Integrating the Firefly Algorithm with ECOLIFE

1. Introduction

The ECOLIFE framework is designed to address the environmental impact of serverless computing by minimizing both embodied and operational carbon emissions in datacenter environments. By leveraging heterogeneous hardware resources, ECOLIFE aims to reduce the overall carbon footprint while maintaining efficient performance. The primary challenge in this endeavor is balancing the energy efficiency of newer hardware with the lower embodied carbon cost of older hardware, all while adapting dynamically to fluctuating workloads and carbon intensity levels of the power grid [1].

To tackle this challenge, ECOLIFE employs advanced optimization techniques to allocate serverless functions across different hardware resources intelligently. Initially, ECOLIFE used Dynamic Particle Swarm Optimization (DPSO) to adapt scheduling decisions based on real-time workload patterns and carbon metrics. However, introducing the Firefly Algorithm (FA) provides a new approach, enhancing ECOLIFE's ability to explore diverse solutions and dynamically respond to highly variable invocation patterns.

This document provides an overview of the ECOLIFE framework, detailing the integration of the FA. It includes a comparative analysis of DPSO and FA, highlighting their strengths and use cases within ECOLIFE. By leveraging both algorithms, ECOLIFE aims to optimize function scheduling with greater robustness, flexibility, and sustainability.

2. Problem Statement

In serverless computing environments, functions are dynamically invoked and executed on cloud servers, which typically retain functions in memory to reduce cold start latency. However, this "always-on" approach leads to substantial carbon emissions, as resources are kept active continuously to avoid cold starts, regardless of the actual demand. The scheduling problem in ECOLIFE seeks to address the following challenges:

- Balancing Embodied and Operational Carbon: Each server in a datacenter has a distinct carbon footprint. Older servers, though energy-inefficient, have lower embodied carbon costs due to their extended lifespans, whereas newer servers are more energy-efficient but carry a higher embodied carbon. The goal is to balance these two carbon sources by strategically assigning workloads across hardware generations to minimize total carbon impact.
- Minimizing Cold Starts with Resource Constraints: Serverless functions experience delays if not actively retained in memory, resulting in cold starts. Keeping functions warm mitigates cold start times and increases operational carbon emissions. Effective scheduling must find an optimal trade-off, prioritizing critical functions for warm starts on suitable hardware while adhering to memory constraints.
- Responding to Dynamic Invocation Patterns and Carbon Intensity: Invocation patterns in serverless computing are highly variable, with workloads fluctuating unpredictably. Additionally, the carbon intensity of the power grid changes over time and is influenced by renewable energy availability. A robust scheduling solution must dynamically adapt

to these variations, adjusting function placements and real-time hardware allocations to optimize performance and sustainability.

Therefore, the scheduling problem in ECOLIFE involves finding an optimal configuration for function placement and memory management that minimizes both response times (cold starts) and the overall carbon footprint. This must be achieved across a heterogeneous mix of hardware generations, considering invocation patterns, carbon intensity fluctuations, and hardware characteristics. The solution to this scheduling problem requires an adaptive, intelligent approach that balances performance and carbon impact in real-time.

3. Firefly Algorithm

The FA is a metaheuristic optimization algorithm inspired by fireflies' natural behavior, specifically their attraction to each other based on light intensity. The algorithm proposed by Xin-She Yang is designed for solving complex optimization problems by mimicking how fireflies communicate and are attracted to one another through bioluminescence [2].

In the FA, each firefly represents a potential solution to the optimization problem, and its "brightness" corresponds to the fitness value of that solution. Fireflies are attracted to brighter (better) solutions, allowing them to explore the search space and move toward optimal solutions over time. The main principles governing the FA are as follows:

3.2. Key Components of the Firefly Algorithm

a. Attractiveness: The degree of attraction between two fireflies depends on their brightness (fitness) and the distance between them. Brighter fireflies (those with better solutions) attract other fireflies, guiding them toward better areas in the search space.

b. Brightness (Fitness): Each firefly’s brightness is determined by the objective function (fitness function) of the optimization problem. In ECOLIFE, for instance, the brightness is based on a combined metric of service time and carbon footprint, where brighter fireflies indicate solutions with lower service time and carbon cost.

c. Distance and Movement: The attractiveness between two fireflies decreases as their distance increases, modeled using an exponential decay function. Firefly i will move towards firefly j if firefly j is brighter, with the attractiveness computed as:

$$\beta = \beta_0 \cdot e^{-\gamma r_{ij}^2}$$

Where:

