algorithm again performed better than the others. The Ant colony and SJF algorithms showed competitive results. However, the RR algorithm's response time increased significantly compared to the previous scenarios, and FCFS had a moderately high response time.

Fourth Scenario:

In this scenario, the PSO algorithm still had the lowest response time, while the SJF, RR, and FCFS algorithms showed higher response times. The Ant colony algorithm also performed well but had a higher response time compared to PSO.

Fifth Scenario:

The PSO algorithm remained consistent with relatively low response times. The Ant colony, SJF, RR, and FCFS algorithms all showed higher response times, with the Ant colony algorithm having the second-lowest response time in this scenario.

Overall, it appears that the PSO algorithm consistently provided the lowest response times across all scenarios. The performance of other algorithms varied depending on the number of tasks and virtual machines involved in each scenario. Ant colony and SJF algorithms generally performed reasonably well, while RR and FCFS algorithms seemed to struggle with larger task sets.

Keep in mind that the choice of the most suitable algorithm depends on various factors, and these results can help guide decision-making in cloud task scheduling based on the specific requirements and constraints of your application.

Let's compare the task scheduling algorithms based on the results obtained and discuss the pros and cons of each. We'll also explore the reasons behind the performance differences observed in the simulations.

1) Particle Swarm Optimization (PSO):

- Pros: PSO consistently achieved the lowest response times in all scenarios. It is a population-based optimization algorithm that is well-suited for continuous search spaces. PSO's ability to explore and exploit the search space efficiently likely contributed to its success in finding good solutions for task scheduling.
- Cons: PSO's convergence rate and performance heavily rely on its parameters and tuning. In some scenarios, it may struggle to find optimal solutions due to premature convergence or getting stuck in local minima.

2) Ant Colony Optimization:

- Pros: Ant colony optimization showed competitive performance in several scenarios, especially with smaller task sets. It is inspired by the foraging behavior of ants and effectively explores solutions through pheromone trails. In certain cases, it managed to outperform other algorithms.

- Cons: Ant colony optimization can be sensitive to parameter settings and may require fine-tuning for different problem instances. It might struggle when dealing with larger task sets or complex search spaces.

3) Shortest Job First (SJF):

- Pros: SJF performed reasonably well in some scenarios, especially with fewer tasks. It prioritizes shorter tasks, leading to reduced response times for such cases.
- Cons: SJF may suffer from "starvation" issues, where long tasks get delayed indefinitely if a continuous stream of short tasks keeps arriving. It is not suitable for all types of task distributions and may not perform optimally in scenarios with high variability in task lengths.

4) Round-Robin (RR):

- Pros: RR ensures fairness in task execution by giving each task equal time slices, which can be beneficial for certain scenarios with multiple users or time-sensitive tasks.
- Cons: RR's performance degraded significantly with larger task sets. It incurs higher overhead due to frequent context switches, and some tasks might experience longer waiting times, resulting in higher response times.

5) First-Come, First-Served (FCFS):

- Pros: FCFS is easy to implement and guarantees fairness based on the order of task arrivals. It can work well for small task sets with relatively homogeneous task lengths.
- Cons: FCFS can lead to higher response times, especially when long tasks arrive early. It is not suitable for handling varying task lengths efficiently and may not scale well with increased task complexity.

Reasons for the observed results:

- The PSO algorithm's strong performance can be attributed to its robust search capabilities and the ability to escape local optima, leading to finding near-optimal solutions for task scheduling problems.

- Ant colony optimization performed well due to its ability to explore the search space through pheromone-based communication, but its performance may have diminished as task sets increased due to the exponential growth in search space complexity.
- SJF showed competitive results with smaller task sets due to its preference for shorter tasks, but it struggled with larger sets and variations in task lengths.
- RR's context switching overhead and inability to prioritize tasks based on their characteristics led to higher response times, particularly with larger task sets.
- FCFS, while fair in terms of order of arrival, lacked adaptability to varying task lengths and struggled to handle more complex task scheduling scenarios.

Overall, selecting the most appropriate task scheduling algorithm depends on the specific characteristics of the workload, the number of tasks, and the complexity of the problem. It's essential to consider these factors and carefully tune the algorithms' parameters to achieve optimal performance in real-world cloud computing environments.

6. Conclusion:

In this research, we highlights the importance of selecting appropriate task scheduling algorithms in cloud computing to optimize response times and system efficiency. PSO proved to be the most effective algorithm among the tested ones, but the selection should be based on specific workload characteristics and problem requirements. Proper parameter tuning and algorithm adaptation to the problem domain can lead to improved performance and successful task scheduling in cloud computing environments. This comparative study offers valuable insights for cloud system designers and administrators in making informed decisions regarding task scheduling strategies.

Based on the obtained results from the simulations, the following conclusions can be drawn:

1) Particle Swarm Optimization (PSO) emerged as the top-performing algorithm in all scenarios, consistently achieving the lowest response times. Its ability to efficiently explore the search space and find near-optimal solutions makes it a promising choice for cloud task scheduling.

2) Ant Colony Optimization demonstrated competitive performance, especially with smaller task sets. However, its effectiveness diminished as the number of tasks increased, and it faced challenges in complex search spaces.
3) Shortest Job First (SJF) exhibited favorable results with fewer tasks, prioritizing shorter tasks and reducing response times. Nevertheless, it struggled to cope with larger task sets and variations in task lengths.

4) Round-Robin (RR) scheduling, although ensuring fairness, suffered from significantly higher response times as the workload increased. Frequent context switches and lack of task prioritization impacted its efficiency.
5) First-Come, First-Served (FCFS) scheduling, while simple and fair in task order, demonstrated relatively higher response times, particularly when dealing with long tasks arriving early.

