Journal of Computer Science

Original Research Paper

Efficient Resource Allocation and Task Scheduling in a Cloud Computing Environment using Swarm Intelligence

Panuganti Hanumantha Rao, Rajakumar Subramanian and Geetha Soman

Department of Computer Science and Engineering, Dr. M.G.R Educational and Research Institute, Chennai, India

Article history
Received: 06-06-2024
Revised: 06-08-2024
Accepted: 09-08-2024

Corresponding Author:
Panuganti Hanumantha Rao
Department of Computer Science and Engineering, Dr. M.G.R Educational and Research Institute, Chennai, India
Email: hanumanthraovbit@gmail.com

Abstract: The new age of network-based computing, known as "cloud computing," is characterized by the distribution and sharing of resources over a network. These resources are available to anyone through the Internet on a pay-per-use basis. Any service that anybody uses can generate massive amounts of data. Therefore, in this scenario, there will be a significant cost associated with transferring data between two dependent resources. Furthermore, if not planned optimally, the overall cost of executing a complicated program could rise due to the application's high number of tasks. An effective allocation method is required to satisfy the ever-increasing demands for resources. Cloud computing has been the focus of extensive research. Present methods aim aiming dynamic resource allocation but are not cost-effective. In light of these issues, this article proposes a heuristic scheduling technique “Enhanced Cat Swarm Optimization” ECSO method to distribute application tasks among available resources, based on Cat Swarm Optimisation (CSO). The foraging nature of cats has served as inspiration for several resource allocations, one of which is Cat Swarm Optimisation (CSO). The proposed novel approach ECSO offers a modification to CSO that adds a crossover mechanism (Uniform crossover) to minimize the total execution cost. To find the optimal solution, the proposed ECSO method takes into account the cost of data transmission between dependent resources as well as the cost of job execution on different resources. The ECSO method is tested with a made-up workflow and evaluates how well it performs in comparison to the state-of-the-art CSO, PSO, and BCO algorithms for scheduling tasks. The experimental findings demonstrate that the proposed ECSO provides a total cost-minimizing task to resources. The ECSO outperformed existing CSOs, PSOs, and BCOs concerning total execution time of 8% lower and execution cost of 4% less. It also guarantees that the available resources are fairly distributed**.**

Keywords: Cloud Computing, Resource Allocation, Swarm Optimization, Uniform Crossover, Total Execution Cost

Introduction

The latest computing technology, cloud computing, offers secure, pay-per-use environments. According to Rajkumar *et al.* (2009), cloud computing saves organizations money by using virtual resources instead of physical servers and equipment. Cloud computing transforms into a more effective system for computing resources through the provision of multilevel abstraction and a succession of virtualization layers. Users can expand or decrease the amount of services available through the elastic scale-up and scale-down properties of cloud services. The fundamental concept is to store information, assets, and services in an abstract form on the Internet. The resources, both software and hardware, are available to users upon demand based on their present needs, whereas the supplier organizations distribute them based on their current availability.

Many sectors of society, including governance, business, and the economy, have been impacted by cloud computing in recent years. As a form of parallel and distributed computing, "the cloud" refers to a network of interconnected remote servers that can store and process data centrally and provide users with online access to various IT services and resources (Pragati *et al*., 2017). One way to lower software expenses is by using cloud computing, where the customer

leases resources instead of buying them (Sreeram *et al*., 2021). Cloud computing also offers on-demand services; which customers can access from anywhere. By leveraging the resources of remote computers, cloud computing eliminates the need to store and retrieve data from a single location (Kesavaraja and Shenbagavalli, 2018). Clients rely on cloud services instead of maintaining their infrastructure, which frees consumers from having to understand the internal operations of the network (Zeebaree *et al*., 2020). Software as a Service (SaaS), Infrastructure as a Service (IaaS), and Platform as a Service (PaaS) are the three ways in which virtualized computer resources can be provided to clients (Salah Farrag *et al*., 2015).

The services are illustrated in Fig. (1). Infrastructure as a Service (IaaS) is in the role of providing virtualized compute resources, such as memory, CPU, servers, storage, and more, as a service. Some instances of infrastructure as a service are AWS, Google Cloud platform, Apple iCloud, Google Drive, and Google Compute Engine (Meduri *et al*., 2023). Platform as a Service (PaaS) offers services such as operating systems and software development frameworks. One example of a PaaS platform is Google App Engine, which allows programmers to build, test, run, and manage apps (Ahmed and Wafaa, 2017). Software as a Service (SaaS) eliminates the need for customers to download, install, and execute applications on their computers; instead, the user simply accesses the application's interface. Google Apps, Cisco WebEx, and Salesforce are some instances of software as a service (Naji *et al*., 2020). When it comes to the demand forecast schedule, the Application Service Providers check in on the rental services at regular intervals, decide how to best allocate goals and resources, and save money by not paying for unnecessary calculations, storage, or data transfers (Zebari *et al*., 2020). Also, instead of demanding complicated bids, resource distribution should be proportional to decentralization (Sadeeq *et al*., 2018). This is because providers, reliant on their resources, may add further complexity by providing services of different kinds or a combination of different types. Multiple customers can compete for similar resources because suppliers require submissions from suppliers, consumers submit to customers, and relevant resources are accessible from various sources (Mostafa *et al*., 2020). When most individuals think about cloud computing, they probably envision issues with data security, power, service availability, managing memory expansion, and job planning.

Cloud computing research, on the other hand, tends to center on planning tasks. To make the most of everything that the cloud has to offer, many tasks require lightning-fast processing speeds, minimal latency, and ample resources. The various roles played by the allocation plan make it imperative that tasks be assigned appropriately.

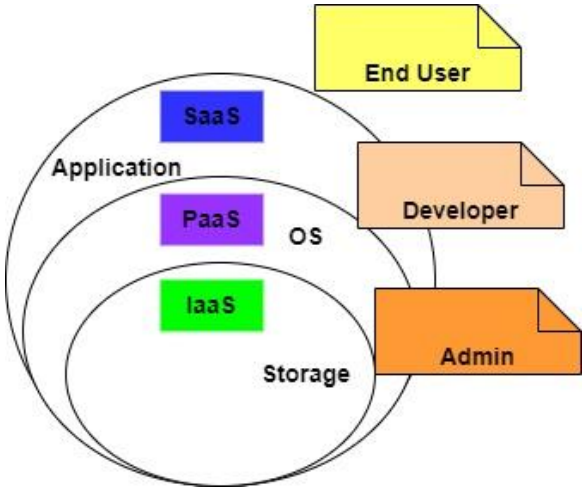

Fig. 1: Services of cloud

There are two primary objectives for the cloud database provider: (1) Satisfying the client's SLAs and (2) Increasing their profit margin. The service provider's ability to accomplish these objectives is dependent on their resource management skills. In order to keep up with competing requests, the service provider needs to wisely distribute scarce resources like CPU and memory. Some additional resources, however, do not have a hard cap but do come with a price. As an example, consider database replication. When you expand your database's replicas, you'll incur costs for both the initial setup (like adding more nodes) and ongoing operations (like synchronization). In addition, companies are using a variety of resources to enhance their services further, with the hope that more clients will subscribe to cloud computing (Zaki and Saad, 2018). Therefore, SLA and resource allocation (Saleh *et al*., 2018), which indicate the amount of customer contentment, are among the most critical elements that impact service quality.

This study concentrates on scheduling algorithms to check for availability and allocate resources accordingly. There will always be a need for an improved resource scheduling algorithm due to the modern trend of ever-increasing resource demands. Heuristic swarm optimization has been the most widely used algorithm for solving the aforementioned problem, and it initially produced outstanding results. The key contribution of this article is to address resource constraints in the cloud. Many researchers suggest scheduling user tasks using the advanced cat optimization algorithm.

Related Work

A significant amount of resources is needed by the data center "cloud providers" are independent third parties that oversee and manage the supply of computing resources, such as hardware, software, and platforms, to users on demand. Adapting your strategy for resource allocation to

meet the varying demands of your clients can be quite a challenge. As a result, many researchers have investigated cloud resource allocation through the use of different scheduling algorithms, resource allocation strategies, RAS, and VM allocation methods. With the hope of enhancing network performance by making use of cloud resources.

