International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING

ISSN:2147-6799 www.ijisae.org **Original Research Paper**

# Optimizing Resource Allocation in Big Data Scheduling Architectures Using a Tuned Firefly Algorithm

**Rohit Kumar Verma[1], Sukhvir Singh[2]**

**Submitted:** 25/01/2024 **Revised:** 03/03/2024 **Accepted:** 11/03/2024

**Abstract*:*** In contemporary research architectures, the efficient allocation of computational resources is paramount to meet the dynamic and diverse demands of tasks and applications. Traditional resource allocation policies often struggle with adaptability to varying workloads and underutilize historical allocation data. To address these limitations and enhance Quality of Service (QoS) in research environments, a rank-based data node allocation system is imperative. This research introduces a novel firefly-based Swarm Intelligence (SI) algorithm, meticulously tuned to strike a balance between resource overutilization and underutilization while upholding essential Service Level Agreements (SLAs). The contributions of this work encompass algorithm design, the delicate equilibrium between resource usage and SLA adherence, the establishment of a robust evaluation framework, and systematic comparisons with state-of-the-art resource allocation methods. Furthermore, this study takes into account different load factors, providing a comprehensive analysis of resource allocation efficiency across varying workloads. The algorithm's performance is evaluated based on QoS metrics, including power consumption and SLA adherence, and compared with existing allocation methods. The results underscore the algorithm's superiority in enhancing resource allocation efficiency and SLA adherence in Big Data Scheduling Architectures.

***Keywords:*** *Resource Allocation, Swarm Intelligence, Quality of Service, Service Level Agreements, Big Data Scheduling, Optimization, Efficiency, Load Factors, Computational Resources, Dynamic Workloads.*

## 1. Introduction

In today's digitally driven world, we are experiencing an exponential explosion of data generation and consumption, a phenomenon commonly referred to as "big data." The era of big data is marked by an overwhelming influx of data from various sources, including social media, sensors, mobile devices, and the Internet of Things (IoT) [1]. These data streams have evolved into massive reservoirs of information, with volumes ranging from terabytes to zettabytes, characterized by the relentless flow of data, and encompassing a wide spectrum of formats and types. Big data, with its tremendous potential, offers unprecedented opportunities for organizations across diverse sectors [2, 3]. It promises transformative insights, informed decision-making, and ground-breaking innovations. However, the journey from data to actionable insights is laden with formidable challenges, and one of the most fundamental among them is the effective allocation of resources in the context of big data due to diverse application and user base seeking to use big data architecture [4, 5].

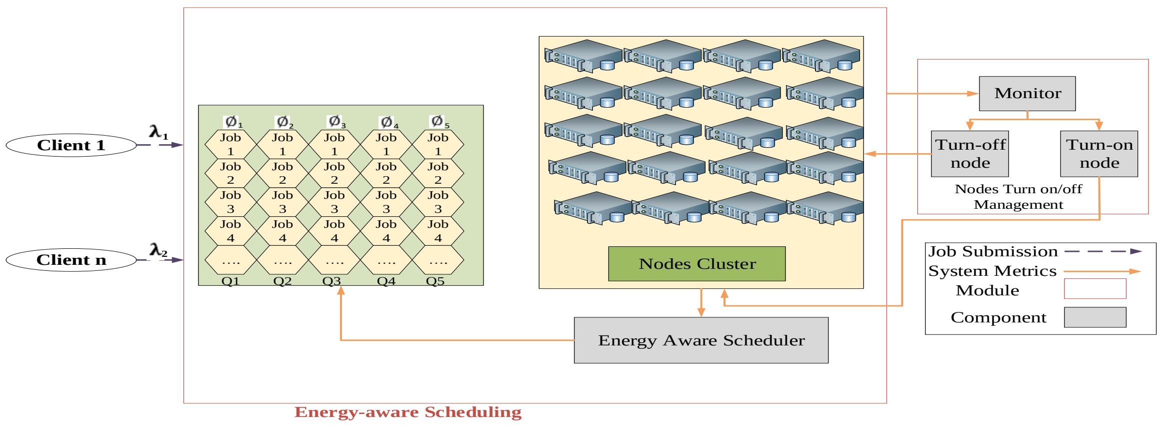


**Fig 1:** Job Scheduler in Big Data

Resource allocation involves the judicious distribution and management of computing resources such as processing capacity, memory, storage, and network bandwidth to enable the efficient processing and analysis[6, 7] of big data. This crucial facet of big data analytics ensures that computational tasks are executed

*1*PhD Research Scholar, Department of Computer Science, Himachal Pradesh University, Shimla, India*
*2Assistant Professor, Department of Computer Science, Himachal Pradesh University, Shimla, India*
** Corresponding Author Email: pverma1542015@gmail.com*

seamlessly and within defined time frames [8].

Power aware allocation has been a part of Big data scheduling architectures since 2010 and different power consumption models have been introduced since then. The power consumption of the scheduling process depends upon two factors namely the static consumption and the dynamic consumption defined in eq(1) and (2).

$$SPc = \frac{J_{R_p}}{Exc_p} * \mathrm{Uc} \tag{1}$$

$$DP_C = \frac{J_C}{Exc_p} * \mathrm{Uc} \tag{2}$$

Where $J_R$ is the remaining job $in\ p^{th}$ buffer $Exc_p$ is the execution capacity of the executer for particular buffer p and Uc is the unit cost of execution in watt. $J_C$ is the current job list. The scheduling environment can be defined as follows.

**Definition 1:** The big data scheduler (BDS) is a scheduler that handles users $U = \{u1, u2, .. un\}$ with multiple locations $L = \{l1, l2, l3 \dots . ln\}$ with varying job request with different priorities P={p1,p2,….pn} with the minimization of the overall power consumption PC={pc1,pc2….pcn} so that neither the system is underutilized or overutilized and minimum Service Level Agreement(SLA) is met.

$$\text{Minimize } \sum_{i=1}^{n} \frac{\mathrm{PC_i}}{\mathrm{SU_i}} \times \frac{\mathrm{SLA_i}}{\mathrm{p_i}} \tag{3}$$

**Lemma**: In the context of the Big Data Scheduler (BDS), the system should achieve a balanced allocation of resources to meet user demands while minimizing overall power consumption. This requires that the system utilization (*SU_i*) for each user *u_i* satisfies the constraint:

$$MinSU < SU_i < MaxSU \tag{4}$$

Where:

*SU_i* represents the system utilization for user *u_i*.

*SLA_i* signifies the Service Level Agreement requirement for user *u_i*.

The SLA violation(SLA-V) is calculated in terms of power consumption as defined in eq(5) as follows.

$$SLA - V = \mathcal{Z}/C \tag{5}$$

Where Z is the total number of occurrences when the data node is consuming more energy than the desired energy threshold and C is the total number of allocations in the allocation table. The allocation table also contains the details of the allocation of the migrated users as the migrated users have to be reallocated as well.

### 1.1. Problem Statement

In contemporary research architectures, the efficient allocation of computational resources plays a pivotal role in meeting the diverse and dynamic demands of tasks and applications. Traditional resource allocation policies, such as minimum cost allocation, often lack adaptability to varying workloads and do not utilize historical allocation data for improved decision-making. To address these limitations and enhance the Quality of Service (QoS) in research environments, there is a pressing need to develop a rank-based data node allocation system.

