Modified Moth-Flame Optimization Algorithm for Service Composition in Cloud Computing Environments

Yeling YANG*, Miao SONG
College of Computer and Internet of Things, Chongqing Institute of Engineering, Chongqing, 400056, China

***Abstract*—Cloud computing service composition integrates services, distributed and diverse by nature, into an integrated entity that can meet a user's requirement with better effectiveness. However, some obstacles regarding high latency and suboptimal Quality of Service (QoS) still exist in a dynamic multi-cloud environment. This study addresses the limitations of traditional optimization algorithms in service composition, specifically the premature convergence and lack of population diversity in the Moth-Flame Optimization (MFO) algorithm. We propose the modified MFO algorithm with a new mechanism called Stagnation Finding and Replacement (SFR) to enhance the diversity of the population. It finds the static solutions based on a distance metric from globally optimal representative solutions and replaces them. MFO-SFR drastically improved all QoS metrics, such as response time, delay, and service stability. Empirical evaluations prove that MFO-SFR outperforms the baseline methods of multi-cloud service composition. It provides a computationally efficient and adaptive solution to cloud service composition problems, ensuring better resource utilization and higher user satisfaction in dynamic multi-cloud environments.**

Keywords*—Cloud computing; quality of service; service composition; edge cloud; moth-flame optimization

I. INTRODUCTION

Due to the growing demand for high-performance computing resources, the computing infrastructure has been transformed over the last several years [1]. Several new computational environments, ranging from cluster to grid and cloud computing models, have been created due to technological innovation [2]. As an architectural model, cloud computing provides users with shared computing capabilities available on-demand, with minimal Cloud Service Provider (CSP) interaction [3]. The main goal of this infrastructure is to consolidate geographically distributed resources to achieve greater efficiency, reliability, and performance [4]. Cloud computing facilitates the sharing of services and offers a diverse range of services that can be accessed from any location worldwide [5].

Cloud deployment models can generally be classified into four categories: public, private, hybrid, and community [6]. In public cloud deployments, multiple organizations subscribe to and utilize the exact cloud resources through a shared infrastructure model [7]. This method encourages cost-effectiveness as businesses only bear expenses for their particular resource usage. Private cloud deployments offer dedicated infrastructure environments for a single organization [8]. This model emphasizes heightened security and control as it houses sensitive applications and data within the company's private cloud environment.

Hybrid cloud setups incorporate aspects of both public and private cloud designs. Companies can benefit from this method by having the flexibility to strategically place data and applications according to their sensitivity and processing needs [9]. Sensitive data that requires high security can be stored in the private cloud, while the public cloud can be utilized for cost-efficient and scalable computing operations. Community cloud deployments aim at a particular community of users with common interests or objectives [10]. These designs offer a shared infrastructure setting for numerous organizations in the community and may encourage cooperation and efficient use of resources.

Cloud computing services are broken down into three major classes: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [11]. PaaS offers businesses and developers a robust platform for deploying and hosting software [12]. IaaS allows companies to monitor and control their network, storage, and servers using cloud computing [13]. SaaS involves the provision of software or applications as a service. External providers manage these programs [14]. Users can run applications and software through their web browsers without installing them on their devices.

As the said cloud service model evolves and expands worldwide, it can improve how services are delivered and controlled, allowing the CSP to respond to the different needs of the Cloud Service User (CSU). Service Level Agreements (SLAs) are essential in this situation as they define the desired level of service quality between the CSP and CSU. An SLA is a legally enforceable contract or formally negotiated agreement establishing the understanding and objectives between the CSP and the CSU. The document describes the specific terms and circumstances that govern the provision of services by the CSP.

Cloud computing relies on ensuring accessibility and efficient allocation of all necessary services [15]. There are two main challenges to overcome: first, it is difficult to anticipate the full range of potential service demands, particularly for software services. To solve this problem, complex services must be broken down into more straightforward, discrete, and essential components offered by different providers. Second, selecting the best combination of individual services from multiple providers with varying QoS attributes is an NP-hard optimization problem. Both challenges can be addressed through service composition. To guarantee user satisfaction,

this approach includes service selection from a diverse pool, adherence to composition constraints, identification of crucial QoS indicators, and accommodating the dynamism of services and network conditions.

The dynamic nature of cloud computing environments necessitates effective service composition strategies [16]. While heuristics, metaheuristics, and machine learning algorithms have been employed to address this challenge, each presents distinct advantages and limitations [17]. Heuristic approaches, often limited to single-objective optimization, may struggle with multi-objective problems [18]. Machine learning techniques, such as Deep Q Network (DQN), ADEC, and DQTS, have shown promise in solving multi-objective service composition problems [19]. However, their reliance on extensive training data can be prohibitive, particularly in complex scenarios. Metaheuristic algorithms, including evolutionary and swarm-based methods, offer a versatile and scalable approach to multi-objective optimization [20].

This paper proposes a novel service composition method for cloud computing environments enhancing the Moth-Flame Optimization (MFO) algorithm. By integrating a Stagnation Finding and Replacing (SFR) mechanism, the MFO algorithm addresses the common challenge of stagnation during optimization processes. This innovative approach dynamically detects and replaces stagnant solutions, effectively rejuvenating the search process and preventing the algorithm from converging prematurely on suboptimal solutions. Briefly, this research contributes to the following areas.

- We propose a novel service composition strategy designed explicitly for multiple-cloud environments. This strategy capitalizes on the distributed characteristics of service elements across multiple clouds to improve service quality.
- We introduce the MFO-SFR algorithm, a significant advancement over the traditional MFO algorithm. The MFO-SFR algorithm demonstrates demonstrably improved performance and diversification capabilities.
- A key innovation of our approach is the SFR strategy. This strategy dynamically detects and replaces stagnant solutions within the optimization process, leading to an overall improvement in performance.
- We incorporate an archive mechanism to enhance solution diversity further and ensure a more comprehensive search space exploration. This mechanism integrates both representative and globally optimal solutions encountered during the search process.