7. References:

[1] Sadiku, M. N., Musa, S. M., & Momoh, O. D. (2014). Cloud computing: opportunities and challenges. *IEEE potentials*, *33*(1), 34-36.

[2] Ullah, A., & Nawi, N. M. (2021). An improved in tasks allocation system for virtual machines in cloud computing using HBAC algorithm. *Journal of Ambient Intelligence and Humanized Computing*, 1-14.

[3] BEN ALLA, S., BEN ALLA, H., Touhafi, A., & Ezzati, A. (2019). An efficient energy-aware tasks scheduling with deadline-constrained in cloud computing. *Computers*, *8*(2), 46.

[4] Mohammadzadeh, A., Masdari, M., & Gharehchopogh, F. S. (2021). Energy and cost-aware workflow scheduling in cloud computing data centers using a multi-objective optimization algorithm. *Journal of Network and Systems Management*, *29*, 1-34.

[5] Umam, M. S., Mustafid, M., & Suryono, S. (2022). A hybrid genetic algorithm and tabu search for minimizing makespan in flow shop scheduling problem. *Journal of King Saud University-Computer and Information Sciences*, *34*(9), 7459-7467.

[6] Devaraj, A. F. S., Elhoseny, M., Dhanasekaran, S., Lydia, E. L., & Shankar, K. (2020). Hybridization of firefly and improved multi-objective particle swarm

optimization algorithm for energy efficient load balancing in cloud computing environments. *Journal of Parallel and Distributed Computing*, *142*, 36-45.

[7] Yuan, H., & Zhou, M. (2020). Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems. *IEEE Transactions on Automation Science and Engineering*, *18*(3), 1277-1287.

[8] Baburao, D., Pavankumar, T., & Prabhu, C. S. R. (2021). Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method. *Applied Nanoscience*, 1-10.

[9] Muteeh, A., Sardaraz, M., & Tahir, M. (2021). MrLBA: multi-resource load balancing algorithm for cloud computing using ant colony optimization. *Cluster Computing*, *24*(4), 3135-3145.

[10] Yiqiu, F., Xia, X., & Junwei, G. (2019, March). Cloud computing task scheduling algorithm based on improved genetic algorithm. In *2019 IEEE 3rd information technology, networking, electronic and automation control conference (ITNEC)* (pp. 852-856). IEEE.

[11] Aladwani, T. (2020). Types of task scheduling algorithms in cloud computing environment. *Scheduling Problems-New Applications and Trends*.

[12] Bibu, G. D., & Nwankwo, G. C. (2019). Comparative analysis between first-come-first-serve (FCFS) and shortest-job-first (SJF) scheduling algorithms.

[13] Manasa, R., & Mabbu, D. (2023). A Comparative Approach on Scheduling Algorithms for Real time Systems. *Journal Electrical and Computer Experiences*, *1*(1), 29-35.

[14] Putra, T. D. (2020). Analysis of Preemptive Shortest Job First (SJF) Algorithm in CPU Scheduling. *International Journal of Advanced Research in Computer and Communication Engineering*, *9*(4), 41-45.

[15] Zouaoui, S., Boussaid, L., & Mtibaa, A. (2019). Priority based round robin (PBRR) CPU scheduling algorithm. *International Journal of Electrical & Computer Engineering (2088-8708)*, *9*(1).

[16] Panda, S. K., & Jana, P. K. (2019). An energy-efficient task scheduling algorithm for heterogeneous cloud computing systems. *Cluster Computing*, *22*(2), 509-527.

[17] Slowik, A., & Kwasnicka, H. (2020). Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, *32*, 12363-12379.

[18] Mohammadi, S., & Hejazi, S. R. (2023). Using particle swarm optimization and genetic algorithms for optimal control of non-linear fractional-order chaotic system of cancer cells. *Mathematics and Computers in Simulation*, *206*, 538-560.

[19] Xue, Y., Zhu, H., Liang, J., & Słowik, A. (2021). Adaptive crossover operator based multi-objective binary genetic algorithm for feature selection in classification. *Knowledge-Based Systems*, *227*, 107218.

[20] Islam, J., Vasant, P. M., Negash, B. M., & Watada, J. (2019, October). A modified crow search algorithm with niching technique for numerical optimization. In *2019 IEEE Student Conference on Research and Development (SCOReD)* (pp. 170-175). IEEE.

[21] Fan, S. K. S., & Jen, C. H. (2019). An enhanced partial search to particle swarm optimization for unconstrained optimization. *Mathematics*, *7*(4), 357.

[22] Zeng, N., Wang, Z., Liu, W., Zhang, H., Hone, K., & Liu, X. (2020). A dynamic neighborhood-based switching particle swarm optimization algorithm. *IEEE Transactions on Cybernetics*, *52*(9), 9290-9301.

[23] Ahuja, J., & Ratnoo, S. (2023, February). Population-Based Meta-heuristics for Feature Selection: A Multi-objective Perspective. In *Proceedings of International Conference on Data Science and Applications: ICDSA 2022, Volume 1* (pp. 243-264). Singapore: Springer Nature Singapore.

[24] Le, D. N., Pal, S., & Pattnaik, P. K. (2022). Cloudsim: A simulator for cloud computing environment. *Cloud Computing Solutions*.

[25] Zhou, Z., Li, F., Zhu, H., Xie, H., Abawajy, J. H., & Chowdhury, M. U. (2020). An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments. *Neural Computing and Applications*, *32*, 1531-1541.