The evolution of swarm intelligence has significantly influenced rank generation in Big Data Scheduling Architectures. Inspired by the collective behaviours of social organisms, swarm-inspired algorithms have introduced dynamic, self-organizing, and adaptive mechanisms for creating rankings within these architectures [9–12]. These algorithms enable data nodes to autonomously assess their performance, adapt to changing conditions, and refine their rankings based on real-time feedback. This evolution has improved resource allocation, enhanced Quality of Service (QoS), minimized power consumption, and ensured efficient resource utilization in the ever-evolving landscape of big data processing, all while promoting scalability and resilience in large-scale architectures. Swarm intelligence continues to play a pivotal role in shaping the future of distributed computing and data analytics [10, 13, 14].

### 1.2. Contributions of the Research

This research explores the design of a novel firefly-based (SI) algorithm, meticulously tuned to strike a balance between resource overutilization and underutilization, all while upholding essential (SLAs). The contributions can be listed as follows.

- "Design and development of a firefly-based Swarm Intelligence (SI) algorithm with a tuned fitness function for optimized resource allocation."
- "A delicate balance between resource overutilization and underutilization while preserving (SLAs)."
- "Establishment of a robust evaluation framework for pre and post-implementation assessment of the proposed algorithm."
- "Demonstration of the algorithm's superiority through systematic comparisons with state-of-the-art resource allocation methods."
- "Enhanced resource allocation efficiency and SLA adherence in Big Data Scheduling Architectures."

The rest of the paper is organized in the following manner. Section2 represents the literature survey that mainly focuses on the SI based algorithm architecture that has been applied in the same context. Section 3 presents the proposed work algorithm and its illustration and Section 4 presents the evaluation of the proposed work and a comparison of proposed work with other state of art algorithms. The paper is concluded in section 5.

## 2. Related Work

The literature review includes a thorough examination of several optimisation methodologies in the context of big-data cloud computing. Numerous studies have been presented in the research community to address the challenges of resource scheduling for big data in cloud environment for efficient management and deployment of various applications [15–17]. In the present work, researcher seeks to provide a comprehensive overview of the approaches used in the quest of efficient resource utilisation within big data scheduling structures by assessing the unique contributions and insights contained within this publication. To start with Wang et al. (2018) proposed a hybrid multi-objective firefly method optimised for big data. To address the complexity inherent in huge data processing, the study used both firefly algorithms and multi-objective optimisation techniques. The authors used popular programming languages and frameworks to develop their algorithm, emphasising its versatility in dealing with various optimisation targets. The algorithm's performance review indicated its ability to achieve many goals at the same time. The study did, however, admit the need for additional research into parameter adjustment for maximum performance [18]. Li et al. (2019) suggested a fault-tolerant replica management policy that is energy-efficient and designed for edge-cloud scenarios. To guarantee optimal resource utilisation, the study focused on meeting both deadline and budget restrictions. To create the policy, the authors used mathematical modelling and optimisation techniques. They used simulation and validation techniques to demonstrate the usefulness of the proposed policy in improving energy efficiency and fault tolerance. However, drawbacks include the mathematical model's reliance on certain assumptions and the need for additional empirical validation in real-world edge-cloud environments [8]. Jayaraj (2019) employed a nature-inspired firefly method with K-means clustering to optimise processes in big-data cloud centres. K-means clustering was used to add an extra layer of optimisation. When compared to traditional approaches, performance evaluation indicated enhanced process optimisation. The study did, however, highlight the algorithm's sensitivity to parameter values, indicating the necessity for careful adjustment [19]. Senthilkumar and Ilango (2020) developed an energy-aware work scheduling approach in big data environments based on a hybrid firefly-genetic algorithm. The research used evolutionary algorithms to improve the exploration-exploitation balance during the optimisation process. The authors used simulation tools to develop the algorithm and evaluate its performance. When compared to standard methods, the hybrid approach revealed enhanced energy-aware job scheduling. However, the study acknowledged the difficulty in determining ideal parameter values for various settings [7].

Alharbi et al. (2021) did a detailed evaluation of swarm intelligence algorithms used for cloud computing scheduling and optimisation. The study was used to synthesise existing knowledge in the topic. A thorough literature review was used by the authors to evaluate the performance of various swarm intelligence strategies. The review emphasised the advantages and disadvantages of various algorithms, providing significant insights for researchers and practitioners. However, the study emphasised the field's dynamic character, requiring constant updates to reflect developing trends [20]. Talwani et al. (2022) proposed a machine-learning-based solution for cloud computing virtual machine allocation and migration. The study used machine learning techniques to improve decision-making in virtual machine allocation and migration. For algorithm construction, the authors used prominent machine learning frameworks and languages. The examination of performance revealed enhanced accuracy and efficiency in virtual machine administration. The study did acknowledge the difficulty of training the machine learning model using representative data, as well as the possible impact of changing workload patterns on model performance [21].

Oduwole et al. (2022) proposed an improved load balancing technique tailored to big-data cloud computing systems. To increase resource use, the study used load balancing methods and optimisation approaches. The authors evaluated the performance of the suggested technique using simulation tools. When compared to previous methods, the enhanced load balancing technology revealed improved load distribution and system performance. However, the study acknowledged the importance of real-world validation and taking into account a variety of workload scenarios [22]. Wang et al. (2023) addressed the issues of distributed computing in cloud data centre networks by utilising a comprehensive methodology that included effective data persistence and data division techniques. They intended to improve the overall performance of distributed computing in cloud data centre networks by employing innovative data management technologies. Their investigation's findings highlighted the usefulness of the proposed strategies in optimising data management, hence boosting the efficiency of distributed computing operations. The study methodically analysed many metrics to evaluate the performance of their technique, providing light on the delicate features of data persistence and partition in the cloud environment. However, the study acknowledged some limitations, emphasising the need for further investigation and refinement to address specific issues connected with the proposed techniques [23]. Kaur et al. (2023) proposed an algorithmic approach to cloud computing virtual machine migration that used the modified SESA algorithm. To establish an effective migration method, the study used algorithmic design and execution techniques. The algorithm's effectiveness in optimising virtual machine migration was demonstrated through performance evaluation. However, the study highlighted the difficulties in finding suitable migration parameters and the need for additional research in dynamic cloud settings [24]. Table 1 the literature review highlights the substantial advances made in optimising many aspects of big-data cloud computing. In tackling the issues connected with optimisation tasks, each study presents unique viewpoints, approaches, and concerns.

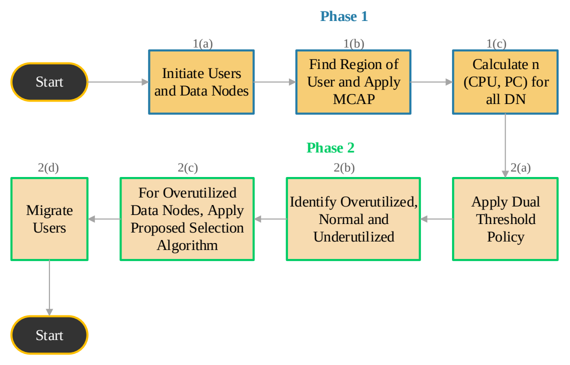
**Table 1:** Comparative Analysis of the Existing Studies