The rest of the paper is organized as follows. Section II presents related work on service composition and optimization techniques, identifying the gaps the current study intends to fill. Section III includes the simulation setup, results, and analysis to illustrate the efficiency of the proposed approach for improving key QoS metrics. Lastly, Section IV concludes the study with an overview of key results and contributions and suggests possible directions for further investigation.

II. Related Work

Cloud computing environments demand real-time execution for quality-of-service conscious service composition. This entails maintaining coordination between achieving the best service configurations and ensuring efficient execution times for service composition. Prior research thoroughly examined combinatorial optimization methods to identify optimal service compositions within a specified time constraint. Nevertheless, the continuous expansion of cloud services results in a proportional increase in the problem's search space size. Consequently, these conventional methods are less effective at efficiently combining services within acceptable time limits.

As outlined in Table I, Karimi, et al. [21] suggested utilizing a genetic algorithm-based method to attain global optimization while complying with SLAs. Their methodology involves using service clustering to decrease the complexity of the search space and using association rule mining to improve service composition efficiency based on historical service consumption data. Experimental evaluations show that the proposed strategy is more efficient than comparable efforts.

TABLE I. Optimization Techniques for Service Composition in Cloud Computing Environments

Reference	Methodology	Key features	Limitations
[21]	Genetic algorithm with service clustering and association rule mining	Decreases complexity of search space; improves efficiency with historical data	Potential scalability issues with growing service datasets
[22]	The hybrid of the artificial bee colony and genetic algorithm	Two-stage optimization: GA for fitness, ABC for selection	High computational complexity
[23]	Capuchin search algorithm	Inspired by capuchin monkeys' social foraging behavior, focusing on both global and local optimization	It may require fine-tuning for different cloud environments
[24]	Honeybee mating optimization with trust-based clustering	Incorporates honeybee reproductive behavior; tackles trust issues	Underperforms in computational time for large-scale problems
[25]	Combining Aquila optimizer and particle swarm optimization	Hybrid approach; adaptive transition strategy	A complex implementation may be resource-intensive
[26]	Ant colony optimization with multi-pheromone mechanism and GA-inspired mutation	Addresses ACO's local optima issue; balanced exploration and exploitation	Potential risk of premature convergence without proper parameter tuning

Sefati and Halunga [22] used the Artificial Bee Colony and Genetic Algorithm (ABCGA) to generate optimal service compositions. This approach utilizes a two-stage optimization process. During the initial phase, a Genetic Algorithm (GA) determines potential services that satisfy particular fitness

requirements. Once the fitness function evaluation produces encouraging outcomes, these potential services are introduced to the Artificial Bee Colony (ABC) algorithm during the second step. The ABC algorithm enhances the service selection process by determining the service most closely matches individual user requirements. The effectiveness of the suggested ABCGA approach was assessed through experimentation utilizing the CloudSim simulator.

To tackle the task of enhancing service composition for multiple Quality of Service (QoS) metrics in cloud environments, Wang [23] introduced a new approach that utilizes the Capuchin Search Algorithm (CapSA). This algorithm mimics capuchin monkey social foraging patterns and exhibits its efficacy in addressing global and local optimization challenges. CapSA is chosen due to its inherent simplicity, reduced processing complexity, and well-rounded approach to exploration and exploitation. By conceptualizing service composition as an optimization problem, the proposed methodology seeks to reduce energy consumption and costs. According to empirical evaluations, the CapSA-based strategy substantially outperforms existing methods for achieving faster convergence and producing superior service compositions.

Zanbouri and Jafari Navimipour [24] investigated how Honeybee Mating Optimization (HMO) can address service composition in cloud computing environments. They focus on the connections between worker bees and the queen bee when choosing a new queen, utilizing knowledge from honeybee reproductive behavior. The optimization algorithm incorporates these biological inspirations to enhance the QoS. In addition, a trust-based clustering technique is used to tackle trust-related concerns specifically. The simulation results obtained from a C# implementation indicate that the suggested method outperforms existing algorithms, including GA, Particle Swarm Optimization (PSO), and the discrete best-guided ABC algorithm, for small-scale service composition problems. The enhancement results from the clustering method diminish the scope of the search and thus enhance the speed of response while also allowing for the choice of more dependable services. However, extensive simulations demonstrate that the computational time performance of the suggested method underperforms the average results of earlier studies.

Liu [25] developed a novel hybrid optimization technique known as the Integrated Aquila Optimizer (IAO), combining the functions of the PSO algorithm and Aquila Optimizer (AO). Hybridization addresses the inherent limitations of individual algorithms, such as their vulnerability to getting stuck in local optima and their limited ability to generate diverse solutions. The proposed IAO algorithm includes an innovative transition strategy for these difficulties. This method allows the AO and PSO algorithms to adjust their search operators flexibly. By employing this method, possible solutions are consistently improved. Utilizing both the AO and PSO algorithms can be a strategic move when each method reaches a standstill or when the range of possibilities decreases. This adaptive behavior improves the effectiveness and efficiency of the IAO approach. The proposed solution was thoroughly evaluated by testing in the CloudSim simulation environment. The numerical data indicate that the IAO technique successfully enhances dependability, availability, and cost optimization within cloud computing.

Bei, et al. [26] discussed the composition of services in multiple cloud scenarios. They proposed an Ant Colony Optimization (ACO) algorithm to optimize QoS parameters, incorporating a multi-pheromone mechanism. This technique seeks to surpass conventional ACO constraints, which may become trapped in local optima. They incorporated a mutation operation influenced by the GA to improve the algorithm's exploration ability and avoid premature convergence. This hybridization approach promotes a more equitable and effective process of exploring and exploiting, resulting in the discovery of service compositions with superior QoS metrics, such as decreased latency and enhanced response times. Proposed method

A. Problem Definition and System Architecture

Cloud computing has made significant advancements in the past decade. Global infrastructure and market expansion have given rise to several cloud computing forms, including central and edge clouds. Central clouds are frequently utilized for extensive data analysis and deep learning training because of their robust processing and storage capacities. On the other hand, edge clouds are essential for collecting data, controlling processes in real-time, perceiving information intelligently, and making quick decisions at the outermost part of the network.