- β_0: Base attractiveness.
- γ: Light absorption coefficient, controlling the effect of distance.
- r_{ij}: Euclidean distance between fireflies i and j.

d. Randomness: To maintain diversity and avoid local minima, each firefly’s movement includes a randomization factor:

$$x_i = x_i + \beta \cdot (x_j - x_i) + \alpha \cdot (rand - 0.5)$$

Where:

- x_i: Position of firefly iii.
- $\beta \cdot (x_j - x_i)$ Attraction towards a brighter firefly.
- α: Randomness factor (step size).
- $rand$: A random number in the range [0, 1].

3.3. Algorithm Workflow

1. Initialization: The firefly population is initialized with random solutions across the search space.
2. Fitness Evaluation: Each firefly's brightness (fitness) is calculated based on the objective function.
3. Firefly Movement: For each pair of fireflies, the less bright firefly moves towards the brighter one based on attractiveness and distance. Randomness is introduced to ensure diversity in the search process.
4. Iteration: Steps 2 and 3 are repeated for a set number of iterations or until a stopping criterion is met, such as a convergence threshold or maximum number of iterations.
5. Convergence: Fireflies cluster around the brightest (most optimal) regions over successive iterations, converging on the best solution.

4. Formulation of the Firefly Algorithm for the ECOLIFE Problem

The FA was adapted to solve ECOLIFE's scheduling problem by finding optimal configurations that balance service time and carbon emissions. Each firefly in the algorithm represents a potential solution, or scheduling configuration, characterized by the function's server placement and keep-alive time. Each firefly's brightness (fitness) corresponds to how well that configuration minimizes service time and carbon footprint.

4.1. Objective Function

The objective of the FA in ECOLIFE is to minimize a combined metric of service time and carbon footprint. The fitness (brightness) f(x) of each firefly x is given by:

$$f(x) = \lambda \cdot T(x) + (1 - \lambda) \cdot C(x)$$

Where:

- $T(x)$: Expected service time for the configuration x, which includes both cold and warm start
- $C(x)$: Expected carbon emissions, accounting for both embodied and operational carbon costs.
- λ: Weighting parameter for balancing service time and carbon emissions.

4.2. Decision Variables

1. Server Selection (ka_loc): Binary decision variable where:

   a. $ka\_loc = 0$: Allocate function to the older, less energy-efficient server with a lower embodied carbon.
   b. $ka\_loc = 1$: Allocate function to the newer, more energy-efficient server with a higher embodied carbon.
2. Keep-Alive Time **(**ka_loc**)**: Continuous variable representing the retention time for each function in memory to minimize cold starts, determined from a set of possible values kat_times.

Each firefly x has a position (ka_loc, kat), which defines its scheduling configuration.

4.3. Constraints

1. Memory Constraints:
$$\sum_i mem_i \cdot x_{ij} \le Mem_j \ \forall j$$
where mem_i is the memory requirement for function i and Mem_j is the memory capacity of server j.
2. Function Execution: Each function must be allocated to exactly one server:
$$\sum_j x_{ij} = 1 \ \forall i$$
3. Cold Start Probability Constraint: To minimize cold starts, the keep-alive time should match or exceed the average interval between invocations:
$$P_{cold}(i) \approx 0 \quad if \quad kat \ge avg\_interval(i)$$
Where $P_{cold}(i)$ is the probability of a cold start for function i.

4.4. Firefly Movement and Optimization Process

1. Fitness Calculation (Brightness): Each firefly's brightness f(x) is calculated using the combined service time and carbon cost defined above. Brighter fireflies represent configurations with lower service times and carbon footprints.

2. Attraction Mechanism: The attractiveness β of a firefly j to another firefly i is based on their brightness and distance:
$$\beta = \beta_0 \cdot e^{-\gamma r_{ij}^2}$$
Where:
   - β_0: Base attractiveness.
   - γ: Light absorption coefficient.
   - r_{ij}: Distance between fireflies i and j, calculated based on the difference in their (ka_loc, kat) values.

3. Movement Rule: If firefly i is less bright than firefly j, it will move towards j according to:
$$x_i = x_i + \beta \cdot (x_j - x_i) + \alpha \cdot (rand - 0.5)$$

Where:

1. x_i: Position of firefly i.
2. α: The randomness parameter adds diversity to the search and prevents premature convergence.
3. $rand$: Random number in the range [0, 1], introducing variability.