| Authors and Reference | Objective of the Study | Methodology and Techniques | Limitations and Observed Research Gaps |
|---|---|---|---|
| Li et al. [8] | Energy-efficient fault-tolerant replica management policy in edge-cloud | Mathematical modeling, Optimization techniques | The application of the mathematical model may be constrained by assumptions. Further empirical validation in real-world edge-cloud environments is necessary. |
| Wang et al. [18] | Hybrid multi-objective firefly algorithm for big data optimisation | Firefly algorithms, Multi-objective optimization | More investigation is required to optimise parameter values for optimal performance. |
| Jayaraj [19] | Process Optimisation of Big-Data Cloud Centre Using Nature-Inspired Firefly Algorithm and K-Means Clustering | Nature-inspired algorithms, K-Means clustering | The performance of the method is sensitive to parameter values, demanding careful customisation. |
| Senthilkumar and Ilango [7] | Energy-aware task scheduling in big data with hybrid firefly-GA | Hybrid firefly-genetic algorithm | The task of establishing appropriate parameter values for various contexts continues to be difficult. |
| Alharbi et al. [20] | A review of the adoption of swarm intelligence algorithms for scheduling and optimisation in cloud computing. | Literature review, Swarm intelligence algorithms | The effectiveness of the review is determined by the accuracy and timeliness of the material included. |
| Talwani et al. [21] | Machine-Learning-Based Approach for Virtual Machine Allocation and Migration | Machine learning techniques | The performance of the model is determined on the quality and representativeness of the training data. The model's response to changing workload patterns should be carefully considered. |
| Oduwole et al. [22] | Enhanced Load Balancing Technique for Big-data Cloud Computing Environments | Load balancing algorithms, Optimization techniques | Real-world validation is required to confirm the effectiveness of the enhanced load balancing technique. The approach may need to be tweaked for different workload scenarios. |
| Saurabh et al. [24] | With an Updated SESA Algorithm, an Algorithmic Approach to Virtual Machine Migration in Cloud Computing | Algorithmic design, improved SESA algorithm using Artificial Bee Colony | Determining effective migration parameters remains a challenge and further research is needed in a dynamic environment. |

In recent years, there has been an exponential increase in interest and research focused on optimising resource allocation and job scheduling in the realm of big data cloud computing. As the volume and complexity of data grows, so does the need for efficient techniques to manage and process this massive amount of data in cloud systems. Scholars like Mangalampalli et al. [25] have investigated work scheduling algorithms, emphasising the need of optimisation in delivering effective cloud computing outcomes. Furthermore, the trend towards optimisation solutions is seen in works such as Hassan et al.'s [26], which provides a cloud-fog framework for resource allocation based on the firefly algorithm, highlighting the need of optimisation in maintaining resource efficiency. Bacanin et al. [27] contribute to this trend by developing a modified firefly algorithm for process scheduling in cloud with an emphasis on flexibility to varied computing landscapes. Elmagzoub et al. [28] also conducted a survey on the use of swarm intelligence-based load balancing approaches in cloud computing, revealing the expanding use of swarm intelligence methodology to address complex optimisation difficulties. The convergence of these research demonstrates a clear and growing preference for optimisation methodologies, particularly those based on swarm intelligence, as a critical enabler for improving the performance and scalability of big data cloud computing systems.

## 3. Proposed Methodology

The proposed work is divided into two distinct phases as shown in Figure 2. The first phase is for the preliminary allocation of the user's resource demand to data nodes based on the Min Cost Allocation Policy (MCAP). The MCAP policy ensures that with minimum cost on the execution side, the SLA is complete. In the second phase, the proposed work uses proposed SI architecture to first calculate the overloaded and the underloaded data nodes. These data nodes will face migrations in terms of users so that they can be brought to a normalised state.



Phase 1
Start
1(a)
Initiate Users and Data Nodes
1(b)
Find Region of User and Apply MCAP
1(c)
Calculate n (CPU, PC) for all DN
Phase 2
2(a)
Apply Dual Threshold Policy
2(b)
Identify Overutilized, Normal and Underutilized
2(c)
For Overutilized Data Nodes, Apply Proposed Selection Algorithm
2(d)
Migrate Users
Start

**Fig 2:** Overall Work Architecture

### 3.1. The Preliminary Allocation via MCAP

The primary objective of the Preliminary Allocation phase, as facilitated by the MCAP algorithm to optimize the allocation of user data within a distributed storage system. This optimization process hinges on a multifaceted approach that takes into account the geographical regions of users and the proximity of data nodes, all while carefully considering the associated costs. At its core, the MCAP algorithm seeks to streamline the allocation process by identifying the region to which each user belongs. By doing so, it aims to enhance the overall convenience and efficiency of data retrieval for users. To achieve this, the algorithm leverages the geographical information associated with both users and data nodes, ensuring that users are allocated to the nearest available data node within their respective regions. However, in scenarios where multiple data nodes coexist within a single region, the MCAP algorithm goes beyond mere proximity considerations. It introduces a cost calculation mechanism that evaluates the potential expenses associated with data allocation decisions. This cost assessment takes into account various factors, including the user's utilization index, the expected delay, and the cost per unit of time for data migration.

**Algorithm 1:** MCAP

```
Function Allocate(active_users, user_region, user_cpu_demand, UI,
  dn_region, dn_cpu, b_p, D_ex, cost_per_unit_time):
  allocation_table ← empty table;
  for each user u in active_users do
      region ← user_region(u);
      demand_cpu ← user_cpu_demand(u);
      feasible_nodes ← data nodes in region;
      feasible_nodes ← filter nodes where dn_cpu > demand_cpu;
      if feasible_nodes not empty then
          costs ← calculate_costs(feasible_nodes, UI(u), b_p, D_ex,
            cost_per_unit_time);
          min_cost_index ← index of min(costs);
          allocation_table.add(u, feasible_nodes[min_cost_index],
            demand_cpu, costs[min_cost_index]);
          dn_cpu[feasible_nodes[min_cost_index]] ←
            dn_cpu[feasible_nodes[min_cost_index]] - demand_cpu;
      end
  end
```

The MCAP algorithm is designed to intelligently allocate user data to data nodes within a distributed storage system while optimizing for factors such as proximity, computational resources, and cost efficiency. It begins by initializing an empty data structure known as the **allocation table** to track the allocation of users to data nodes. Then, for each user in the set of active users, the algorithm proceeds with a series of steps. First, it identifies the geographical or logical region to which the user belongs, using the **user region(u)** function. This regional information is crucial because it helps determine which data nodes are in proximity to the user, enhancing data retrieval efficiency. Next, the algorithm calculates the CPU demand of the user using the **user CPU demand**, which quantifies the computational resources needed by the user for various data operations. It then identifies feasible data nodes within the same region as the user, as these are potential candidates for data allocation. To narrow down the options, the algorithm filters out data nodes with CPU capacities lower than the user's CPU demand, ensuring that only suitable data nodes are considered.
Crucially, MCAP goes beyond simple proximity considerations by introducing a cost calculation step. It calculates the costs associated with allocating the user to each of the remaining feasible data nodes. The overall cost of the allocation is calculated by adding the static power cost and the dynamic cost using equation (2) and (3).

### 3.2. Proposed scheme of balancing to optimize resource allocation

#### 3.2.1. Identification of overutilized and underutilized data nodes using dual threshold policy

After the initial allocation of user data to data nodes within the distributed storage system, a subsequent process is initiated to assess and categorize these data nodes based on their CPU utilization and power consumption. This categorization is achieved by applying a Dual Threshold Policy concept, which aims to classify data nodes into one of three distinct categories: "underutilized," "overutilized," and "neutral" data nodes.
The primary objective of this categorization process is to strike a balance between CPU utilization and power consumption, ensuring efficient resource allocation and optimal system

performance. A Dual Threshold Policy(DTP) is adopted that uses an upper threshold and a lower threshold as follows.

$$T_{UPPER} = \chi + m \quad (5)$$

$$T_{Lower} = \chi - m \quad (6)$$

Where $\chi$ is the normalised threshold for power consumption and CPU utilization and m is a set of threshold, $m \in \{10 - 30\}\%$[ref]

$$\chi = \frac{\sum_{a=1}^{TA} norm(PC) + norm(CPU)}{TA} \quad (7)$$

Where TA is total number of allocations attained via MCAP. The algorithm is designed to evaluate and categorize data nodes within a distributed storage system based on their resource utilization, with a specific focus on CPU utilization and power consumption. Its primary objective is to assess the efficiency of these data nodes and identify potential areas for resource optimization. The algorithm starts by collecting essential data, such as the total CPU and power allocation across all data nodes (represented as TCA and TPA) and determining the total number of data nodes in the system (TDN).