In contrast to centralized cloud infrastructures, edge computing provides users access to robust computational resources while mitigating the delay challenges inherent in remote data center interactions. This dramatically minimizes the data transmitted on the leading network and guarantees quick response times for upcoming services requiring minimal delays. As a result, the widespread use of these services in edge clouds is anticipated to grow prevalent.

This paper explores the architecture of cloud-edge devices, where service elements are mainly placed on a centralized cloud. Docker and other containerization technologies facilitate seamless and efficient migration to the cloud when consumers need a particular service component. This methodology enables the combination of services and the virtualization of resources (such as storage and computation) to meet users' requirements, as shown in Fig. 1. Docker containers are gaining popularity in cloud computing, as evidenced by their use in constructing genuine cloud environments for research purposes. Cloud services are highly advantageous in a dynamic cloud environment due to their effectiveness and ease.

The current service landscape is experiencing a significant change towards autonomous and loosely connected service designs, commonly called microservice architectures. Although service components can be spread out throughout different edge clouds, there is still a need to investigate and understand the current approaches for combining services in multiple-cloud setups. On the other hand, a multiple-cloud setup enables consumers to select from a range of services that perform better than single CSPs with limited computing capacity. In addition, multiple-cloud deployments provide built-in redundancy, which helps prevent equipment failures and improve the system's overall stability.

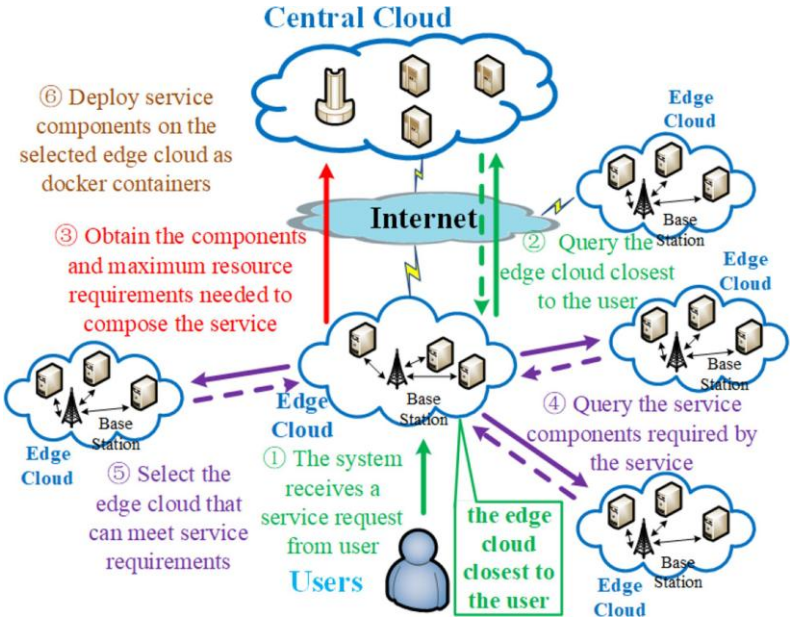

Fig. 1. Architecture of cloud-edge-devices integration.

This paper introduces a multi-cloud service composition architecture, depicted in Fig. 1, comprising M consumers, N edge clouds, and a central cloud. The central cloud is a repository for comprehensive service-related information and hosts a global network controller. Service components are distributed across an edge cloud infrastructure. User service requests are initially directed to the nearest edge cloud for preliminary processing. Each edge cloud maintains a local database containing information about neighboring edge clouds and available service components.

Table II is a crucial system component, providing essential network topology information. The index n_i varies from 0 to N-1, where N is the total number of edge clouds in the network. This ensures that all possible connections between edge clouds are considered. $path_i$ represents the optimal route linking the current edge cloud and another edge cloud in terms of the fewest hops. This information is crucial for routing data efficiently. hop_i quantifies the number of network hops between the current edge cloud and any other edge cloud within the network. A lower hop count generally indicates a more efficient communication path.

TABLE II. NETWORK TOPOLOGY INFORMATION FOR EDGE CLOUDS

Edge cloud count	Path	Hop
n_1	$path_1$	hop_1
n_2	$path_2$	hop_2
...	...	...
n_i	$path_i$	hop_3

Table III serves as a critical repository for service component metadata within the edge cloud environment. Service element names identify the service items available in the edge cloud and its neighbors. QoS attributes provide essential performance metrics for each service element, such as delay and reliability. The QoS attribute for the j^{th} parameter of the i^{th} service element is represented as η_{ij}. This standardized notation facilitates data manipulation and analysis.

TABLE III. SERVICE ELEMENT DATABASE

Service element	Edge cloud count	QoS
$element_1$	n_1	η_1
$element_2$	n_2	η_2
...	...	...
$element_i$	n_i	η_3

Upon receipt of a service request, the proximate edge cloud initiates communication with a central controller to procure optimal computational resources, storage capacity, and network bandwidth. The service composition process proceeds in situ if the local edge cloud possesses sufficient residual capacity to fulfill the service's maximal requirements. Conversely, if resource constraints are encountered, the edge cloud embarks on a search for an adjacent edge cloud with minimal network hops. This iterative exploration continues until an edge cloud with ample resources to accommodate the service composition is identified.

Eq. (1) quantifies the resource demands (R_l) of service l. The set L encapsulates the services scheduled for orchestration on edge cloud i, while C_i represents the aggregate resource capacity of edge cloud i. These parameters constitute critical determinants in edge cloud service provisioning.

$$\sum_{l \in L} R_l \leq C_i, i \in N \quad (1)$$

Containerization virtualization has played a significant role in microservice adoption. Cloud computing can utilize containerization to flexibly install, migrate, or scale virtual machines under changing service demands. Containerization benefits conventional virtual machines by using the host's operating system kernel. This strategy minimizes the

administrative burden of delivering resources as needed and promotes optimal resource utilization. Containerized microservices typically involve the simultaneous creation of lightweight components within containers, which are then provisioned and scaled based on their requirements.

Expanding on these ideas, this method enables the quick and flexible deployment of service components by utilizing containerization technologies such as Docker in a multiple-cloud setting. This allows for deploying all necessary service components for composition onto respective edge clouds. A notification mechanism is built to guarantee real-time accuracy of records held in edge clouds and service components. Whenever one edge cloud stops serving customers or undergoes modifications to its deployed service components, it sends these updates to all connected edge clouds. Utilizing this broadcast technique allows for the timely updating of databases on other edge clouds, ensuring data consistency throughout the system.