4. Iteration and Convergence:
   a. Over multiple iterations, fireflies update their positions based on their interactions, with each firefly adjusting toward brighter (better) configurations.
   b. After the set number of iterations, the brightest firefly (best solution) represents the optimal scheduling configuration regarding service time and carbon footprint for the ECOLIFE framework.

5. Firefly Algorithm for ECOLIFE

`firefly.py` contains the FA implementation specifically adapted for the EcoLife project. The algorithm optimizes scheduling and resource allocation to balance carbon footprint and service time for serverless computing functions. Each firefly represents a potential solution for scheduling, characterized by parameters for execution on two types of servers with distinct carbon and service time profiles.

5.1.Class Structure and Attributes

- **`__init__(…)`**: Initializes the algorithm parameters, target server pairs, and environmental variables.
  - `params`: A dictionary of algorithm parameters (population size, alpha, beta, gamma).
  - `server_pair`: List containing the two types of servers (old and new).
  - `function_name`: The name of the function being optimized.
  - `ci_avg`: Average carbon intensity of the grid.

The class initializes the population of fireflies based on the population_size parameter and assigns each firefly an initial brightness value (fitness).

- **`initialize_population(self)`**: This method creates an initial population of fireflies with random configurations and returns an array of firefly positions.
- **`fitness(self, firefly_position, ci, interval)`**: The `fitness` function calculates a firefly's fitness (brightness), combining service time and carbon footprint based on configuration. The `ci` and `interval` allow real-time adjustment based on carbon intensity and invocation frequency.
- **`prob_cold(self, interval, kat)`**: Computes the probabilities of cold and warm starts given invocation intervals and keep-alive time (`kat`).
- **`move_firefly(self, i, j)`**: Moves firefly `i` towards firefly `j` if `j` has a better fitness value. Uses attractiveness and random movement factors (`alpha`, `beta`, `gamma`) to determine the extent of movement.

main(self, ci, interval): Calculates fitness for each firefly, compares their brightness, and updates positions accordingly and returns the best firefly position after updating all fireflies.

6. Firefly Optimizer Integration in ECOLIFE

The `ff_opt.py` file implements the FA as an optimization technique within the ECOLIFE framework, allowing it to balance service time and carbon emissions in serverless computing. The `optimizer_FF` class manages the FA's execution to determine optimal scheduling configurations (server choice and keep-alive time) for functions running on a heterogeneous server infrastructure.

6.1.Class Structure and Attributes

- **__init__(…):** Initializes the optimizer, setting parameters such as window size, memory limits, and carbon intensity.
    - `Traces`: Invocation traces for each function.
    - `trace_function_names`: Names of the functions being optimized.
    - `server_pair`: List of two server types (old and new) with different energy efficiencies.
    - `kat_time`: List of keep-alive times to choose from, to reduce cold starts.
    - `st_lambda`: Weight parameter for balancing service time and carbon emissions.
    - `carbon_intensity`: List of carbon intensities over time for calculating operational carbon.
    - `window_size`: Window size for invocation history to calculate invocation patterns.
    - `ci_avg`: Average carbon intensity used as a baseline.
    - `ff_size`: Population size for the FA.
    - `ff_param`: Dictionary of FA-specific parameters (`alpha`, `beta`, `gamma`), controlling randomness, attraction, and light absorption in firefly movement.
    - `region and interval`: Additional parameters for region-specific data and optimization time interval.
- **Optimize (self)**: This is the main optimization loop, iterating over time intervals, tracking invocation intervals for functions, and coordinating scheduling decisions.
- **check_warm_pool_expiry(self, ...)**: Checks and clears expired functions from warm pools, recalculating carbon costs based on active time.
- **apply_decision(self, decision, i, j, old_decision, new_decision)**: Applies the best decision found by the Firefly Algorithm for each function at a given interval.
- **check_memory_and_adjust(self, ...)**: Manages memory constraints, adjusting warm pool contents and discarding functions if limits are exceeded.
- **save_results(self, result_carbon, result_st)**: Saves the final carbon footprint and service time data to output JSON files.