**Algorithm 2:** Datanode Utilization Analysis Algorithm with Custom Thresholds

```
Function AnalyzeUtilization(allocation_table, dn_cpu, delta):
    total_cpu_allocated ← sum of all allocated CPUs from
      allocation_table;
    total_data_nodes ← total number of data nodes;
    mean_cpu_utilization ← total_cpu_allocated / total_data_nodes;
    upper_threshold ← mean_cpu_utilization + delta;
    lower_threshold ← mean_cpu_utilization - delta;
    overutilized_nodes ← empty list;
    underutilized_nodes ← empty list;
    for each data node dn do
        total_allocated_cpu ← sum of allocated CPUs from
          allocation_table for data node dn;
        average_utilization ← total_allocated_cpu / dn_cpu(dn);
        if average_utilization > upper_threshold then
            overutilized_nodes.append(dn);
        end
        if average_utilization < lower_threshold then
            underutilized_nodes.append(dn);
        end
    end
    return overutilized_nodes, underutilized_nodes;
```

These initial metrics serve as the foundation for subsequent analysis. To gain an understanding of the overall resource utilization, the algorithm calculates the mean CPU utilization (MCA) and mean power consumption (MPA) by dividing the total allocated CPU and power by the total number of data nodes. These mean values provide a reference point for assessing the performance of individual data nodes. However, since data nodes may have different capacities, the algorithm normalizes the CPU utilization (NCA) and power consumption (NPA) of each data node. This normalization process involves dividing the allocated resources by the capacity of the respective data node (DN_C and DN_PC), ensuring a fair and equitable comparison among data nodes. The crux of the algorithm lies in the determination of thresholds. Upper and lower thresholds are calculated for both CPU utilization (UTC and LTC) and power consumption (UPC and LPC). These thresholds are established by adding and subtracting a predetermined delta value (DC for CPU and DP for power) to the normalized mean values. These thresholds serve as critical reference points for categorization. Data nodes are categorized based on whether their normalized CPU utilization and power consumption fall outside the established thresholds. Data nodes that exceed both the upper CPU and power thresholds are classified as "overutilized," indicating that they are consuming more resources than anticipated, potentially leading to inefficiencies or performance degradation. Conversely, data nodes that fall below both lower thresholds are labelled as "underutilized," suggesting that they have available resources that are not fully utilized, presenting an opportunity for resource optimization. Throughout this process, the algorithm maintains two distinct lists: "overutilized nodes" (ON) and "underutilized nodes" (UN). These lists store the data nodes that meet the respective categorizations, providing system administrators with clear insights into which nodes may require resource adjustments to enhance overall efficiency. In summary, this algorithm serves as a powerful tool for evaluating and categorizing data nodes within a distributed storage system, contributing to the optimization of resource allocation and the efficient operation of the infrastructure.

### 3.2.2. Selection of the users from overutilized data nodes

Once the data nodes have been identified as overutilized or underutilized, the objective is to neutralise both kind of data nodes. If the data node is underutilized, it is holding all the resources that are not getting used at all and hence in such a scenario, all the users will have to be migrated from this data node and the data node becomes a candidate where other users can be located. As the overutilized data node has more number of users to handle and that is why it is not able to use its resources properly, few of the users have to migrated to a less burden holding data node. Swarm Intelligence (SI) based algorithms are used for selecting users from overutilized data nodes because they draw inspiration from the collective behaviour of social animals or organisms, such as ants, bees, birds, or fireflies, to solve complex optimization problems. In the context of selecting users from overutilized data nodes, SI algorithms can help identify an optimal or near-optimal solution by mimicking the way these social creatures cooperate and coordinate to find the best path or solution in a decentralized and self-organizing manner.

One specific SI algorithm is the Firefly algorithm. The Firefly algorithm is named after the behavior of fireflies in nature. Fireflies use bioluminescence to communicate and attract mates. Similarly, in the Firefly algorithm, artificial "fireflies" represent potential solutions to an optimization problem. The proposed work views the Firefly algorithm with the following constraints.

For data nodes D, and a set of users U each data node has a certain utilization and power consumption level, the aim is to select a subset of users from U to be relocated to different data nodes in order to balance the utilization and power consumption across all data nodes. Let's define the problem mathematically:

- **Variables:**
  - Binary decision variables: Let $x_{ui}$ be a binary variable that takes the value 1 if user u from set U is selected to be relocated to datanode i from set D, and 0 otherwise.
- **Parameters:**
  - $d_i$: The current utilization level of datanode i.
  - $U\_max$ : The maximum number of users that can be relocated to any datanode.
  - $U_{min}$: The minimum number of users that should be relocated to any datanode to achieve a balanced load.

## 3.3. Objective Function:

The proposed firefly algorithm has multiple objective functions and they are illustrated as follows.

### 3.3.1. Neutralisation of the CPU utilization

Minimize the imbalance in data node utilization and power consumption. This can be represented as an objective function:

Minimize:

$$\sum_{i \in D} CPU_{U_i} - avgCPU_U \quad (8)$$

Here, the $avgCPU$ is the total utilization across all data nodes divided by the number of data nodes and $CPU_U$ is the cpu utilization for current data node

Ensure that each data node receives a balanced load:

$$U_{min} \le \frac{\sum_{u \in U} U_u}{U} \le U_{max} \tag{9}$$

### 3.3.2. Neutralisation of the PC

Minimize the imbalance in data node utilization and power consumption. This can be represented as an objective function:

Minimize:

$$\sum_{i \in D} PC_i - avgPC \tag{10}$$

Where avgPC is the average power consumption across all data nodes.

Ensure that each data node receives a power consumption

$$PC_{min} \le \frac{\sum_{u \in U} PC_u}{U} \le PC_{max} \tag{11}$$

## 3.4. Firefly Algorithm

The Firefly Algorithm is inspired by the flashing behavior of fireflies in nature. Fireflies are attracted to each other based on their brightness, and this concept is applied to optimization problems. In the algorithm, each solution (represented as a firefly) has a brightness value, and fireflies move towards brighter ones to find optimal or near-optimal solutions.

### 3.4.1. Mathematical Architecture and Steps

Here's a step-by-step breakdown of how the Firefly Algorithm can be applied to your problem:

- **Initialization:**
  - Initialize a population of fireflies, each representing a potential solution to the problem of user selection from overutilized data nodes.
  - Define a fitness function that quantifies how well a solution balances datanode utilization. Brighter fireflies have higher fitness values.
- **Attraction Index (Attractiveness):**
  - Calculate the attraction (attractiveness) between two fireflies (solutions) using an attractiveness function. The attractiveness between firefly i and firefly j can be represented as $\beta(e)$, where $e$ is the Euclidean distance between fireflies i and j.
  - The attractiveness function could be defined as follows:
    $$\beta(e) = \beta_0 \cdot e^{-\gamma r 2} \tag{11}$$
  - Here, $\beta_0$ and γ are algorithm-specific parameters and r is the distance between two fireflies.
- **Firefly Movement:**
  - Fireflies move towards other fireflies with higher attractiveness values. The movement of a firefly toward another firefly is governed by the following equation:
  - Move towards firefly j:
  - $\boldsymbol{x_i}(t+1) = \boldsymbol{x}_i(t) + \beta \exp^{-\gamma(xj(t)-xi(t))^2} \times r + \alpha \cdot (random\ step)$ (12)
  - In proposed context, this means that solutions (representing user configurations) that provide better load balancing will attract other solutions, causing them to adjust their user selections accordingly.
- **Update Brightness (Fitness):** After the movement step, update the brightness (fitness) of each firefly based on how well its associated solution balances data node utilization. A lower fitness value indicates better load balancing.
- **Iteration:** Repeat the attraction, movement, and brightness update steps for a predefined number of iterations or until a convergence criterion is met.
- **Solution Extraction:**
  - The final positions of the fireflies represent a set of users relocated to data nodes, providing a balanced load across data nodes. Select the solution (firefly) with the lowest fitness value as the optimized configuration.