B. QoS model

Services typically comprise k distinct groups, each containing abstract service component definitions with specific order requirements. Users seek a combination of services that fulfill user-specified requirements and QoS constraints to complete their desired operations during service composition.

The service composition process is divided into K steps according to the user's requirements. Every individual step, S_i, is linked to a particular service set. The algorithm chooses service components from each set S_i to fulfill the user's operation. The selection procedure yields numerous possible routes from the initial service component set (S_1) to the final set (S_k). The ideal combination of services is attained by determining the pathway that produces the most advantageous service combination.

When choosing a service, both the functional and non-functional aspects are considered. Functional attributes pertain to the explicit purpose and content offered by a service, whereas non-functional attributes encompass the overall quality of the service, as evaluated using QoS measurements.

Services are evaluated on the essential aspects of QoS, as described by internationally recognized standards organizations. QoS, as specified by these standards, includes non-functional features such as throughput, availability, response time, and dependability.

Ensuring high-quality service while combining multiple services is essential for distinguishing between the various components of the service. This optimization method assesses the QoS attributes of the constructed service. QoS parameters can be divided into two main categories: dynamic attributes, which include response speed, dependability, and availability, and fixed attributes, which include security, accuracy, and robustness.

This study focuses on throughput, reliability, delay, and response time. Throughput indicates the maximum rate at which data can be processed or transmitted successfully. Availability refers to the likelihood that service components are operational and ready for use in a particular environment. Delay refers to the time it takes for data packets to travel between a server hosting a service component and a client.

Response time represents the time the service provider takes to respond to a user's service request. Table IV presents QoS attribute formulas for composed services. Calculations rely on j (number of service components chosen from service set i) and k (total number of service sets).

TABLE IV. QOS ATTRIBUTE FORMULAS FOR COMPOSED SERVICES

QoS parameters	Expression
Delay	$\sum_{i=1}^{k} L(\eta_{ij})$
Throughput	$\sum_{i=1}^{k} L(\eta_{ij})$
Availability	$\sum_{i=1}^{k} A(\eta_{ij})$
Response time	$\sum_{i=1}^{k} R(\eta_{ij})$

It is crucial to optimize various QoS parameters during service composition. However, it is equally essential to guarantee service stability and other relevant metrics. This study introduces a novel concept of QoS parameter stability, defined by the absolute value of each parameter across service elements. Eq. (2) represents the stability calculation for QoS parameter j within the service.

$$Sta_j = \sum_{i=1}^{k-1} \left\| \eta_{(i+1)(j+1)} - \eta_{ij} \right\| \quad (2)$$

Services with minimal cumulative absolute differences between their QoS parameters (QoS_i) are considered more stable. This approach mitigates significant fluctuations in QoS. Additionally, to prevent data size variations across service sets from skewing the final results, this paper incorporates a data normalization step for the QoS information associated with the service components. Following normalization, higher parameter values correspond to superior performance. Consequently, all subsequent references to QoS metrics (response time, availability, throughput, and delay) within this work will pertain to their normalized values.

This paper proposes a methodology that considers all four QoS criteria to determine the most effective technique for composing consumer services. This technique guarantees the optimization of these crucial parameters. The following section will explore an improved service composition technique based on the modified MFO algorithm. This approach has been specifically developed to boost the optimization of QoS.

C. Enhanced MFO algorithm

The MFO algorithm mimics the behavior of moths in nature. The unique navigational strategies of moths have generated considerable interest among researchers studying metaheuristics. Moths are nocturnal creatures that rely on lunar illumination for navigation Shehab, et al. [27]. Moth flight patterns can be mathematically modeled using the transverse orientation mechanism (Fig. 2). This strategy approximates a straight-line trajectory by maintaining a constant angular relationship with the moon. When faced with artificial light sources, moths divert from this path. When the moth is close to

the light source, it initiates a helical flight pattern that guides it towards the flame. Each moth symbolizes a potential solution, and every position is represented as a matrix of decision variables, as shown below.

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n-1} & x_{1,n} \\ x_{2,1} & \ddots & \cdots & \cdots & x_{2,n} \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ x_{N-1,1} & \cdots & \cdots & \ddots & x_{N-1,n} \\ x_{N,1} & x_{N,2} & \cdots & x_{N,n-1} & x_{N,n} \end{bmatrix} \end{bmatrix} \quad (3)$$

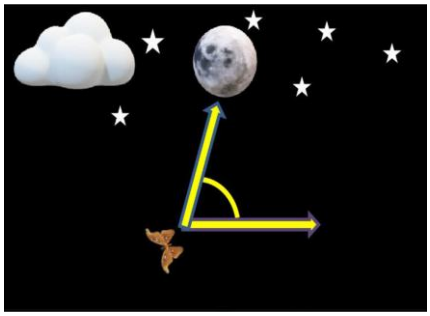

Fig. 2. Moth flight patterns model using transverse orientation mechanism.

In Eq. (3), *N* stands for the population size, equal to the total number of moths in the swarm. Also, *n* indicates the problem dimension, which measures how many variables are involved in the optimization process. The fitness of a particular moth is determined as follows.

$$Fit[X] = \begin{bmatrix} Fit[X_1] \\ Fit[X_2] \\ \vdots \\ Fit[X_n] \end{bmatrix} \quad (4)$$

Eq. (5) shows the flame matrix. Since all moths fly around a flame, the size must match the moth matrix previously defined.

$$FM = \begin{bmatrix} FM_1 \\ FM_2 \\ \vdots \\ FM_N \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} Fm_{1,1} & Fm_{1,2} & \cdots & Fm_{1,n-1} & Fm_{1,n} \\ Fm_{2,1} & \ddots & \cdots & \cdots & Fm_{2,n} \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ Fm_{N-1,1} & \cdots & \cdots & \ddots & Fm_{N-1,n} \\ Fm_{N,1} & Fm_{N,2} & \cdots & Fm_{N,n-1} & Fm_{N,n} \end{bmatrix} \end{bmatrix} \quad (5)$$