Virtualization was suggested by Tenepalli and Appini (2014) as a means to offer active cloud resource allocation. By moving Virtual Machines (VMs) from a busy server (the "hot spot") to an idle one (the "cold spot"), the offered method allows for the efficient distribution of numerous virtual resources. They calculated server resource utilization using the "skewness" approach, which allows them to anticipate future resource needs by looking at usage logs of previously used resources. By combining green computing with dynamic resource allocation, this method can conserve server energy by reducing wasteful server consumption and optimizing burdens to meet virtual machine needs regarding server capacity.

Methods for allocating central processing units and network resources in shared hosting environments are presented by Urgaonkar *et al.* (2002). A linear system under control and offline parameter identification were the underlying assumptions of most of the earlier investigations (Gandhi *et al*., 2002). For different types of requests, Lu *et al*. dynamically change the cache size (Ying *et al*., 2004). The DBMIN technique, developed by Hong-Tai and David (1998). is used to control the amount of data stored in a relational database's buffer pool (Chou and David). It makes sense to undertake dynamic power allocation if we think of power as just another resource in the system. Energy and server resource management in data centers is crucial, according to Jeffrey *et al*. (2001). In addition, regulates power consumption and application-level performance (Xu *et al*., 2010). To improve the workflow's execution time, Gurmeet Singh and others suggest a task grouping approach. Reducing some of the time spent waiting in queue is how it's done (Singh *et al*., 2008). Another approach to dynamically scheduling several workflows is a planner-guided strategy. With the suggested method, performance improves with increasing numbers of concurrent workflows by dynamically scheduling each task in each workflow (Navjot *et al.*, 2011). The inflexibility of above above-discussed scheduled algorithms and the effort required to arrange and develop a timeline are two major pitfalls. In contrast to heuristic approaches, which focus on a single answer, genetic algorithms consider a population of options while making decisions, making them ideal for production scheduling challenges. To optimize Quality of Service (QoS), deadline, and budget, a straightforward Genetic Algorithm (GA) is suggested. This method discovers the ideal solution in polynomial time. To discover the best solution, this algorithm uses a heterogeneous and reservation-based service-oriented setup (Yu and Buyya, 2006).

Many studies have investigated the possibility of using swarm intelligence to allocate cloud computing resources. Particle Swarm Optimisation (PSO) (Sreeram *et al*., 2021) is the most studied and long-standing swarm algorithm; it takes its cues from the way fish and birds forage. Artificial life and swarming theory are identified as the two theoretical foundations around which PSO was built.

The Ant Colony Optimisation Algorithm (Warneke and Kao, 2011) is another algorithm that takes inspiration from ant behavior, specifically their biological foraging strategy of leaving pheromones behind to guide future ant colonies. Cloud load balancing using Ant Colony Optimisation (ACO) was suggested by Basha *et al*. (2017) as an elastic and dynamic approach. In the bio-enthused technique, which was suggested, the artificial algorithm is trained to mimic the behavior of biological ants by adding and removing certain characteristics. Ants utilize trail pheromones to mark the journey from food sources to their nests while they look for food. Ants locate pheromone trails when they forage, and they adhere to the ones that have the greatest concentration of pheromone deposits (the shortest path).

Optimization of Bee Colonies (BCO) (Son *et al*., 2013) is another methodology taken into consideration. An artificial colony of bees has also been defined using an algorithm as a multi-agent system. Research into the reasoning behind honeybees' nectar harvesting technique formed the basis of this theory. The different NPC challenges are effectively addressed by a Bee Colony Optimization (BCO) method that is suggested for simple application task/job scheduling. To decrease the time, it takes to complete a work or task, this method strives for a fair distribution of the load across the dispersed resources. According to the experimental findings of this suggested strategy, BCO outperforms GA in terms of the time it takes to complete tasks (Bitam, 2012).

An activity-based scheduling technique is suggested for tasks that take into consideration a collection of resources that are going to be used (Qi *et al*., 2009). Workload distribution and job scheduling are addressed using a Multiple Ant Colony Optimization algorithm (MACO). The basic premise is that ant colonies can discover the optimal value in the search space on a global scale by exchanging information about their best values. Cooperation between them allows for the faster resolution of an issue with many objectives. Performance is higher compared to FCFS and ACS methods in the experiments (Liang *et al*., 2010).

The scientific process scheduling in the cloud can be improved with the help of a particle swarm optimization technique. The goal of the author is to ensure that all

resources are evenly distributed and that the overall cost of workflow calculation is minimized. The basic idea is that a group of particles (the population in PSO) will begin to seek the solution space, and as they do so, they will be influenced by each other to find the optimal position. Particle velocities and states are optimized at the local level in each iteration to reach the global optimal point. In comparison to the state-of-the-art Best Resources Selection (BRS) algorithm, this algorithm demonstrates superior performance. A discrete PSO is an improved version of the basic PSO that achieves better results concerning the makespan and cost optimization ratio (Wu *et al*., 2010). When compared to PSO, GA, ACO, BCO, etc. all fall short in terms of makespan. This is because PSO converges among swarm algorithms at a good pace and reaches the local optima at a far faster rate than evolutionary algorithms. When the application size is huge, iteratively obtaining the optimal answers becomes more of a hassle with PSO and all other methods.

The authors propose using CSO to schedule workflow tasks as a solution. While CSO is quite similar to basic PSO, it differs in a few key ways that make it superior. People who are CSOs can switch between two states: Seeking and tracing. When cats are in searching mode, they do not move at all. Only the next best position is their goal. However, when in tracking mode, they swiftly go to the next optimal place. In other words, not every cat is hopping around in solution space simultaneously. They can discover the optimal next position with relative ease due to the searching mode, which decreases the number of repetitions necessary to get a solution (Chu and Tsai, 2007).

Pei-Wei *et al*. (2008) created the Parallel Cat Swarm Optimisation (PCSO) Algorithm after the standard CSO was introduced; it enhanced the convergence speed of CSO for tiny population sizes. Binary optimization of CSO (BCSO), first proposed by Sharafi *et al*. (2013), was applied to various benchmark optimization tasks and the zero-one knapsack issue. It was determined that this method outperformed competing binary optimization techniques. To improve the algorithm's searching mode step, an alternate method is to employ different chaotic maps. The best options were logistic and sinusoidal maps. Sharafi *et al*. (2013) came up with this and termed it the Chaotic Cat Swarm Algorithm (CCSA). Beyond that, the Harmonious-CSO (HCSO) was introduced by Kuan *et al*. (2014). This approach modified the searching mode formula by including the idea of the Hirschberg-Sinclair algorithm. The algorithm's performance was verified using a Support Vector Machine (SVM), which outperformed CSO in the experimental findings. Wang (2015) significantly improved CSO by dynamically modifying the parameter Mixture Ratio (MR).

The incorporation of crossover approaches is one way that swarm optimization algorithms have been enhanced to produce better outcomes. The enhanced technique was suggested by Sharma *et al*. (2018) who enhanced the Particle Swarm Optimisation Algorithm by adding a crossover operator. A comprehensive overview of the several genetic algorithm crossover operators was given by Padmavathi and Priyanka (2017).

The summary of the survey carried out is presented in the below Table (1).

Table 1: Analysis of existing literature

Ref	Author	Approach	Strength	Weakness	Scope in Proposed Method
Urgaonkar *et al*., 2002	Urgaonkar *et al*.	This experiment assumed a linear system under control and offline parameter identification	Depends on topological changes	Difficult to identify offline parameters	Task sequencing
Sharma *et al*., 2018	Sharma *et al*.	Enhanced particle swarm optimization (PSO)	Incorporated crossover operator to particle swarm optimisation algorithm	Genetic algorithm	Crossover operator to decide the optimal solution
Jeffrey *et al*., 2001	Jeffrey *et al*.	Dynamic power allocation	Energy efficiency	Virtual machine allocation is not done according to the requirement	Dynamically allocating the resources
Basha *et al*., 2017	Basha *et al*.	Ant Colony Optimisation (ACO)	Elastic and dynamic approach	Not optimal	Chose shortest path
Son *et al*., 2013	Son *et al*.	Optimization of Bee Colonies (BCO)	Nectar harvesting technique	Exceeds deadlines in some cases	Strives for a fair distribution of the load across the dispersed resources
Pei-Wei *et al*., 2008	Pei-Wei *et al*.	Parallel Cat Swarm Optimisation (PCSO) algorithm	Enhanced the convergence speed of CSO for tiny population sizes	More execution time	Extension to the Cat Swarm Optimisation (CSO)

This study aims to create a system for dynamically allocating resources and scheduling tasks using cat swarm optimization. To maximize throughput while minimizing predicted total make span through optimum scheduling using cat swarm intelligence.