7. Comparison of DPSO and Firefly Algorithms in ECOLIFE

7.1. Overview of DPSO and Firefly Algorithms

- DPSO: DPSO is a variant of the standard PSO, designed to dynamically adjust parameters based on environmental changes, like function invocation patterns and carbon intensity in ECOLIFE. Each particle in DPSO represents a possible scheduling configuration, and particles adjust based on their position, velocity, and the best-known positions within the swarm.
- FA: The FA is inspired by fireflies' attraction to each other based on brightness, which corresponds to fitness. Fireflies move towards brighter (better) solutions, adjusting based on attractiveness, distance, and a randomness component. FA is well-suited for exploring diverse solutions and escaping local minima.

7.2. Comparison of Key Characteristics

Feature	DPSO	Firefly Algorithm
Inspiration	Social behavior of particles	Light attraction among fireflies
Optimization Approach	Velocity and position updates	Attraction and random movements
Parameter Adaptability	Dynamic adjustment of inertia and learning rates based on invocation patterns and carbon intensity	Fixed or slowly adaptive alpha, beta, and gamma values
Handling of Past Invocations	Uses past_interval to update velocity and position to avoid repeated cold starts and balance workload	Uses cur_interval to calculate cold start probability and optimize warm pool configurations
Convergence Speed	Tends to converge quickly, with adjustments in direction and speed to focus on best-known solutions	Slower convergence, but provides better exploration of the solution space
Diversity of Solutions	Moderate; particles often converge to similar positions	High; fireflies are more dispersed, allowing for the exploration of varied solutions
Best Use Case in ECOLIFE	Useful when invocation patterns are relatively stable, allowing DPSO to converge on optimal solutions	Beneficial when invocation patterns vary frequently, as FA can explore a broader range of configurations

7.3. Advantages of Firefly Over DPSO in ECOLIFE

- Better Exploration of Solution Space**:** Firefly's randomness and attraction mechanism provide diversity, enabling the algorithm to explore a broader range of configurations. This helps ECOLIFE adapt more flexibly to varying invocation patterns and carbon intensity levels, as fireflies can more easily escape local minima than DPSO particles.
- Robustness in Dynamic Environments: Firefly's reliance on individual fitness (brightness) rather than the global best solution allows it to perform better in highly dynamic environments, where invocation patterns and server conditions change

frequently. DPSO can struggle with rapid environmental shifts as particles converge towards a single global best, which may become suboptimal.

- Suitability for Avoiding Cold Starts: By calculating cold start probabilities within the `fitness` function, Firefly can make adaptive decisions on prioritizing warm starts across different servers. DPSO while effective, may require more tuning to dynamically account for cold start penalties, especially with varying invocation rates.
- Better Handling of Heterogeneous Hardware: The FA's gradual movement toward optimal configurations provides a smoother adaptation to hardware generation differences in ECOLIFE. Firefly's adaptability can more effectively balance old and new servers, optimizing service time and carbon footprint without the risk of premature convergence.

7.4. Advantages of DPSO Over Firefly in ECOLIFE

- Faster Convergence on Stable Patterns: DPSO is well-suited for relatively stable invocation patterns, as it can converge faster to optimal configurations by leveraging personal and global best positions. In scenarios where invocation and carbon intensity changes are less frequent, DPSO may achieve efficient configurations more quickly.
- Easier to Tune for Real-Time Adjustments: DPSO's dynamic adjustments of inertia and learning rates allow it to be more responsive in real-time scheduling. When integrated with ECOLIFE's perception-response mechanism, DPSO can efficiently respond to smaller variations without significantly increasing computational overhead.

7.5. Choosing Between DPSO and Firefly in ECOLIFE

ECOLIFE benefits from both DPSO and FA depending on the specific scheduling demands and serverless computing patterns:

- FA: This algorithm is recommended for deployments with high invocation variability or unpredictable carbon intensity changes. FA is ideal for exploring diverse solutions, making it more robust against fluctuating workloads and dynamic conditions.
- DPSO: Suitable for environments with more predictable, stable invocation patterns where rapid convergence to the optimal configuration is beneficial. DPSO's efficient parameter updates offer an advantage in steady-state conditions, optimizing service time and carbon footprint with lower computational costs.

References

[1] Y. Jiang, R.B. Roy, B. Li, D. Tiwari, EcoLife: Carbon-Aware Serverless Function Scheduling for Sustainable Computing, arXiv preprint arXiv:2409.02085, DOI (2024).
[2] X.-S. Yang, Firefly Algorithms for Multimodal Optimization, in: O. Watanabe, T. Zeugmann (Eds.) Stochastic Algorithms: Foundations and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 169-178.