Eq. (6) determines the corresponding fitness of the flame matrix.

$$Fit[FM] = \begin{bmatrix} Fit[FM_1] \\ Fit[FM_2] \\ \vdots \\ Fit[FM_n] \end{bmatrix} \quad (6)$$

The MFO algorithm relies heavily on two primary components: flames and moths. Moths fly through flames to achieve desired results. As shown in the equation below, the logarithmic spiral function is used to model the spiral movement of the moth.

$$X_i^{K+1} = \begin{cases} \delta_i . e^{bt} . \cos(2\pi t) + Fm_i(k) & i \le N.FM \\ \delta_i . e^{bt} . \cos(2\pi t) + Fm_{N.FM}(k) & i \ge N.FM \end{cases} \quad (7)$$

δ_i represents the Euclidean distance between a moth's current position (X_i^K) and its corresponding flame (Fm_i). This value indicates the moth's proximity to a possible optimal solution. Spiral flight patterns of moths are determined by *b* and *t*, a uniformly distributed random number between -1 and 1. Moths and flames are attracted to each other based on these parameters, as shown in Fig. 3. The moth's trajectory towards the flame is depicted in Fig. 4. Throughout the optimization process, *t* gradually decreases toward a balance between exploitation (focusing on promising areas) and exploration (searching the entire search area). The mathematical representation of *t* is presented below, and Fig. 5 depicts the moth's next position.

$$r = -1 + Currrent_{iter}(\frac{-1}{Max_{iter}}) \quad (8)$$

$$t = (r-1) \times k + 1 \quad (9)$$

The optimization process depends on three variables: Max_{iter}, *k*, and *r*. Max_{iter} specifies the maximum number of iterations, *k* indicates a uniformly distributed random number between 0 and 1, and *r* singularity ensures convergence. The value of *r* is linearly reduced throughout the optimization to balance exploration (searching the entire search space) and exploitation (focusing on promising regions).

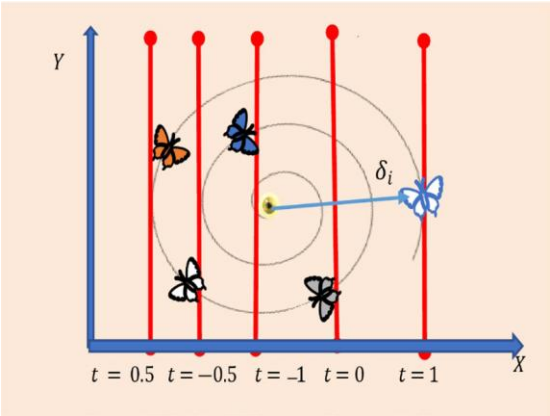

Fig. 3. Attraction mechanism between moths and flames.

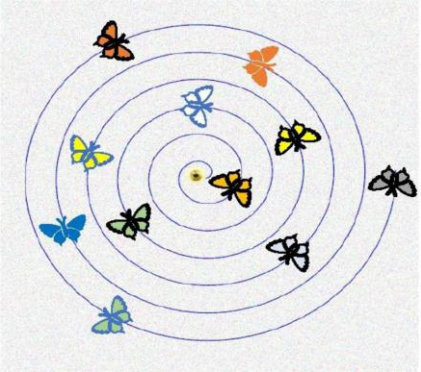

Fig. 4. Moth's spiral trajectory toward the flame.

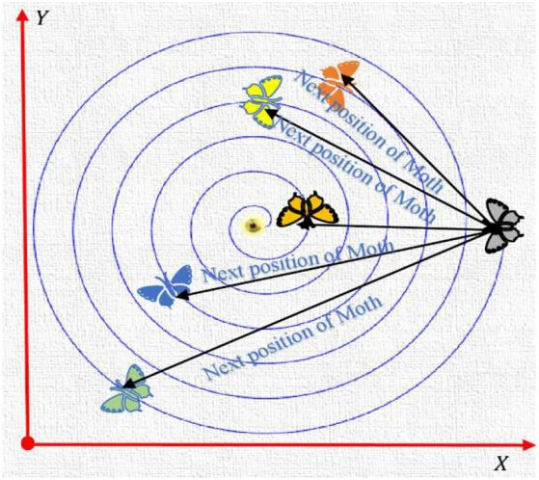

Fig. 5. Decreasing parameter t for balancing exploration and exploitation in optimization.

During the optimization process, the moths with the highest fitness values continually move towards the most promising solutions, indicated by the flames. This phenomenon can be explained by the mechanism in which the number of flames (represented as *N.FM* in Equation 10) gradually reduces with each cycle. This decrease in the number of flames efficiently focuses the search effort on the most favorable areas of the search space.

$$N.FM = round\left(N.FM_{Last\ iter} - Current_{iter}\frac{(N.FM_{Last\ iter}-1)}{Max_{iter}}\right) \quad (10)$$

A population of moths is represented by the matrix $X(t) = \{X_{1D}(t), \dots, X_{iD}(t), \dots, X_{ND}(t)\}$ in a D-dimensional search space at iteration *t*. Each element $X_{iD}(t)$ represents the position of the i^{th} moth within the problem space. The initial positions of all moths are generated randomly using a uniform distribution during the first iteration ($t = 1$). In subsequent iterations ($t \geq 2$), the SFR mechanism is used to update moth positions based on Eq. (11).

$$X_i(t+1) = \begin{cases} D_i^{\alpha}(t) \times e^{b\tau} \times \cos(2\pi t) + F_j(t) & if\ i \leq R(t) \\ D_i^{\beta}(t) \times e^{b\tau} \times \cos(2\pi t) + F_R(t) & else \end{cases} \quad (11)$$

SFR is characterized by its core components, represented by Eq. (12) and Eq. (13). The constant *b* determines the shape of the logarithmic spiral employed by the moth, and τ indicates a random number uniformly distributed between -1 and 1. $F_j(t)$ and $F_R(t)$ represent the positions of the j^{th} and the R^{th} flame, respectively. The parameter *r* is calculated using Eq. (8).