Background

In 1995, PSO was initially presented by Kennedy and Eberhart (1995). For non-linear function optimization, they offer PSO, a population-based stochastic optimization heuristic algorithm. Several social models were simulated to produce it. A swarm is an association of many particles. While PSO is comparable to the Genetic Algorithm (GA), it differs in a few key respects, such as the absence of mutation and crossover operators. PSO's low convergence rate and ability to swiftly achieve local optimal solutions have made it a popular choice for optimization problems in several domains. A worldwide optimal solution can thus be found in less time (Gil *et al*., 2007).

One optimization strategy that uses PSO, a soft computing methodology, is to improve its local candidate solutions about the fitness function in each iteration until it finds the global best solution. Every one of the particles that make up the PSO population has its unique speed and location in the solution space/search space. To discover the best possible solution in the search space, the social behavior of the particles affects their relative positions and velocities.

On the other hand, chu and Tsai's Cat Swarm Optimisation (CSO) technique is more efficient in computing terms (Sharma *et al*., 2018). A sort of swarm intelligence optimization algorithm, it takes its cues from the way cats hunt for food, as the name implies. The majority of cats are in seeking mode, where they look for the best nearby prey, while the minority are in tracing mode, where they follow their prey. After the seeking mode ends, the normal CSO algorithm disregards other cats and immediately selects potential prey for tracing. The present second-best cat, however, has the potential to surpass all others and become the best in the future. Consequently, this study introduces Assorted Cat Swarm Optimisation (ACSO), a variant of CSO that integrates the aforementioned improvements. A new heuristic algorithm for optimization called "cat swarm optimization" takes its cues from the cooperative nature of feline social networks. The idea of CSO was born out of studying two feline behaviours: Seeking and tracing. So, seeking mode and tracing mode are the two components of CSO. The trials and results demonstrate that CSO outperforms PSO with a weighting factor in terms of performance and results. However, in general, PSO with a weighting factor exhibits greater performance than plain PSO. The authors centered their attention on PSO and CSO to minimize total cost and improve performance. Both PSO and CSO were used to obtain the minimal cost as an optimized mapping technique, but CSO was found to achieve the outcome in the fewest iterations, which was the authors' goal.

Further, in this article, Cat swarm optimization is extended with crossover operators to select the best optimal solution for resource allocation and task scheduling. This article presents Enhanced Cat Swarm Optimization ECSO, a novel approach based on cat swarm intelligence and the adoption of crossover operators for choosing the best optimal solution for task and resource scheduling.

This study's primary goals are as follows: (A) To offer an ECSO workflow scheduling that provides an optimal scheduling method. (B) While scheduling workflows, the suggested algorithm is contrasted with existing CSO, PSO, and BCO algorithms. (C) The analysis of resource load balancing is performed.

Motivation

The significance of cloud computing is growing at an exponential rate, making it evident that it is the most recent and greatest technology. Cloud computing has been, or is rapidly becoming, an integral part of any company's operations.

Given the heavy usage, keeping the computation as rapid and effective as possible is of the utmost importance. To do this, one must determine the optimal order in which to distribute the jobs to the available virtual machines. Although numerous swarm intelligence algorithms can handle this, they fail to take into account a significant factor. In several cases, the solution or order task allocation to Virtual Machines (VMs) that seems to be the worst option at the moment ends up being the best option. However, the swarm intelligence algorithm only took the best ones into account for further processing. This led researchers to conclude that a better final solution can be achieved by combining the best and second-best options. As one of the latest and most effective swarm intelligence algorithms, this method was considered and integrated into CSO to establish a sequence for allocating cloud computing resources.

Biological Basis

The instincts of cats to seek out and gather prey are the inspiration for CSO. At its core, CSO is based on cat swarm intelligence, which describes how swarms of cats communicate with one another and their surroundings. During their foraging activities, cats are seen either actively seeking out prey or tracking its scent. While working on the CSO algorithm, Chu and Tsai came up with these phrases. Chu and Tsai presented Cat Swarm Optimisation (CSO), a novel swarm-based adaptive method, in 2007. The social behavior of cats serves as an inspiration for it.

There are two distinct behavioural states that cats can go into (1) Seeking and (2) Tracing.

Seeking mode: Although cats spend much of their time lounging around, they are quite alert and are always looking for food. We call this "seeking mode." They don't have speed, only state, because they simply remain in one place and sense the best course of action.

Tracing mode: As soon as it spots its prey, the cat goes into tracking mode to get after it. In tracing mode, cats behave similarly to hunters by swiftly repositioning themselves to be in the best possible position.

Materials and Methods

It is always a vital and challenging effort to represent and schedule all the processes in a data-intensive application workflow. As previously mentioned, the data generated by these intensive applications is quite vast. Consequently, processing and transferring this data across jobs will incur a substantial cost. As a result, the authors have narrowed in on these hefty expenses and are working to reduce them while simultaneously improving performance compared to the current CSO in terms of iterations. Figure (2) shows a sample workflow that was taken for experiments. It consists of 9 jobs that need to be scheduled onto these 4 resources. It is assumed that the magnitude of the data flow between Ti and Tj, represented by di,j, remains constant for all activities.

The authors of this study provide a customized algorithm that, using the CSO concept as its foundation, seeks to minimize the overall cost of task execution in a cloud environment by allocating resources efficiently. An initial population of N cats is used by the proposed method; while some of these cats are actively searching, others are more focused on tracing their prey, according to MR. Depending on the mode a cat is in, a task-resource mapping may be updated. By evaluating the cats' fitness level, determine the least expensive mapping can be determined. Every time around, a new set of cats are picked to use as our trading partners. As a result of maximizing efficiency, the optimal solution represented by the best cat position provides the lowest-cost mapping possible. The present second-best cat may eventually surpass and become the best of its kind, as stated aforementioned. As a result, the suggested method is an adaptation of CSO that finally gives due consideration to the second-best cat. Seeking and tracing are the two main models that contain the whole algorithm. Additionally, the standard CSO incorporates the suggested adjustment of several crossovers. The entire process is depicted in Fig. (3).

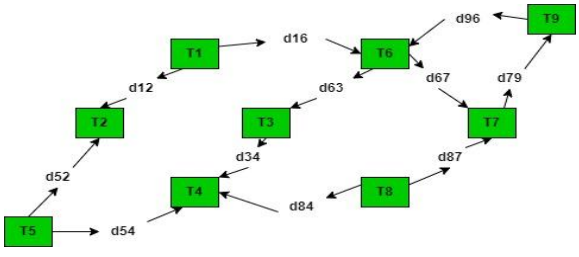

Fig. 2: Sample workflow

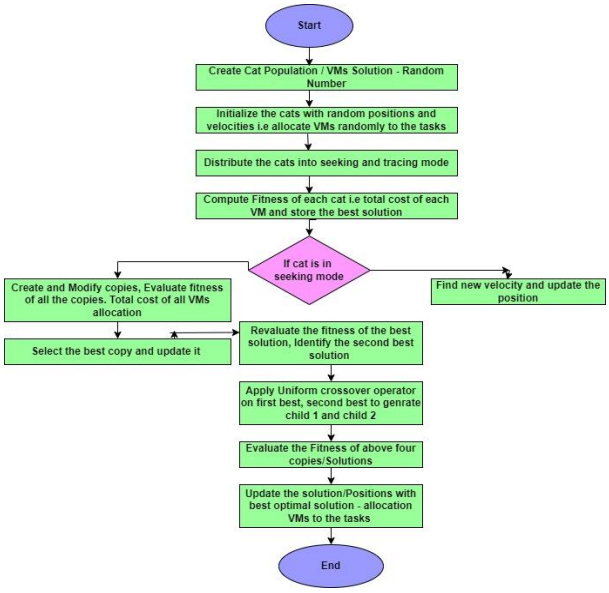

Fig. 3: Proposed method process flow initial steps

The proposed method ECSO can be explained as:

1. Allocate the VMs to the tasks that are waiting randomly
2. Divide the tasks into waiting and executing classes
3. Calculate the cost of VMs for present allocation. This will be the first best optimal allocation
4. Calculate the cost of VMs for task allocation which are in waiting. This could be the second-best optimal allocation
5. Then apply a uniform crossover operator on the first and second best solutions to get the final best solution
6. Accordingly, apply the best solution for the allocation of tasks to the VMs

Initial Steps

The whole process is performed on a single data center with numerous hosts, each with multiple Virtual Machines (VMs) (i.e. 4 in our example). One representation of an m-dimensional solution is a cat. For a cat, each dimension reflects its coordinate value. In cloud computing, the cat represents a solution vector with a length equal to the number of jobs (i.e., 10 in our example). The vector indexes indicate tasks, with the value at each index indicating the VM allocated to that job. Tasks are completed on the assigned VM. If two jobs share the same VM, they are executed sequentially. The lower-indexed job is given the Virtual Machine (VM) first because it is considered to have arrived earlier. Taking into account the burst time of each task in this sequence allows us to estimate the entire execution time.

Initially, the quantity of cats is manually set. A manually chosen value is used to begin the process of determining the total number of cats. In other words, it

serves as a representation of the solution size space, which is the number of possible solutions. By the nature of cats that was stated earlier, the solution space has been partitioned into two subgroups, with one subgroup undergoing the searching mode and the other subgroup executing the tracing mode for that specific iteration. This divide is carried out at the start of each iteration to ensure that every prospective solution has the opportunity to participate in both modes randomly. To make this difference, a variable known as the "Mixture Ratio (MR)" is utilized. The number of cats (solutions) in the tracing mode divided by the number of cats in the seeking mode is the definition of this variable. A smaller number of cats are in tracing mode, which represents how a single cat would likely spend most of its time hunting. To reduce the number of cats in this mode, MR is usually given a moderate value between 0 and 1.

Fitness Function

It is possible to determine the fitness value of any possible solution, as well as any temporary solutions generated during implementation, by using the fitness function. The extent to which the solution fits our situation is indicated by it. With each iteration, we have minimized the maximum fitness value using a minimization strategy.

The fitness value in the suggested algorithm is the total of the execution and transfer costs. The virtual machine's execution cost, ECi, is calculated as follows:

$$ECi = TaskDataUnitsExeci \times VmDataExeCosti \quad (1)$$

where, $VmDataExeCosti$ stands for the execution cost per unit of data and $TaskDataUnitsExeci$ is the total number of data units required by the task to be done on V mi. Vmi's transfer cost, Ti, can be expressed as:

$$TFi = \sum_{ij}(DTij \times VmTransCosti,j) \quad (2)$$

where, j represents each possible Vm to which data can be sent from $V\ mi\ to\ V\ mj$.

And, DTij represents the data units transferred from $VMi\ to\ VMj$.

$VmTransCosti,j$ represents $Vm's$ transmission cost from $VMi\ to\ VMj$.

The total cost, $Costi$, is the sum of execution ECi and transfer costs TFi:

$$Costi = ECi + TFi \quad (3)$$

This suggested approach ECSO incorporates the existing second-best solution to minimise the total cost that was described above. Using the minimization technique, the fitness value is calculated here based on the expenditures incurred. Results for fitness evaluations will be more conservative when expenses are reduced. Assigning jobs to different Virtual Machines (VMs) on the hosts in the cloud using this technique will result in extremely low costs, according to the final solution calculated after several iterations.

Different cloud service providers offer different pricing strategies that can be used to estimate the cost of their services in advance. Consider Amazon and Gogrid, both of which have distinct pricing structures for various customer segments. The Amazon Web Services (AWS) cost calculator allows users in Amazon to evaluate their total service costs.

Seeking Mode

Most cats search the globe while resting using elegant position updates. The method employs two fundamental factors: CDC and SMP. SMP (seeking memory pool) indicates the number of copies per cat. CDC (count of dimension to change) determines the number of assignments to be modified in a single copy. This state is also known as the resting state. There is an initial period of inactivity for the cat in this state. It performs little more than examine its environment for threats and possible prey. Similarly, to get closer to the ultimate solution, the current seeking-mode possible solutions in the solution space are changed to locate the ideal next potential solution. Making a copy of the cat's current position (i.e., the configuration of virtual machine allocation to the tasks indicated by the indices of this possible solution) and slightly modifying each copy to check neighboring locations allows us to make this option. It is necessary to initialize the following parameters:

- Seeking memory pool: The Seeking Memory Pool (SMP) is the set of all possible solutions multiplied by the number of copies made before choosing the best one. It specifies the maximum allowable dimensional positional change for each cat while copying and is called the
- Seeking Range of the chosen Dimension (SRD): Specifically, the margin of error that can be used to adjust the index of the Virtual Machine (VM) assigned to each job to generate a new solution
- Counts of Dimension to Change CDC: Every solution dimension remains unchanged in every iteration, according to the CDC (Counts of Dimension to Change) method. The assigned Virtual Machine (VM) for each job remains unchanged. Therefore, the number of changed dimensions is specified by the CDC
- Self-Positioning Consideration (SPC): The present virtual machine job allocation might be taken into account more or less. The value of the boolean variable SPC dictates this choice

Procedure-ESCO-Seeking Mode

Step 1: Produce j identical copies of the i^{-th} cat using SMP:

$$j = \begin{cases} SMP - 1 ; SPC = 1 \\ SMP ; otherwise \end{cases} \quad (4)$$

Step 2: Randomly change the copy CDC dimensions:

$$Y = (1 \pm SRD \times K) \times X \quad (5)$$

X denotes the current position, Y indicates the next position, and K is a randomly produced value between 0 and 1, in this case.

Step 3: Assess each copy's fitness.

Step 4: Select the most cost-effective solution from all copies.

The probability of $cati$ is determined by:

$$PRi = \frac{FTi - FTmax}{FTmax - FTmin} \quad (6)$$

Throughout this context, PRi denotes the $cati$ probability, FTi denotes the $cati$ fitness value, $FTmax$ denotes the upper bound of the fitness function, and $FTmin$ describes the lower bound.

If each cat's fitness score is exactly 1, then the preceding selection probability is 1.

Step 5: Pick a solution at random and swap it out for the i^{-th} cat.

Uniform Crossover

The former cat swarm optimization algorithm's seeking mode culminated with the cats pursuing the cat allocated to the global best, which served as prey in the tracing mode. It has been noted, however, that the best solution is not always the most globalized one. A better pick may be the second-best cat prospect on occasion. This becomes a crucial aspect of the CSO algorithm to be disregarded.

A genetic algorithm's crossover operator was integrated to embrace the second-best cat position as well, and the crossovers of the global best and second-best cat positions were additionally taken into account. Two crossover operators were tested. Two-point crossover and uniform crossover are used in this context. The results were recorded, plotted, and compared. Later, the results are discussed in the Performance section. We found that the uniform crossover operator yielded the best results. So, this algorithm uses it for better outcomes.

The proposed method incorporates a uniform crossover operator in the CSO algorithm and selects the optimal solution for task and resource scheduling.

Procedure-ESCO-Uniform Crossover Operator

The new algorithm's implementation of this section follows these four simple steps:

***Algorithm*:** ESCO_Uniform Crossover Operator *Begin*

1. *Assign the global best-fitting cat as Parent1 using the most recent changes to the positions and velocities of the cats (the solutions and future modifications to the virtual machine allocation of each job, respectively). Locate the cat that ranks second in terms of fitness and designate it as Parent2.*
2. *Create two solutions, Child1 and Child2, by applying the Uniform crossover operator to Parent1 and Parent2.*
3. *Keep track of Child1's and Child2's fitness values.*
4. *Pick the best fitness candidate from among Parent1, Parent2, Child1, and Child2 for global best cat and update its value.*
5. ***End***

The second-best possible sequence for virtual machines' work allocation is also taken into consideration here. By selecting the best option from the set consisting of the best, second best, and their generated descendants, the best possible option for this configuration is updated. This global best is now being considered as a possible solution to be followed at the tracing phase.