### 3.4.2. Proposed Firefly

The "Proposed Firefly Algorithm (PFA)" is designed for the selection of users from overutilized data nodes. It employs a firefly-inspired selection procedure where the "brighter fireflies" correspond to users categorized under "neutral data nodes. The main goal of this algorithm is to intelligently select users from a group of overutilized data nodes while taking into account their CPU utilization and power consumption characteristics. It achieves this by drawing inspiration from the behaviour of fireflies in nature.

**Input Parameters:**

- Allocation Table (AT): This table contains information about the allocation status of users on various data nodes.
- Overutilized Data Node (OD): The algorithm targets a specific overutilized data node for user selection.
- Neutral Data Nodes (ND): These represent data nodes where users exhibit stable and neutral behavior.

**Algorithm 1** Proposed Firefly Algorithm (PFA) for User Selection

```
procedure PFA(Allocation Table (AT), Overutilized Data Node (OD), Neutral Data Nodes (ND))
    Output: U_List
    Brighter Firefly = $AT[ND,(CPU,PC)]$ ← Find the parameters of neutral data nodes
    $\max_{gen} = 1000,\ g = 0,\ AI = [],\ \alpha = 0.01,\ \beta = 1,\ \gamma = [0,1]$
    while $g \le \max_{gen}$ do
        for $i = 1$ to $AT[OD,(CPU,PC)]$ for all $k \in U$ do
            for $j = 1$ to $BF$ do
                $P_f = [U_i, U_{rindex}]$ ← Pairing of current fly with random other flies
                $W_1 = 0.7,\ W_2 = 0.3$    ▷ Weights for CPU and PC utilization
                $r_i = [P_f[p < \epsilon U, CPU] \times W_1 + P_f[k \in U, PC] \times W_2]$
                $r_j = BF[j, CPU] \times W_1 + BF[j, CPU] \times W_2]$
                $r = (r_i - r_j)$
                $ai = AI[i,k] + \beta \exp(-\gamma r^2) xr + \alpha\gamma$ ▷ Calculate attractiveness
                if $ai > AI[i,k]$ then
                    $AI[i,k] = ai$    ▷ Update AI if attractiveness is higher
                end if
            end for
        end for
        $g = g + 1$    ▷ Increment generation counter
    end while
    Choose Users with Max AI and Create U_List    ▷ Select users with the highest AI
    Return U_List
end procedure
```

**Initialization**: The algorithm begins by identifying the "Brighter Firefly." In the context of this algorithm, the Brighter Firefly refers to the data node that is categorized as a neutral data node and serves as a reference point for selecting users with similar characteristics. Several key parameters are set up:

- Maximum generations (max_gen): This parameter determines how many generations or iterations the algorithm will perform. It is set to 1000.
- Generation counter (g): Initialized to 0, this counter keeps track of the current generation.
- AI (Attractiveness Index) List: An empty list is created to store the calculated attractiveness values for users.
- Constants: Three constants are defined - alpha (0.01), beta (1), and gamma (a range between 0 and 1).

**Algorithm Main Loop**: The algorithm operates in a loop, where each iteration represents a generation. It continues to execute until the generation counter (g) reaches the maximum specified number of generations (max_gen). Within each generation, the algorithm proceeds as follows:

1. **User Selection**: It iterates through all the users on the overutilized data node (OD) for all users (k) in the set U.
2. **Pairing with Brighter Fireflies**: For each user (i), the algorithm pairs the user with random other users or "fireflies" to simulate the interaction between users. These fireflies represent users from neutral data nodes (ND).
3. **Weighted Evaluation**: The algorithm assigns weight factors, W_1 and W_2, to CPU utilization and power consumption, respectively. These weight factors reflect the importance or priority of these two characteristics in the user selection process. The algorithm then calculates r_i and r_j, representing the weighted evaluation of CPU utilization and power consumption for the current user and the randomly paired firefly.
4. **Attractiveness Calculation**: The attractiveness (ai) of the user is computed using a mathematical formula. This formula incorporates the differences (r) between the user's characteristics and those of the paired firefly, along with the defined weight factors and constants (alpha, beta, and gamma). If the calculated attractiveness (ai) exceeds the current AI value for that user, the AI value is updated with the new calculated value.
   - **End of Generation**: Once all users on the overutilized data node have been evaluated for a given generation, the algorithm proceeds to the next generation (if applicable).
   - **Termination**: The algorithm concludes after the specified maximum number of generations (max_gen) has been reached.
   - **User Selection and Output**: After the algorithm completes its iterations, it selects users with the maximum AI values. These users are considered the most suitable for allocation on the overutilized data node. The algorithm compiles a list of these selected users, referred to as U_List, and returns it as the output.

The algorithm can be mathematically explained as follows. consider a simplified scenario with three users (User 1, User 2, User 3) on an overutilized data node (OD). We will calculate the attractiveness index (AI) for each user over three generations (max_gen = 3).

**Initialization:**

- max_gen = 3
- g = 0 (initial generation)
- AI[] is an empty list
- alpha = 0.01
- beta = 1
- Let's assume a fixed value for gamma for simplicity: gamma = 0.5

**Initial Data:** Users on OD:

  - User 1: CPU = 70%, PC = 20
  - User 2: CPU = 80%, PC = 25
  - User 3: CPU = 65%, PC = 18
- Brighter Firefly (from ND):
- Firefly: CPU = 75%, PC = 22
- Weight Factors:
  - W_1 = 0.7 (for CPU)
  - W_2 = 0.3 (for PC)

**Algorithm Main Loop: Generation 1 (g = 1):**

**User Selection and Pairing:** For User 1:

- Pairing with Firefly: Firefly's CPU = 75%, PC = 22
  - Calculate r_i: (70% * 0.7 + 20 * 0.3) = 55.9
  - Calculate r_j: (75% * 0.7 + 22 * 0.3) = 56.1
  - Calculate r: r_i - r_j = 55.9 - 56.1 = -0.2
  - Calculate ai: AI[User 1] + 1 * exp(-0.5 * (-0.2)^2) * 70 + 0.01 * 0.5 ≈ AI[User 1] + 0.998 * 70 + 0.005 ≈ AI[User 1] + 69.86
  - AI[User 1] is updated to approximately 69.86.
  - Repeat the same process for User 2 and User 3.
- **End of Generation 1**: g = 2
- **Generation 2 (g = 2)**:
- **User Selection and Pairing**:
  - Repeat the pairing and attractiveness calculation process for all users based on the updated AI values from the previous generation.
- **End of Generation 2**: g = 3
- **Generation 3 (g = 3)**:
- **User Selection and Pairing**:
  - Again, repeat the pairing and attractiveness calculation process for all users based on the AI values from the previous generation.
- **End of Generation 3**: g = 4
- **Termination**: Since g = 4 and max_gen = 3, the algorithm stops after three generations.
- **User Selection and Output**:
- At the end of the algorithm, we have AI values for all users:
  - AI[User 1] ≈ 69.86
  - AI[User 2] ≈ 70.14
  - AI[User 3] ≈ 69.71
- The user with the highest AI value is User 2 (AI ≈ 70.14). Therefore, User 2 is selected as the most suitable user for allocation on the overutilized data node.