$$D_i^{\alpha}(t) = \left|F_j(t) - M_i(t)\right| \quad (12)$$

$$D_i^{\beta}(t) = \begin{cases} \left|F_j(t) - X_i(t)\right| & \varphi_i > 0 \\ Select\ a\ random\ position\ form\ Arc & \varphi_i = 0 \end{cases} \quad (13)$$

Eq. (14) calculates the mean distance, denoted by φ_i, for each moth. This distance is computed based on the individual dimensions (X_{iq}) of the i^{th} moth and the corresponding dimensions (F_{jq}) of its associated flame (*j*). The index *j* for each moth is determined using Eq. (15). This equation involves sorting the results obtained from Eq. (14) in descending order to identify the most "distant" moths and subsequently utilizing these indices as flame indexes within Eq. (13).

$$\{\varphi_1, \dots, \varphi_i, \dots, \varphi_N\} \leftarrow \varphi_i = \frac{1}{D} \times \sum_{q=1}^{D}\left|F_{jq}(t) - X_{iq}(t)\right| \quad (14)$$

$$\{\varphi_1, \dots, \varphi_j, \dots, \varphi_N\} \leftarrow Sort(\varphi_1, \dots, \varphi_i, \dots, \varphi_N) \quad (15)$$

The archive construction process serves a dual purpose: enhancing population diversity and accelerating convergence towards promising regions within the search space. This is achieved by storing representative flames and the best solutions encountered during optimization. The archive, denoted by *Arc*, is represented by the matrix $M = \{M_1, \dots, M_i, \dots, M_K\}$, where *K* signifies the predefined archive size. Each element $M_i = [m_{i1}, m_{i2}, \dots, m_{iD}]$ represents a vector position within the archive memory.

The construction of the archive involves two key steps: generating Representative Flame (RF) and archiving entries. The first step leverages the dual population ($dual_{Pop}$) and dual fitness values ($dual_{Fit}$) created based on the flame construction process outlined in Fig. 6. Eq. (16) calculates the RF position, representing the average of all flame positions. Here, *C* denotes the total number of moths considered, and F_{id} represents the d^{th} dimension of the i^{th} flame. Two new entries are added to the archive memory *M*: the global best flame position and the calculated RF position. If the archive reaches its total capacity (*K*), a random replacement strategy is implemented, replacing two existing entries with new entries.

$$RF_d(t) = \frac{1}{C}\sum_{i=1}^{C} F_{id}(t) \quad (16)$$

Input: *X*: the positions of moths, Fit: the fitness values of moths, F: the position of the flame, and *OF*: the fitness values of flames.

Flame construction in the first iteration when $t = 1$.

1. Sort the vector Fit in ascending order and extract the sorted index in $\{j_1, j_2, \dots, j_N\}$.
2. Construct the flame matrix $F(t) = \{F_1 \leftarrow X_{j1}, F_2 \leftarrow X_{j2}, \dots, F_N \leftarrow X_{jN}\}$.

Flame construction for the rest iteration when $t > 1$.

1. Construct matrix $dual_{Pop}$ by combining matrices $F(t)$ and $X(t-1)$.
2. Construct vector $dual_{Fit}$ by combining vectors $OF(t)$ and $Fit(t-1)$.
3. Sort the vector $dual_{Fit}$ in ascending order and extract the sorted index in $\{j_1, j_2, \dots, j_{2N}\}$.
4. Construct the flame matrix $F(t) = \{F_1 \leftarrow X_{j1}, F_2 \leftarrow X_{j2}, \dots, F_N \leftarrow X_{jN}\}$.

Fig. 6. Construction of representative flame and archiving entries.

III. SIMULATION AND RESULTS

A series of tests were performed on a Windows 8.1 computer powered by an Intel Core i5-460M processor at 2.53 GHz and 16 GB of RAM. This study employed a system model comprising 32 edge clouds and a primary cloud, as illustrated in Fig. 1. The Quality of Service for Web Services (QWS) dataset contained 2507 services, each characterized by nine QoS features: description, delay, best practices, consistency, reliability, throughput, availability, and response time. For this research, delay, throughput, availability, and response time were the primary QoS parameters, with their respective ranges and units detailed in Table V. A comparative evaluation was conducted to measure the proposed algorithm against traditional MFO, PSO, and WOA algorithms, evaluating fitness, stability, delay, and response time.

TABLE V. QOS PARAMETERS AND RANGES

Parameters	Dimensions	Unit
Delay	0.1-4500	ms
Throughput	0.1-50	Mbps
Availability	5-100	%
Response time	30-5000	ms

Fig. 7 and Fig. 8 compare the proposed algorithm and its counterparts regarding fitness and stability, respectively. To conduct this analysis, 100 to 1000 service instances were chosen at random extracted from the QWS dataset, with inclusion criteria limited to services comprising at least five components. Fig. 7 demonstrates the better fitness performance of the developed algorithm compared to its competitors. While fitness is a crucial metric, the ultimate objective is to maximize service QoS and maintain stability.

Fig. 9 and Fig. 10 illustrate the comparative performance of the algorithms in terms of delay and response time. All QoS parameters were normalized to mitigate the influence of varying parameter scales. Consequently, higher values indicate improved optimization outcomes. The results in Fig. 9 clearly reveal the superiority of the proposed algorithm in minimizing delay. Similarly, Fig. 10 reveals a significant advantage of the proposed algorithm in reducing response time compared to other methods.

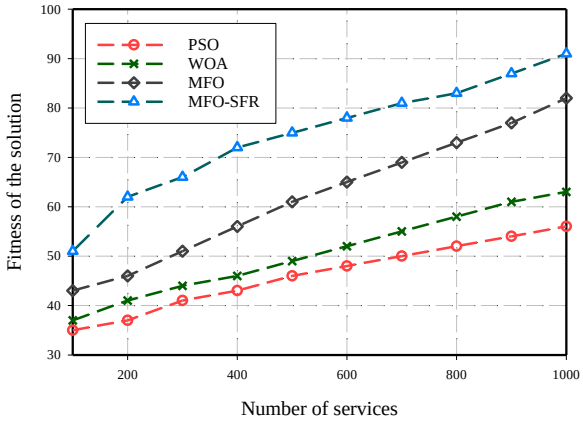

Fig. 7. Fitness comparison.