Tracing Mode

The cat begins to follow its prey as it spots it. Before pursuing its prey, it determines its velocity, which includes both its speed and direction. This triggers an abrupt transition from a state of sedentary rest to one of vigorous activity. Cats in this state are energetically searching the local space by moving toward the next best position quickly and efficiently. Within the framework of cloud-computing resource allocation, this comparison implies that after obtaining the optimal sequence of Virtual Machines (VMs) to assign to tasks, potential solutions may, while in tracing mode, determine the difference between the index of the present VM assigned to a task and the optimal VM to be assigned to it (i.e., compute its velocity) and swap the indexes accordingly. This is done for all the tasks. This means that the global best solution is used to update the index of virtual machines kept at each task of a solution.

Here is a general outline of the steps:

1. The following formula is used to update the velocity. Present velocity:

$$vi1 = wt.vi + p.c.(Xbest - Xi) \quad (7)$$

where, c is the acceleration constant, wt is the inertia weight, and p is a random number such that $0 <= p <= 1$. The present position is denoted as Xi, the best location is $Xbest$, and the previous velocity is vi:

2. Assign the upper bound to the cat's velocity if its updated velocity is greater than it

3. Using the current position as input and the revised velocity, we can update the cat's location using the following formula:

$$Xi1 = Xi + vi1 \quad (8)$$

where, Xi is the current position, Xi 1 is the new location and vi1 is the updated velocity:

4. Determine the cats' fitness level
5. Incorporate the solution set with the present iteration's best positions

Procedure for Enhanced CSO-ECSO

Procedure-ESCO

***Algorithm*:** ESCO *Begin*

1. *Accept user input for parameters.*
2. *Create the initial cat population by randomly allocating location (X) and velocity (V) vectors for each dimension.*
3. *Initiate MR/SPC.*
4. *Repeat steps 5-7 until culmination requirements are met.*
5. *Place modes (seeking or tracing) for all cats per MR.*
   Evaluate cat fitness and determine the global best.
6. *For each cat:*
   a) *If the cat is in seeking mode, execute seeking mode. Perform various Crossover Optimised local best selection.*
   b) *For cats in tracing mode, execute tracing mode.*
7. ***End***

Simulation Setup

The Cloudsim simulator is used to simulate the algorithm that is being suggested. By facilitating the creation of Virtual Machines (VMs) in data centers, each with its own processing capability, the Cloudsim toolkit (Yu and Shi, 2008) allows for the modeling of systems of cloud computing environments. This can be used to simulate jobs, which are cloudlets, and then allocate them to different virtual machines. The values that were initialized for the simulation are shown in Table (2).

Experimental Data

The data produced during simulation is presented in Tables (3-4). The cost matrix is presented in Table (3). It represents the execution cost of VMs to the tasks. Table (4) shows the transfer costs between VMs.

$$EM[i,j] = Execution\ cost\ of\ Ti\ on\ PCi\ in\ cents$$

$$TM[ij\ Transmission\ cost\ from\ PC_i\ to\ PC_j\ in\ cents/MB$$

Table 2: Simulation configuration

No. of datacenters	1
RAM	2560 MB
Host memory	1000000 MB
Bandwidth	10000
No virtual machines	6
No of hosts	4
SMP	5
SRD	0.8
CDC	5
MR	Random Value
SPC	Random Value

Table 3: Execution cost computation

	PC1	PC2	PC3	PC4
T1	1.24	1.10	1.12	1.13
T2	1.12	1.10	1.10	1.14
T3	1.23	1.11	1.15	1.16
T4	1.19	1.14	1.22	1.15
T5	1.24	1.15	1.16	1.15
T6	1.14	1.13	1.12	1.14
T7	1.24	1.13	1.16	1.17
T8	1.10	1.12	1.14	1.16
T9	1.23	1.14	1.15	1.18
T10	1.15	1.14	1.26	1.13

Table 4: Transfer cost matrix

	PC1	PC2	PC3	PC4
PC1	0.01	0	0.15	0.19
PC2	0.15	0.15	0	0.20
PC3	0.19	0.19	0.20	0
PC4	0.20	0.01	0.11	0.15

Results and Discussion

Performance Benchmarks

The following benchmarks were used to evaluate and compare the proposed strategy:

a) Total cost required to schedule all workflow tasks using available resources. The overall cost comprises execution costs for all jobs and transmission costs for data flow between tasks
b) Total number of Iterations till optimal solution space outcome is achieved
c) Load Distribution over available resources

Results and Analysis

A comparison was made between the traditional Cat Swarm Optimization algorithm CSO, Particle Swarm Optimization algorithm PSO, Bee Colony Optimization Algorithm BCO, and the Enhanced Cat Swarm Optimization algorithm ECSO, which was put into practice. To choose the best crossover method, however, several were tested before the actual ECSO was put into existence.

Crossover Comparison

The outcomes of. Comparing uniform crossover, and two-point crossover, in terms of execution time, and total cost are as follows.

Execution Time

Figure (4) and Table (5) show that the execution time grows with the no. of jobs since more jobs need to be accomplished before the full work can be completed. When looking at the execution times of several methods, it is clear that uniform crossover has the lowest compared to two-point crossover. Uniform crossover exhibits less execution time of 4% reduction over two-point crossover for an average of 5 jobs.

Total Cost

The overall cost vs the number of jobs is plotted in Fig. (5). There is a direct correlation between the number of jobs and the total cost. A higher number of tasks necessitates more computing power, which is exactly why this occurs. Uniform crossover and two-point crossover have the lowest total execution costs. Table (6) shows the total cost comparison of various crossovers concerning no of obs. It is evident from the chart and table that uniform crossover exhibits lower total cost over an average of 5 jobs with a 5% reduction when compared to a two-point crossover.

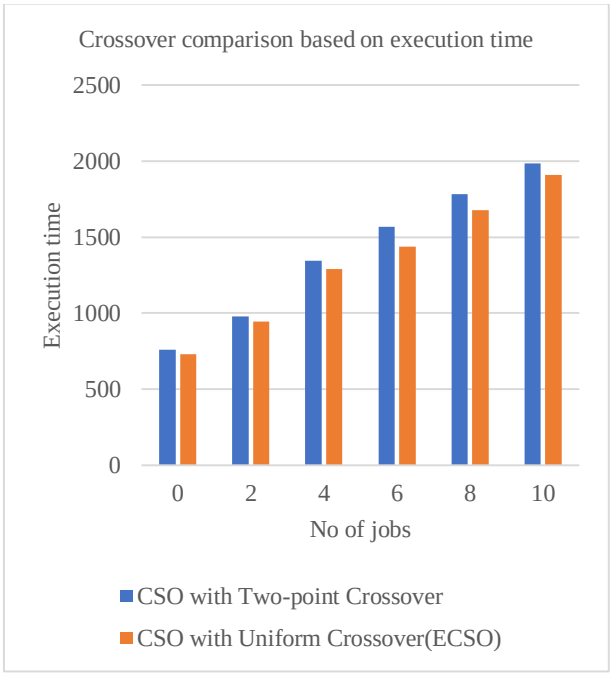

Fig. 4: Crossover comparison based on execution time

Table 5: Crossover comparison depending on execution time

No of Jobs	CSO with two-point crossover	CSO with uniform crossover(ECSO)
0	760	730
2	980	946
4	1343	1289
6	1567	1436
8	1784	1678
10	1985	1908

Table 6: Crossover comparison depending on execution time

No of Jobs	CSO with Two-point Crossover	CSO with Uniform Crossover(ECSO)
0	80086	80002
2	80134	80076
4	92345	91254
6	100042	99068
8	118589	115462
10	123587	118486