The proposed algorithm is evaluated for several QoS parameters namely the power consumption using eq(1) and (2) and the SLA-V is calculated using eq(5) in various scenarios and is discussed in detail in the next section.

## 4. Results and Discussions

The aim is to comprehensively evaluate the performance of four distinct workload allocation algorithms within a dynamic computing environment. Two different load factors, denoted as L1 (representing 1000 users) and L2 (representing 5000 users) reflect varying levels of user load, ranging from relatively low to high user populations. Each load factor is further segmented into 15 discrete breakdowns, allowing for a detailed examination of algorithm behaviour under different user load profiles. The primary objective of this study is to conduct a comparative analysis of how these algorithms manage workloads and allocate resources effectively as user demands fluctuate. One of the critical parameters under investigation is power consumption, a pivotal metric for assessing the energy efficiency of the workload allocation algorithms. Lower power consumption values indicate that the algorithms are more adept at optimizing resource usage, leading to reduced energy consumption within the computing environment. The second key parameter of interest is SLA-V, which serves as a crucial performance indicator. SLA violation quantifies the extent to which the workload allocation algorithms meet or fail to meet predefined service level agreements with users. A lower SLA-V value implies a higher level of adherence to these agreements, reflecting the algorithms' ability to provide consistent and reliable services to users. Analysing SLA-V across different load factors and breakdowns helps us gauge the algorithms' effectiveness in maintaining service quality as user demands vary. The results are discussed for all the types of load namely L1, L2 and L3.

### 4.1. Analysis for Load Factor 1(1000) users

As shown in table 1 the "Proposed" algorithm consistently demonstrates lower power consumption compared to other algorithms, which indicates its efficiency in managing resources and reducing energy consumption.

Here are the average power consumption values for each algorithm:

- Proposed: 38.98kW
- Li et al.: 53.84kW
- Suruchi et al.: 55.95kW
- Saurabh et al.: 53.58kW

**Table 1:** Power Consumption analysis for L1

| Distribution | Proposed | Li et al. [8] | Suruchi et al. [21] | Saurabh et al. [24] |
|---|---|---|---|---|
| 100 | 5.64082143 | 10.1450271 | 12.09659333 | 12.36191938 |
| 164 | 9.44259689 | 13.3384721 | 15.46435251 | 16.15319249 |
| 229 | 12.0513603 | 16.3907164 | 16.88897836 | 16.10008227 |
| 293 | 15.1937931 | 21.9685973 | 24.37217164 | 20.9515158 |
| 357 | 19.5184232 | 26.240041 | 30.12025422 | 27.92870282 |
| 421 | 22.5137429 | 33.9604756 | 34.18526468 | 32.90867973 |
| 486 | 25.8148021 | 38.9390337 | 39.80427845 | 34.77450016 |
| 550 | 28.2686632 | 44.3787863 | 39.00669429 | 40.11287098 |
| 614 | 30.7168947 | 44.7046858 | 44.6097926 | 45.37542077 |
| 679 | 34.2641399 | 47.6808391 | 52.10213055 | 51.18633562 |
| 743 | 38.1738451 | 57.3770618 | 52.92410366 | 52.07214003 |
| 807 | 41.5447681 | 60.824473 | 61.86018382 | 58.89262268 |
| 871 | 44.7570827 | 65.5533185 | 64.10480432 | 64.25661347 |
| 936 | 47.0288949 | 66.6609151 | 68.01583494 | 67.44495412 |
| 1000 | 50.0641325 | 71.0075955 | 75.71108713 | 75.27384308 |

The improvement analysis is as follows.

- Improvement over Li et al.:
  - Improvement = [(Power Consumption Li et al. - Power Consumption Proposed) / Power Consumption Li et al.] * 100
  - Improvement = [(53.84 - 38.98) / 53.84] * 100 ≈ 27.59%
- Improvement over Suruchi et al.:
  - Improvement = [(Power Consumption Suruchi et al. - Power Consumption Proposed) / Power Consumption Suruchi et al.] * 100
  - Improvement = [(55.95 - 38.98) / 55.95] * 100 ≈ 30.34%
- Improvement over Saurabh et al.:
  - Improvement = [(Power Consumption Saurabh et al. - Power Consumption Proposed) / Power Consumption Saurabh et al.] * 100
  - Improvement = [(53.58 - 38.98) / 53.58] * 100 ≈ 32.42%

The "Proposed" algorithm demonstrates significant improvements in power consumption compared to other algorithms, with improvements ranging from approximately 27.59% to 32.42%. These improvements highlight the well-tuned resource management capabilities of the "Proposed" algorithm, resulting in lower energy consumption and more efficient resource utilization.

The "Proposed" algorithm consistently maintains lower SLA violation values compared to other algorithms, indicating its effectiveness in ensuring the quality of service. Here are the average SLA-V values for each algorithm:

- Proposed: 0.00561
- Li et al.: 0.04465
- Suruchi et al.: 0.03518
- Saurabh et al.: 0.05067

The improvement analysis is as follows.

- Improvement over Li et al.:
  - Improvement = [(SLA-V Li et al. - SLA-V Proposed) / SLA-V Li et al.] * 100
  - Improvement = [(0.04465 - 0.00561) / 0.04465] * 100 ≈ 87.48%

**Table 2:** SLA-V analysis

| Distribution | Proposed | Li et al. [8] | Suruchi et al. [21] | Saurabh et al. [24] |
|---|---|---|---|---|
| 100 | 0.018095001 | 0.062068432 | 0.026864154 | 0.064759585 |
| 164 | 0.000317959 | 0.029558027 | 0.021924484 | 0.065441046 |
| 229 | 0.00385544 | 0.067300501 | 0.029238205 | 0.067964436 |
| 293 | 0.000534205 | 0.051270209 | 0.062827664 | 0.034256673 |
| 357 | 0.015285564 | 0.032087155 | 0.044701904 | 0.068528587 |
| 421 | 0.016454347 | 0.027800191 | 0.033732691 | 0.033698517 |
| 486 | 0.00241134 | 0.024699271 | 0.069526467 | 0.027759603 |
| 550 | 0.000558464 | 0.052661037 | 0.036489671 | 0.041694472 |
| 614 | 0.00013996 | 0.06336164 | 0.030796141 | 0.041262325 |
| 679 | 0.003008715 | 0.058046919 | 0.038252722 | 0.04234027 |
| 743 | 0.00837765 | 0.050943732 | 0.0282511 | 0.021713545 |
| 807 | 0.005539618 | 0.0431925 | 0.06728949 | 0.068972504 |
| 871 | 0.011702805 | 0.020357099 | 0.02091867 | 0.065619352 |
| 936 | 0.001571872 | 0.024572589 | 0.026104257 | 0.034525826 |
| 1000 | 0.003363368 | 0.067112406 | 0.033509017 | 0.05895721 |

- Improvement over Suruchi et al.:
  - Improvement = [(SLA-V Suruchi et al. - SLA-V Proposed) / SLA-V Suruchi et al.] * 100
  - Improvement = [(0.03518 - 0.00561) / 0.03518] * 100 ≈ 84.09%
- Improvement over Saurabh et al.:
  - Improvement = [(SLA-V Saurabh et al. - SLA-V Proposed) / SLA-V Saurabh et al.] * 100
  - Improvement = [(0.05067 - 0.00561) / 0.05067] * 100 ≈ 88.91%