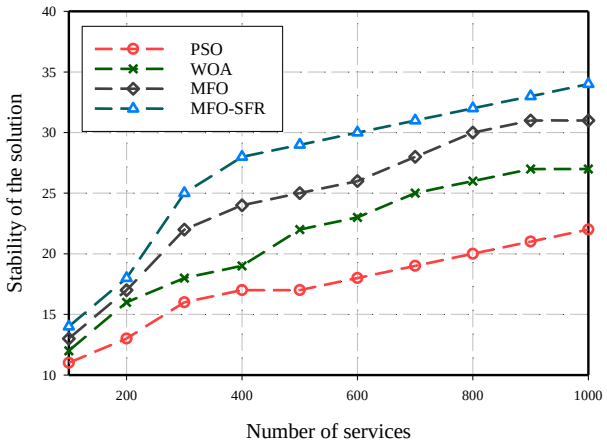

Fig. 8. Stability comparison.

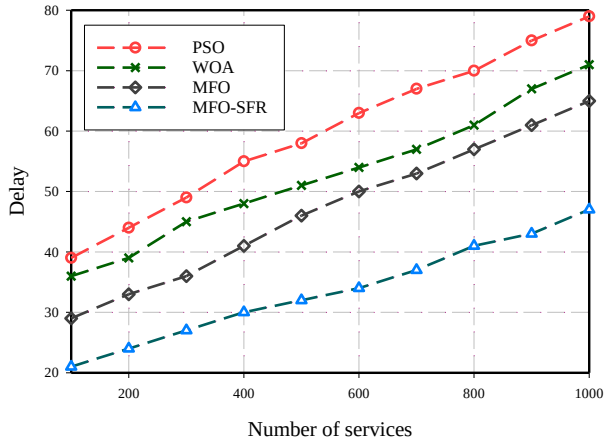

Fig. 9. Delay comparison.

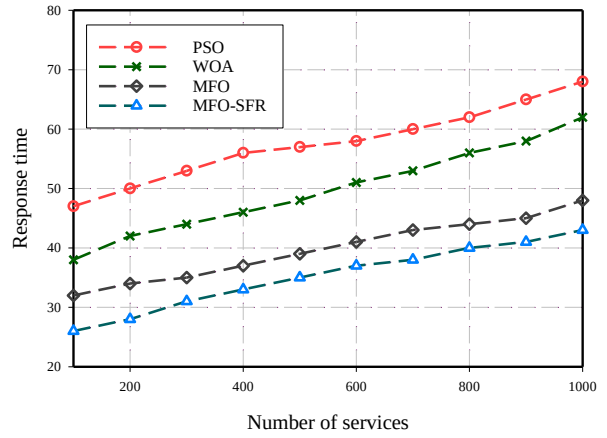

Fig. 10. Response time comparison.

The experimental results distinctly demonstrate the superiority of the proposed algorithm over traditional optimization methods. From the fitness performance, it is obvious that the proposed algorithm constantly finds higher-quality solutions. This is mainly because the SFR mechanism can dynamically detect and replace the stagnant solution, making the search process more diverse and effective. Enhanced fitness will naturally provide better QoS outcomes,

crucial for service composition in multi-cloud environments. Moreover, the proposed algorithm outperforms others in service stability, which shows its robustness and adaptability to dynamic cloud scenarios.

The proposed MFO-SFR algorithm guarantees great efficacy when the delay and response time parameters are analyzed. A minimum delay shows how it may optimize the critical time-sensitive aspect of cloud service delivery, guaranteeing users' satisfaction with the service. On the other hand, the response times that could be retrieved by using this algorithm also promise to ensure that it may further enhance efficiency in the performance of any service. Therefore, the findings support the stated objectives of the study on QoS parameters in cloud environments and further indicate the practical relevance of the algorithm. Compared to traditional approaches, the proposed method yields better results in scalability and efficiency in solving complex service composition challenges in multi-cloud ecosystems.

IV. CONCLUSION

The swift advancement of cloud computing has led to the widespread growth of a wide range of cloud-based services. However, guaranteeing awareness of QoS during the construction of services is a substantial difficulty in cloud systems. Many individual services cannot handle complex requests and different needs that arise in real-world situations. Often, a solitary service may not be enough to fulfill the particular needs of consumers, therefore requiring the amalgamation of many services to attain the needed functionality. Due to its intrinsic NP-hard difficulty, service composition has been widely studied using various metaheuristic algorithms. This paper proposed an improved MFO algorithm with the SFR mechanism for the optimization of service composition in multi-cloud computing environments. Our approach overcomes the deficiency of early convergence in the traditional MFO by maintaining the diversity in the population with the aid of the SFR mechanism. It ensures that static solutions are identified and replaced with promising ones, which enhances the whole optimization process. The empirical results showed that our approach enhances significantly the QoS metrics such as stability of service, response time, and delay. We evaluated an algorithm using a realistic system model and the QWS dataset, considering the main QoS parameters. The comparative analysis confirmed the superior fitness and stability of our algorithm.

While the results are encouraging, many directions are open to future research: First, the proposed algorithm can be extended by allowing multi-objective optimization scenarios in which many QoS attributes can be optimized simultaneously. Second, exploring the integration of machine learning techniques and metaheuristics may lead to advanced adaptability and efficiency in service composition strategies. Third, using the MFO-SFR algorithm in other domains, like IoT-enabled edge computing, hybrid cloud environments, or real-time service orchestration, is a promising avenue for further exploration. Finally, real-world cloud deployments of the proposed approach can shed light on many practical feasibility and scalability issues.

ACKNOWLEDGMENT

This work was supported by project of Chongqing Natural Science Foundation (No. CSTB2022NSCQ-MSX1298) and project of Research on Science and Technology of Chongqing Municipal Education Commission (No. KJZD-K202101901).

REFERENCES

[1] S. S. Gill et al., "Modern computing: Vision and challenges," Telematics and Informatics Reports, p. 100116, 2024.