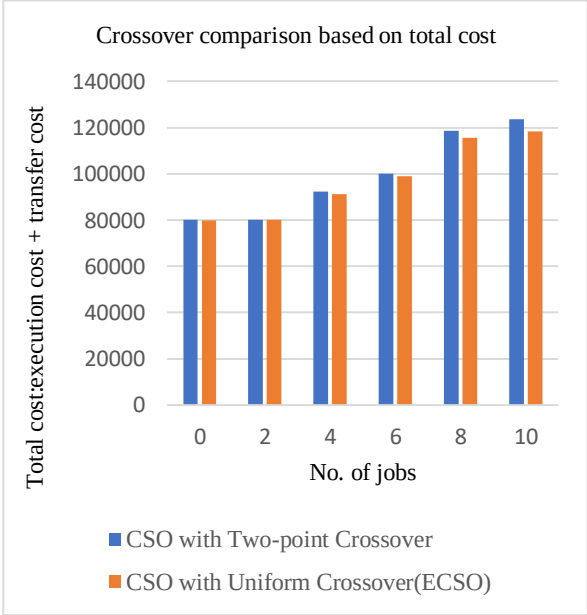

Fig. 5: Crossover comparison depending on total cost

Fitness Comparison

Figure (6) shows that fitness values drop as iteration counts go up. This is the natural consequence of the reasoning that states a lower fitness value is obtained with an increasing number of iterations, leading to greater performance. Once again, when plotting Fitness values against several iterations, the uniform crossover was much ahead of the two-point crossover strategy. Table (7) shows the stats of Fitness values mapping over no of iterations for uniform and two-point crossovers. Uniform crossover exhibits a 4% lowering over two-point crossover for an average number of iterations.

Table 7: Fitness value comparison over Uniform Crossover and Two-point crossover

No of Jobs	CSO with Two-point Crossover	CSO with Uniform Crossover (ECSO)
0	21945	21324
20	18898	18045
40	14756	13985
60	12368	11895
80	9584	8983
100	7568	6756

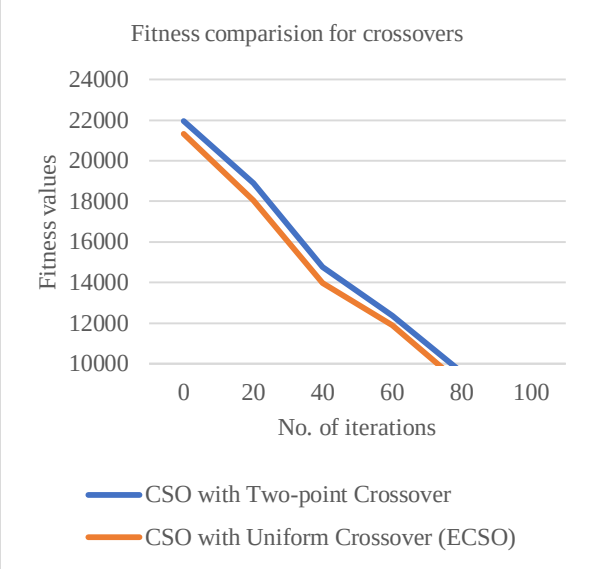

Fig. 6: Fitness comparison over crossovers

Performance Analysis of Proposed ECSO Methods

After deciding that the uniform crossover was the best crossover method, it was used in CSO to create the Enhanced Cat Swarm Optimization proposed method ECSO. In addition, proposed ECSO, regular-pure CSO, PSO, and BCO were evaluated in terms of fitness value, total execution cost, and execution time.

Fitness Value

It was evident that ECSO performed better than pure CSO, PSO, and BCO when looking at the results of fitness values versus several iterations as shown in Fig. (7) and Table (8) which were acquired by running the above-mentioned algorithms. Because a minimization model was employed in the proposed ECSO, it is preferred to have lower fitness values. ECSO produces more satisfactory fitness values. ECSO shows 8% lower fitness values when compared with other algorithms.

Total Execution Cost

Figure (8) shows an association between job number and total execution cost. This graph shows that ECSO has a lower total cost than pure CSO, PSO, and BCO. ECSO's uniform crossover operation leads to faster, better solutions in fewer iterations, reducing costs in subsequent iterations. In pure CSO and other algorithms, this does not occur, resulting in low-quality outcomes. Table (9) shows the total execution costs for the proposed ECSO and other existing algorithms like pure CSO, PSO, and BCO. The stats show that ECSO exhibits a lower cost of 12% whereas pure CSO shows 21%, PSO shows 24%, and BCO shows a 27% lower cost over an average no of jobs.

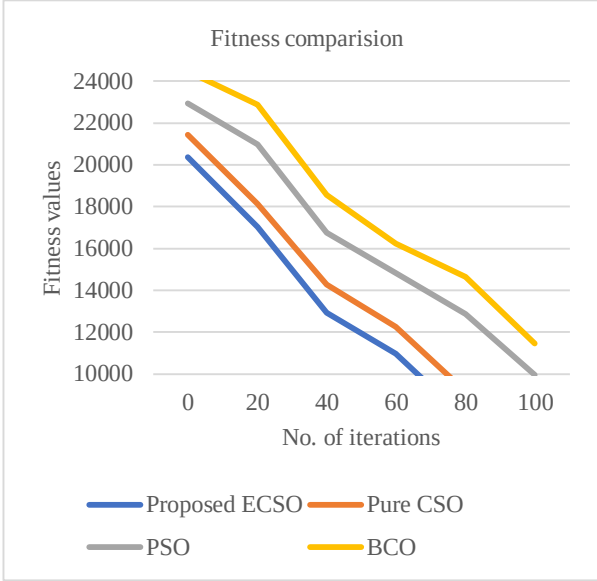

Fig. 7: Fitness comparison for ECSO, CSO, PSO, and BCO algorithms

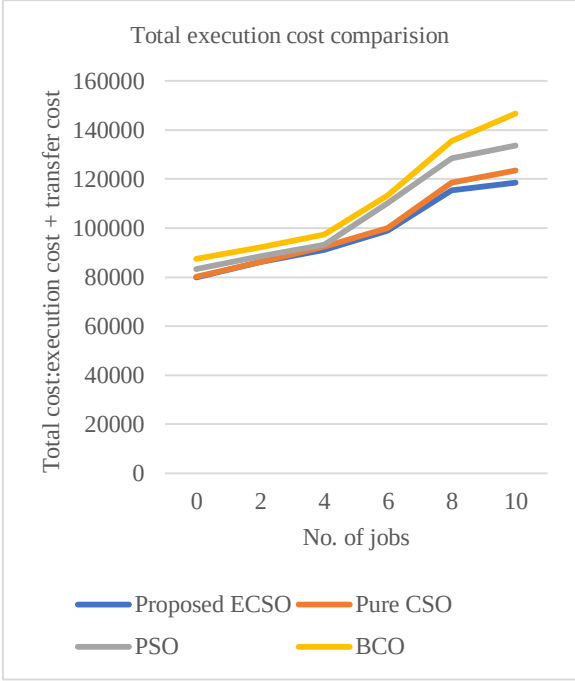

Fig. 8: Total execution cost for ECSO, CSO, PSO, band CO algorithms

Table 8: Fitness value comparison for proposed ECSO, CSO, PSO, and BCO

No of Jobs	Proposed ECSO	Pure CSO	PSO	BCO
0	20354	21445	22945	24455
20	17035	18128	20989	22891
40	12925	14265	16756	18556
60	10955	12248	14824	16242
80	7843	9143	12864	14642
100	5546	7184	9945	11453

Table 9: Total execution cost comparison for proposed ECSO, CSO, PSO, and BCO

No of Jobs	Proposed ECSO	Pure CSO	PSO	BCO
0	80012	80036	83245	87425
2	86054	86114	88423	92243
4	91224	92315	93245	97452
6	99018	100031	110234	113314
8	115423	118535	128354	135564
10	118434	123537	133637	146672

Table 10: Total execution cost comparison for proposed ECSO, CSO, PSO, and BCO

No of Jobs	Proposed ECSO	Pure CSO	PSO	BCO
0	520	516	712	978
2	896	882	1034	1156
4	1192	1174	1254	1467
6	1563	1556	1738	1985
8	1772	1764	1992	2134
10	1928	1905	2284	2376

Total Execution Time

As seen in Fig. (9), execution time was plotted versus job count. ECSO takes equivalent execution times as pure CSO. ECSO lags in this field a bit. The reasons for this are legitimate. The addition of a new operator (Uniform crossover) for each iteration incurs additional computation time, unlike pure CSO. However, this additional step led to other improvements and yielded satisfactory outcomes. Table (10) shows the total execution time of all the algorithms ECSO, pure CSO, PSO, and BCO. From the graph and Table (9) it is clear that the proposed ECSO shows a little bit high execution time compared to pure CSO and less execution time compared to PSO and BCO algorithms. ECSO shows 1.2% more execution time whereas pure CSO, PSO, and BCO show 2.4, 3.8, and 4.6% respectively over an average no of jobs, due to the incorporation of crossover operator.