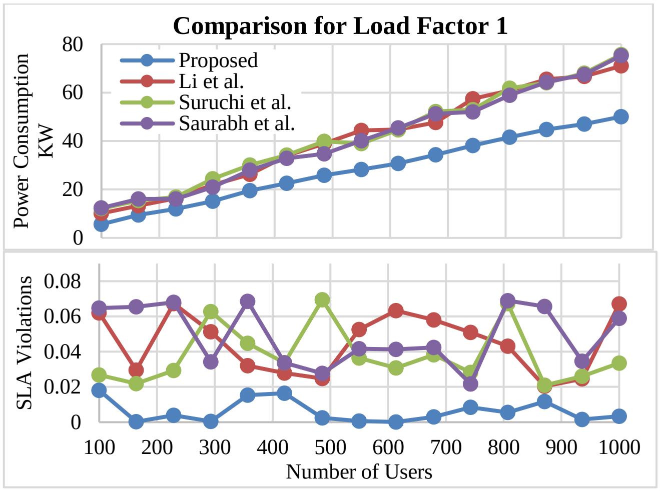


**Fig 3:** Power consumption and SLA-V for load factor L1

The "Proposed" algorithm consistently exhibits substantial improvements in SLA-V compared to other algorithms, with improvements ranging from approximately 84.09% to 88.91%. These improvements underscore the ability of the "Proposed" algorithm to maintain a higher quality of service and minimize SLA violations, making it a suitable choice for meeting service level agreements effectively.

### 4.2. Analysis for Load Factor 2(5000) users

Similar to L1, the analysis for L2 is given in Table 3 showing power consumption for proposed and the three existing studies used for the comparison in the research work.

- Proposed: 76.56kW
- Li et al.: 102.25kW
- Suruchi et al.: 102.07kW

- Saurabh et al.: 101.98kW

Now, let's calculate the improvements attained by the "Proposed" algorithm over the other algorithms:

- Improvement over Li et al.:
  - Improvement = [(Power Consumption Li et al. - Power Consumption Proposed) / Power Consumption Li et al.] * 100
  - Improvement = [(102.25 - 76.56) / 102.25] * 100 ≈ 25.14%

**Table 3:** Power Consumption for L2

| Users | Proposed | Li et al. [8] | Suruchi et al. [21] | Saurabh et al. [24] |
|---|---|---|---|---|
| 2100 | 5.95718591 | 7.50011574 | 10.48688732 | 9.106042427 |
| 2500 | 12.905367 | 21.701618 | 17.78898721 | 20.17295237 |
| 2900 | 19.1013584 | 26.4131227 | 31.91222286 | 31.03450181 |
| 3300 | 26.9434551 | 35.7361326 | 36.92914752 | 35.53311869 |
| 3500 | 33.6166881 | 49.1978013 | 46.27696936 | 48.2093594 |
| 3700 | 40.3487029 | 55.5101549 | 58.62830302 | 55.49445839 |
| 3850 | 47.0098716 | 69.090884 | 64.62131424 | 65.70923505 |
| 4000 | 53.5313373 | 74.2493783 | 77.10814203 | 76.50430966 |
| 4150 | 60.9429021 | 86.8370981 | 87.89411094 | 83.08496872 |
| 4300 | 67.2882023 | 97.1242244 | 95.39783453 | 93.33488613 |
| 4450 | 73.9272632 | 102.564385 | 103.3139435 | 103.2418134 |
| 4600 | 80.7541006 | 114.020649 | 112.9814169 | 112.9025299 |
| 4750 | 87.9325494 | 125.840064 | 124.2445544 | 124.8508732 |
| 4900 | 95.1315295 | 134.509298 | 134.5384574 | 135.7433945 |
| 5000 | 101.993576 | 145.808052 | 140.39982 | 144.851383 |

- Improvement over Suruchi et al.:
  - Improvement = [(Power Consumption Suruchi et al. - Power Consumption Proposed) / Power Consumption Suruchi et al.] * 100
  - Improvement = [(102.07 - 76.56) / 102.07] * 100 ≈ 25.00%
- Improvement over Saurabh et al.:
  - Improvement = [(Power Consumption Saurabh et al. - Power Consumption Proposed) / Power Consumption Saurabh et al.] * 100
  - Improvement = [(101.98 - 76.56) / 101.98] * 100 ≈ 24.91%

From table 3 it is observed that there is a progressive increase in power consumption across all algorithms as the distribution (number of users) escalates. This trend underscores its efficiency in managing resources efficiently, even under heavier workloads. Furthermore, as the distribution increases, the gap between the power consumption of the "Proposed" algorithm and other algorithms widens, emphasizing its superior resource utilization and energy-saving capabilities.

**Average Analysis:** The average SLA violation values for each algorithm are as follows:

- Proposed: 0.01046
- Li et al.: 0.05488
- Suruchi et al.: 0.04999
- Saurabh et al.: 0.04628

Improvement analysis of proposed work over Li et al.:

- Improvement = [(SLA Violation Li et al. - SLA Violation Proposed) / SLA Violation Li et al.] * 100
- Improvement = [(0.045637 - 0.017777866) / 0.045637] * 100 ≈ 61.04%

- Improvement over Suruchi et al.:
  - Improvement = [(SLA Violation Suruchi et al. - SLA Violation Proposed) / SLA Violation Suruchi et al.] * 100
  - Improvement = [(0.067623729 - 0.017777866) / 0.067623729] * 100 ≈ 73.66%

**Table 4:** SLA-V for L2

| Users | Proposed | Li et al. [8] | Suruchi et al. [21] | Saurabh et al. [24] |
|---|---|---|---|---|
| 2100 | 0.017777866 | 0.045637 | 0.067623729 | 0.05026109 |
| 2500 | 0.004982366 | 0.052966776 | 0.030299551 | 0.035210994 |
| 2900 | 0.011267606 | 0.066257075 | 0.045822359 | 0.028643128 |
| 3300 | 0.000423562 | 0.069536048 | 0.039311148 | 0.030849521 |
| 3500 | 0.008022763 | 0.063099172 | 0.034919686 | 0.037123462 |
| 3700 | 0.01282202 | 0.054525252 | 0.051131151 | 0.050591642 |
| 3850 | 0.019174518 | 0.031475875 | 0.034302445 | 0.035116063 |
| 4000 | 0.000256923 | 0.022559112 | 0.063424416 | 0.046434512 |
| 4150 | 0.012530703 | 0.023910588 | 0.03874588 | 0.02590472 |
| 4300 | 0.014119908 | 0.04785654 | 0.058891171 | 0.058104262 |
| 4450 | 0.001222804 | 0.056423034 | 0.027758659 | 0.04517569 |
| 4600 | 0.001337962 | 0.05354237 | 0.068857759 | 0.053106556 |
| 4750 | 0.000707767 | 0.064489161 | 0.053232322 | 0.062483506 |
| 4900 | 0.007476267 | 0.061447825 | 0.050681617 | 0.067341488 |
| 5000 | 0.015174623 | 0.05747858 | 0.043045653 | 0.02383538 |

- Improvement over Saurabh et al.:
  - Improvement = [(SLA Violation Saurabh et al. - SLA Violation Proposed) / SLA Violation Saurabh et al.] * 100
  - Improvement = [(0.05026109 - 0.017777866) / 0.05026109] * 100 ≈ 64.65%