[2] V. Hayyolalam, B. Pourghebleh, A. A. P. Kazem, and A. Ghaffari, "Exploring the state-of-the-art service composition approaches in cloud manufacturing systems to enhance upcoming techniques," The International Journal of Advanced Manufacturing Technology, vol. 105, no. 1-4, pp. 471-498, 2019.

[3] V. Hayyolalam, B. Pourghebleh, M. R. Chehrehzad, and A. A. Pourhaji Kazem, "Single - objective service composition methods in cloud manufacturing systems: Recent techniques, classification, and future trends," Concurrency and Computation: Practice and Experience, vol. 34, no. 5, p. e6698, 2022.

[4] J. Zou, K. Wang, K. Zhang, and M. Kassim, "Perspective of virtual machine consolidation in cloud computing: a systematic survey," Telecommunication Systems, pp. 1-29, 2024.

[5] H. Wu, "Black widow optimization algorithm for efficient task assignment in cloud computing," Journal of Engineering and Applied Science, vol. 71, no. 1, p. 139, 2024.

[6] J. Alonso et al., "Understanding the challenges and novel architectural models of multi-cloud native applications–a systematic literature review," Journal of Cloud Computing, vol. 12, no. 1, p. 6, 2023.

[7] D. Tohanean and S.-G. Toma, "The Impact of Cloud Systems on Enhancing Organizational Performance through Innovative Business Models in the Digitalization Era," in Proceedings of the International Conference on Business Excellence, 2024, vol. 18, no. 1: Sciendo, pp. 3568-3577.

[8] J. DesLauriers, J. Kovacs, T. Kiss, A. Stork, S. P. Serna, and A. Ullah, "Automated generation of deployment descriptors for managing microservices-based applications in the cloud to edge continuum," Future Generation Computer Systems, vol. 166, p. 107628, 2025.

[9] J. Lei, Q. Wu, and J. Xu, "Privacy and security-aware workflow scheduling in a hybrid cloud," Future Generation Computer Systems, vol. 131, pp. 269-278, 2022.

[10] J. L. Schaefer et al., "A framework for diagnosis and management of development and implementation of cloud-based energy communities-Energy cloud communities," Energy, vol. 276, p. 127420, 2023.

[11] F. K. Parast, C. Sindhav, S. Nikam, H. I. Yekta, K. B. Kent, and S. Hakak, "Cloud computing security: A survey of service-based models," Computers & Security, vol. 114, p. 102580, 2022.

[12] M. Barakat, R. A. Saeed, and S. Edam, "A Comparative Study on Cloud and Edgeb Computing: A Survey on Current Research Activities and Applications," in 2023 IEEE 3rd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA), 2023: IEEE, pp. 679-684.

[13] A. K. Samha, "Strategies for efficient resource management in federated cloud environments supporting Infrastructure as a Service (IaaS)," Journal of Engineering Research, vol. 12, no. 2, pp. 101-114, 2024.

[14] S. Aleem, R. Batool, S. Alkobaisi, F. Ahmed, and A. Khattak, "SaaS Application Maturity Assessment Model," IEEE Access, 2024.

[15] H. U. Khan, F. Ali, and S. Nazir, "Systematic analysis of software development in cloud computing perceptions," Journal of Software: Evolution and Process, vol. 36, no. 2, p. e2485, 2024.

[16] C. Li, M. Song, M. Zhang, and Y. Luo, "Effective replica management for improving reliability and availability in edge-cloud computing environment," Journal of Parallel and Distributed Computing, vol. 143, pp. 107-128, 2020.

[17] A. Azadi and M. Momayez, "Review on Constitutive Model for Simulation of Weak Rock Mass," Geotechnics, vol. 4, no. 3, pp. 872-892, 2024, doi: https://doi.org/10.3390/geotechnics4030045.

[18] M. A. Nezafat Tabalvandani, M. Hosseini Shirvani, and H. Motameni, "Reliability-aware web service composition with cost minimization perspective: a multi-objective particle swarm optimization model in multi-cloud scenarios," Soft Computing, vol. 28, no. 6, pp. 5173-5196, 2024.

[19] W. Ma and H. Xu, "Skyline-enhanced deep reinforcement learning approach for energy-efficient and QoS-guaranteed multi-cloud service composition," Applied Sciences, vol. 13, no. 11, p. 6826, 2023.

[20] B. Pourghebleh, A. A. Anvigh, A. R. Ramtin, and B. Mohammadi, "The importance of nature-inspired meta-heuristic algorithms for solving virtual machine consolidation problem in cloud environments," Cluster Computing, pp. 1-24, 2021.

[21] M. B. Karimi, A. Isazadeh, and A. M. Rahmani, "QoS-aware service composition in cloud computing using data mining techniques and genetic algorithm," The Journal of Supercomputing, vol. 73, pp. 1387-1415, 2017.

[22] S. S. Sefati and S. Halunga, "A hybrid service selection and composition for cloud computing using the adaptive penalty function in genetic and artificial bee colony algorithm," Sensors, vol. 22, no. 13, p. 4873, 2022.

[23] M. Wang, "A new QoS-aware service composition technique in cloud computing using capuchin search algorithm," Journal of Intelligent & Fuzzy Systems, no. Preprint, pp. 1-12, 2023.

[24] K. Zanbouri and N. Jafari Navimipour, "A cloud service composition method using a trust - based clustering algorithm and honeybee mating optimization algorithm," International Journal of Communication Systems, vol. 33, no. 5, p. e4259, 2020.

[25] X. Liu, "Hybrid Integrated Aquila Optimizer for Efficient Service Composition with Quality of Service Guarantees in Cloud Computing," International Journal of Advanced Computer Science and Applications, vol. 14, no. 10, 2023.

[26] L. Bei, L. Wenlin, S. Xin, and X. Xibin, "An improved ACO based service composition algorithm in multi-cloud networks," Journal of Cloud Computing, vol. 13, no. 1, p. 17, 2024.

[27] M. Shehab, L. Abualigah, H. Al Hamad, H. Alabool, M. Alshinwan, and A. M. Khasawneh, "Moth–flame optimization algorithm: variants and applications," Neural Computing and Applications, vol. 32, no. 14, pp. 9859-9884, 2020.