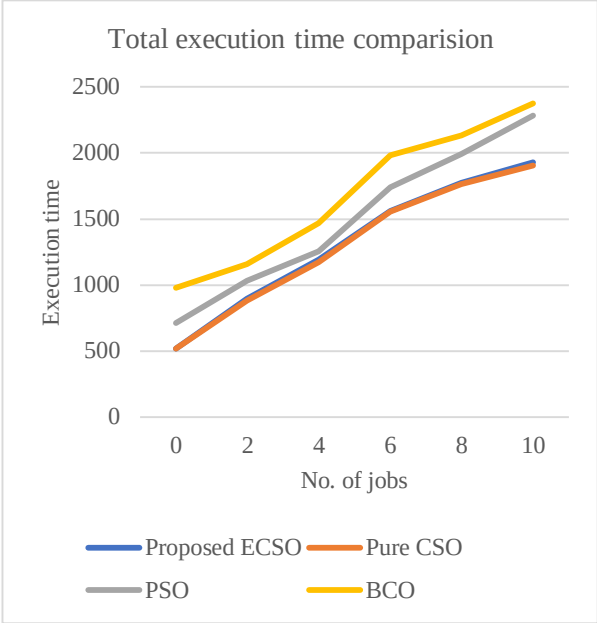

Fig. 9: Total execution time for ECSO, CSO, PSO, and BCO algorithms

Conclusion and Future Directions

The results of this study reveal that swarm intelligence algorithms consistently disregard the second-best sequence for allocating resources. In cases when the final sequence may have included elements from the second-best sequence this resulted in additional cycles to get the same outcome. Because of this, the best-sequence resource allocation algorithm's execution cost went up slightly. As a result, researchers looked into crossover methods. After comparing the two-point and uniform crossover operators the chosen problem was best solved by the uniform crossover.

Enhanced cat swarm optimization is a new swarm-based strategy that is introduced in this study as a scheduling technique. The convergence speed of this method is much faster than that of CSO, PSO, and BCO. The proposed method ECSO achieves the ideal solutions in significantly fewer iterations by utilizing two modes of operation: Calculated update of cat positions and reduced energy wastage in random movement. Therefore, it is more efficient than pure CSO, PSO, and BCO since it converges to solutions faster. The next step in this line of research could be to expand ECSO such that it can schedule tasks with several goals such as minimizing execution time and energy consumption. The Enhanced Cat Swarm Optimisation method (ECSO), a modified version of the CSO algorithm, was the subsequent contribution; it included this crossover mechanism. Using the uniform crossover operator, this method can successfully consider the second-best option at every

point in time. By doing so, a more effective algorithm is created and utilized to eliminate the problem of disregarding the second-best option. To save costs, the best technique to assign current jobs to the available virtual machines was provided by the final solution. As a result, the method for allocating resources in the cloud is now superior. The results were plotted and analyzed according to several characteristics such as execution time execution cost and fitness function. The results showed that the overall execution cost of the jobs on the accessible VMs dropped substantially and the fitness values dropped to good levels in this minimization scenario. However, the time it took to complete the jobs slightly increased and that was for reasonable and practical reasons. This study's foundational finding is that when using ECSO to allocate resources to conduct the necessary activities on available VMs there is a cost improvement.

There are several ways in which the current work on this topic can be expanded. Optimization of the tasks' response time, waiting time, and throughput should take precedence over total execution cost and execution time. Adding a uniform crossing step has recently reduced the execution time. It is possible to try and improve upon the same. The pattern of the CSO population can be better matched by synthesizing a new crossover technique.

Acknowledgment

The authors acknowledge the support and cooperation rendered by all the members directly and indirectly.

Funding Information

The authors have no support or funding to report.

Author's Contributions

Panuganti Hanumantha Rao: Problem notification, and implementation.

Rajakumar Subramanian: Results and conclusion.

Geetha Soman: Overall edited and proof checking.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Ahmed, O. M., & Wafaa, A. M. (2017). A Review on Recent Steganography Techniques in Cloud Computing. *Academic Journal of Nawroz University*, *6*(3), 106–111. https://doi.org/10.25007/ajnu.v6n3a91

Basha, Shaik. M., & Padmavathi, M. (2017). Dynamic and Elasticity ACO Load Balancing Algorithm for Cloud Computing. *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, 77–81. https://doi.org/10.1109/iccons.2017.8250571

Bitam, S. (2012). Bees Life Algorithm for Job Scheduling in Cloud Computing. *Proceedings of the Third International Conference on Communications and Information Technology*, 186–191.

Chu, S.-C., & Tsai, P.-W. (2007). Computational Intelligence Based on the Behavior of Cats. *International Journal of Innovative Computing, Information and Control*, *3*(1), 163–173.

Gandhi, N., Tilbury, D. M., Diao, Y., Hellerstein, J., & Parekh, S. (2002). MIMO Control of an Apache Web Server: Modeling and Controller Design. *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, 4922–4927. https://doi.org/10.1109/acc.2002.1025440

Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L., & Myers, J. (2007). Examining the challenges of scientific workflows. *Computer*, *40*(12), 24-32. https://doi.org/10.1109/MC.2007.421

Hong-Tai, C., & David J., D. (1998). *An Evaluation of Buffer Management Strategies for Relational Database Systems*. Morgan Kaufmann Publishers Inc.

Jeffrey, S. C., Darrell, C. A., Prachi, N. T., Amin, M. V., & Ronald, P. D. (2001). Managing Energy and Server Resources in Hosting Centers. *ACM SIGOPS Operating Systems Review*, *35*(5), 103–116. https://doi.org/10.1145/502059.502045

Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, 1942–1948. https://doi.org/10.1109/icnn.1995.488968

Kesavaraja, D., & Shenbagavalli, A. (2018). QoE Enhancement in Cloud Virtual Machine Allocation Using Eagle Strategy of Hybrid Krill Herd Optimization. *Journal of Parallel and Distributed Computing*, *118*, 267–279. https://doi.org/10.1016/j.jpdc.2017.08.015

Kuan, C. L., Kai, Y. Z., & Jason, C. H. (2014). Feature Selection of Support Vector Machine Based on Harmonious Cat Swarm Optimization. *2014 7th International Conference on Ubi-Media Computing and Workshops*, 205–208. https://doi.org/10.1109/u-media.2014.38

Liang, B., Yan-Li, H., Song-Yang, L., & Wei-Ming, Z. (2010). Task Scheduling with Load Balancing Using Multiple Ant Colonies Optimization in Grid Computing. *2010 Sixth International Conference on Natural Computation*, 2715–2719. https://doi.org/10.1109/icnc.2010.5582599

Meduri, R. K. K., Gutha, S., & Jadala, V. C. (2023). An Architectural Review of Multi-Tenancy in Cloud Computing. In J. Zhao, V. Kumar, R. Natarajan, & T. Mahesh (Eds.), *Handbook of Research on Advancements in AI and IoT Convergence Technologies* (pp. 178–196). IGI Global. https://doi.org/10.4018/978-1-6684-6971-2.ch010

Mostafa, S. A., Gunasekaran, S. S., Mustapha, A., Mohammed, M. A., & Abduallah, W. M. (2020). Modeling an Adjustable Autonomous Multi-Agent Internet of Things System for Elderly Smart Home. In H. Ayaz (Ed.), *Advances in Neuroergonomics and Cognitive Engineering. AHFE 2019* (Vol. 953, pp. 301–311). Springer International Publishing. https://doi.org/10.1007/978-3-030-20473-0_29

Naji, H., Abdulraheem, J. A., & Lailan, M. H. (2020). CPU Scheduling Techniques: A Review on Novel Approaches Strategy and Performance Assessment. *Journal of Applied Science and Technology Trends*, *1*(1), 48–55.https://doi.org/10.38094/jastt1215

Navjot, K., Taranjit, S. A., & Rajbir, S. C. (2011). Comparison of Workflow Scheduling Algorithms in Cloud Computing. *International Journal of Advanced Computer Science and Applications*, *2*(10). https://doi.org/10.14569/ijacsa.2011.021013

Padmavathi, K., & Priyanka, Y. (2017). Crossover Operators in Genetic Algorithms: A Review. *International Journal of Computer Applications*, *162*(10), 34–36. https://doi.org/10.5120/ijca2017913370

Pei-Wei, T., Jeng-Shyang, P., Shyi-Ming, C., Bin-Yih, L., & Szu-Ping, H. (2008). Parallel Cat Swarm Optimization. *2008 International Conference on Machine Learning and Cybernetics*, 3328–3333. https://doi.org/10.1109/icmlc.2008.4620980

Pragati, P., Rakesh, D. R., Manoj, K. J., & Bhaskar B., G. (2017). Understanding and Predicting the Determinants of Cloud Computing Adoption: A Two Staged Hybrid SEM - Neural Networks Approach. *Computers in Human Behavior*, *76*, 341–362. https://doi.org/10.1016/j.chb.2017.07.027

Qi, C., Zhi-Bo, W., & Wen-Mao, G. (2009). An Optimized Algorithm for Task Scheduling Based on Activity Based Costing in Cloud Computing. *2009 3rd International Conference on Bioinformatics and Biomedical Engineering*, 1–3. https://doi.org/10.1109/icbbe.2009.5162336

Rajkumar, B., Chee Shin, Y., Srikumar, V., James, B., & Ivona, B. (2009). Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems*, *25*(6), 599–616. https://doi.org/10.1016/j.future.2008.12.001

Sadeeq, M. A. M., Zeebaree, S. R. M., Qashi, R., Ahmed, S. H., & Jacksi, K. (2018). Internet of Things Security: A Survey. *2018 International Conference on Advanced Science and Engineering (ICOASE)*, 162–166. https://doi.org/10.1109/ICOASE.2018.8548785

Salah Farrag, A. A., Mahmoud, S. A., & El-Horbaty, E. S. M. (2015). Intelligent Cloud Algorithms for Load Balancing Problems: A Survey. *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, 210–216. https://doi.org/10.1109/intelcis.2015.7397223

Saleh, A., Abdullah, A., & Mohammad, A. S. (2018). Impact of Virtualization on Cloud Computing Energy Consumption: Empirical Study. *Proceedings of the 2nd International Symposium on Computer Science and Intelligent Control*, 1–7. https://doi.org/10.1145/3284557.3284738

Sharafi, Y., Khanesar, M. A., & Teshnehlab, M. (2013). Discrete Binary Cat Swarm Optimization Algorithm. *2013 3rd IEEE International Conference on Computer, Control and Communication (IC4)*, 1–6. https://doi.org/10.1109/ic4.2013.6653754

Sharma, D. K., Garg, A., & Jha, A. (2018). Assorted Cat Swarm Optimisation for Efficient Resource Allocation in Cloud Computing. *2018 Fourteenth International Conference on Information Processing (ICINPRO)*, 1–6. https://doi.org/10.1109/icinpro43533.2018.9096807

Singh, G., Su, M.-H., Vahi, K., Deelman, E., Berriman, B., Good, J., Katz, D. S., & Mehta, G. (2008). Workflow Task Clustering for Best Effort Systems with Pegasus. *Proceedings of the 15th ACM Mardi Gras Conference: From Lightweight Mash-Ups to Lambda Grids: Understanding the Spectrum of Distributed Computing Requirements, Applications, Tools, Infrastructures, Interoperability and the Incremental Adoption of Key Capabilities*, 1–8. https://doi.org/10.1145/1341811.1341822

Son, S., Jung, G., & Jun, S. C. (2013). An SLA-Based Cloud Computing that Facilitates Resource Allocation in the Distributed Data Centers of a Cloud Provider. *The Journal of Supercomputing*, *64*(2), 606–637.https://doi.org/10.1007/s11227-012-0861-z

Sreeram, G., Pradeep, S., Rao, K. S., Raju, B. D., & Nikhat, P. (2021). Moving Ridge Neuronal Espionage Network Simulation for Reticulum Invasion Sensing. *International Journal of Pervasive Computing and Communications*, *17*(1), 64–77. https://doi.org/10.1108/ijpcc-05-2020-0036

Tenepalli, D., & Appini, N. R. (2014). Active Resource Provision in Cloud Computing through Virtualization. *2014 IEEE International Conference on Computational Intelligence and Computing Research*, 1–4. https://doi.org/10.1109/iccic.2014.7238373

Urgaonkar, B., Shenoy, P., & Roscoe, T. (2002). Resource Overbooking and Application Profiling in Shared Hosting Platforms. *ACM SIGOPS Operating Systems Review*, *36*(SI), 239–254. https://doi.org/10.1145/844128.844151

Wang, J. (2015). A New Cat Swarm Optimization with Adaptive Parameter Control. In H. Sun, C.-Y. Yang, C.-W. Lin, J.-S. Pan, V. Snasel, & A. Abraham (Eds.), *Genetic and Evolutionary Computing* (Vol. 329, pp. 69–78). Springer International Publishing. https://doi.org/10.1007/978-3-319-12286-1_8

Warneke, D., & Kao, O. (2011). Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud. *IEEE Transactions on Parallel and Distributed Systems*, *22*(6), 985–997. https://doi.org/10.1109/tpds.2011.65

Wu, Z., Ni, Z., Gu, L., & Liu, X. (2010). A Revised Discrete Particle Swarm Optimization for Cloud Workflow Scheduling. *2010 International Conference on Computational Intelligence and Security*, 184–188. https://doi.org/10.1109/cis.2010.46

Xu, Z., Tu, Y.-C., & Wang, X. (2010). Exploring Power-Performance Tradeoffs in Database Systems. *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 485–496. https://doi.org/10.1109/icde.2010.5447840

Ying, L., Tarek, A., & Avneesh, S. (2004). Design, Implementation and Evaluation of Differentiated Caching Services. *IEEE Transactions on Parallel and Distributed Systems*, *15*(5), 440–452. https://doi.org/10.1109/tpds.2004.1278101

Yu, J., & Buyya, R. (2006). Scheduling Scientific Workflow Applications with Deadline and Budget Constraints Using Genetic Algorithms. *Scientific Programming*, *14*(3–4), 217–230. https://doi.org/10.1155/2006/271608

Yu, Z., & Shi, W. (2008). A Planner-Guided Scheduling Strategy for Multiple Workflow Applications. *2008 International Conference on Parallel Processing Workshops*,1–8. https://doi.org/10.1109/icppw.2008.10

Zaki, K., & Saad, H. (2018). Adoption of Cloud Human Resource Information System in Egyptian Hotels: An Experimental Design Research. *International Journal of Heritage, Tourism and Hospitality*, *12*(1), 233–245.https://doi.org/10.21608/ijhth.2018.31514

Zebari, R. R., Abdulazeez, A. M., Zeebaree, D. Q., Zebari, D. A., & Saeed, J. N. (2020). A Comprehensive Review of Dimensionality Reduction Techniques for Feature Selection and Feature Extraction. *Journal of Applied Science and Technology Trends*, *1*(1), 56–70. https://doi.org/10.38094/jastt1224

Zeebaree, S. R., Jacksi, K. F., & Zebari, R. R. (2020). Impact Analysis of SYN Flood DDOS Attack on HAPROXY and NLB Cluster-Base Web Servers. *Indonesian Journal of Electrical Engineering and Computer Science*, *19*(1), 505–512. https://doi.org/10.11591/ijeecs.v19.i1.pp505-512