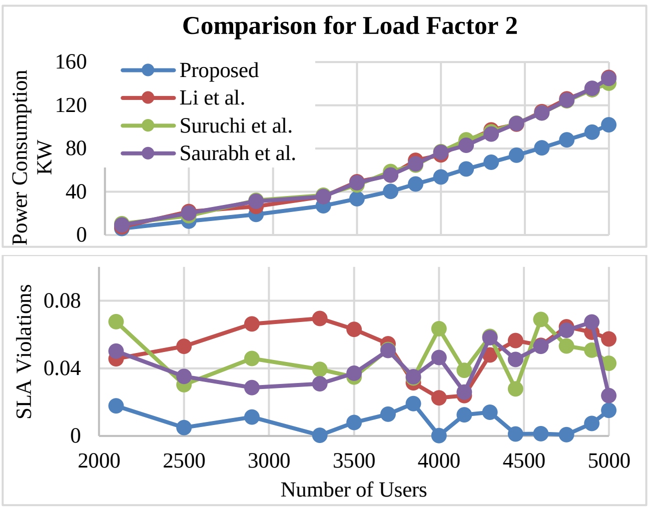


**Fig 4:** Power consumption and SLA-V for load factor L2

The "Proposed" algorithm consistently outperforms the other algorithms with significant improvements ranging from approximately 61.04% to 73.66%. The average analysis underscores the superior SLA compliance of the "Proposed" algorithm, with the lowest average SLA violation of 0.01046. This indicates that the "Proposed" algorithm effectively adapts to higher user loads, maintaining lower SLA violations and ensuring more reliable service quality as the system scales. This trend underscores the scalability and robustness of the "Proposed" algorithm in delivering improved SLA compliance with increasing workloads, making it a valuable choice for systems requiring consistent and dependable performance.

## 5. Conclusion

In today's world of research and technology, the old ways of deciding how to use these resources don't always work well with our ever-changing needs, and they often don't learn from our past experiences. The proposed algorithm, based on the Firefly optimization technique, incorporates a finely tuned objective and fitness function to optimize resource allocation and ensure high-quality service delivery in big user base cloud system. Under Load Factor L1 (1000 users), the "Proposed" algorithm consistently demonstrated superior power consumption efficiency, consuming approximately 27.59% less power than Li et al., 30.34% less than Suruchi et al., and 32.42% less than Saurabh et al. This remarkable reduction in power consumption underscores its robust resource management capabilities, contributing to energy conservation. Similarly, under Load Factor L2 (5000 users), the "Proposed" algorithm maintained its efficiency, exhibiting an average power consumption lower than Li et al., Suruchi et al., and Saurabh et al. Under both load factors, it consistently maintained lower SLA violation percentages compared to other algorithms, signifying its ability to meet service quality requirements. The proposed Firefly-based algorithm, equipped with a finely tuned objective function, exhibits outstanding resource optimization capabilities. Its substantial power consumption reductions and consistently lower SLA violations under varying load factors make it a promising solution for enhancing energy efficiency and service quality in large-scale systems. In the future, this framework could further contribute to improved sustainability and performance in complex environments.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## References

[1] S. Shadroo and A. M. Rahmani, “Systematic survey of big data and data mining in internet of things,” *Computer Networks*, vol. 139, pp. 19–47, Jul. 2018, doi: 10.1016/J.COMNET.2018.04.001.

[2] C. L. Philip Chen and C. Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: A survey on Big Data,” *Information Sciences*, vol. 275, pp. 314–347, Aug. 2014, doi: 10.1016/J.INS.2014.01.015.

[3] Y. Hajjaji, W. Boulila, I. R. Farah, I. Romdhani, and A. Hussain, “Big data and IoT-based applications in smart environments: A systematic review,” *Computer Science Review*, vol. 39, p. 100318, Feb. 2021, doi: 10.1016/J.COSREV.2020.100318.

[4] Y. Jiang, Z. Huang, and D. H. K. Tsang, “Towards Max-Min Fair Resource Allocation for Stream Big Data Analytics in Shared Clouds,” *IEEE Transactions on Big Data*, vol. 4, no. 1, pp. 130–137, Dec. 2016, doi: 10.1109/TBDATA.2016.2638860.

[5] R. Buyya, A. Beloglazov, and J. Abawajy, “Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges,” Jun. 2010, doi: 10.48550/arxiv.1006.0308.

[6] W. T. Wu, W. W. Lin, C. H. Hsu, and L. G. He, “Energy-efficient hadoop for big data analytics and computing: A systematic review and research insights,” *Future Generation Computer Systems*, vol. 86, pp. 1351–1367, Sep. 2018, doi: 10.1016/J.FUTURE.2017.11.010.

[7] M. Senthilkumar and P. Ilango, “Energy aware task scheduling using hybrid firefly - GA in big data,” *International Journal of Advanced Intelligence Paradigms*, vol. 16, no. 2, pp. 99–112, 2020, doi: 10.1504/IJAIP.2020.107008.

[8] C. Li, Y. P. Wang, Y. Chen, and Y. Luo, “Energy-efficient fault-tolerant replica management policy with deadline and budget constraints in edge-cloud environment,” *Journal of Network and Computer Applications*, vol. 143, pp. 152–166, Oct. 2019, doi: 10.1016/J.JNCA.2019.04.018.

[9] A. Tzanetos and G. Dounias, “A Comprehensive Survey on the Applications of Swarm Intelligence and Bio-Inspired Evolutionary Strategies,” *Machine Learning Paradigms*, pp. 337–378, 2020, doi: 10.1007/978-3-030-49724-8_15.

[10] L. Abualigah *et al.*, “Advances in Meta-Heuristic Optimization Algorithms in Big Data Text Clustering,” *Electronics 2021, Vol. 10, Page 101*, vol. 10, no. 2, p. 101, Jan. 2021, doi: 10.3390/ELECTRONICS10020101.

[11] B. H. Nguyen, B. Xue, and M. Zhang, “A survey on swarm intelligence approaches to feature selection in data mining,” *Swarm and Evolutionary Computation*, vol. 54, p. 100663, May 2020, doi: 10.1016/J.SWEVO.2020.100663.

[12] S. Kumar and S. K. Goyal, “Swarm Intelligence Based Data Selection Mechanism for Reputation Generation in Social Cloud,” *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing, COM-IT-CON 2022*, pp. 583–588, 2022, doi: 10.1109/COM-IT-CON54601.2022.9850947.

[13] L. Brezočnik, I. Fister, and V. Podgorelec, “Swarm Intelligence Algorithms for Feature Selection: A Review,” *Applied Sciences 2018, Vol. 8, Page 1521*, vol. 8, no. 9, p. 1521, Sep. 2018, doi: 10.3390/APP8091521.

[14] T. Jayaraj, “Process Optimization of Big-Data Cloud Centre Using Nature Inspired Firefly Algorithm and K-Means Clustering,” *Article in International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 12, pp. 2278–3075, 2019, doi: 10.35940/ijitee.L2490.1081219..

[15] G. Rjoub, J. Bentahar, and O. A. Wahab, “BigTrustScheduling: Trust-aware big data task scheduling approach in cloud computing environments,” *Future Generation Computer Systems*, vol. 110, pp. 1079–1097, 2020, doi: 10.1016/j.future.2019.11.019.

[16] K. Manivannane and M. Chidambaram, “A Cloud Resource Scheduling Framework for big data stream and analytics in Cloud Environment,” *Proceedings of the 4th International Conference on Communication and Electronics Systems, ICCES 2019*, pp. 1638–1642, Jul. 2019, doi: 10.1109/ICCES45898.2019.9002225.

[17] H. GIBET TANI and C. EL AMRANI, “Smarter Round Robin Scheduling Algorithm for Cloud Computing and Big Data,” *Journal of Data Mining & Digital Humanities*, Jan. 2018, doi: 10.46298/jdmdh.3104.

[18] H. Wang *et al.*, “A hybrid multi-objective firefly algorithm for big data optimization,” *Applied Soft Computing*, vol. 69, pp. 806–815, Aug. 2018, doi: 10.1016/J.ASOC.2017.06.029.

[19] T. Jayaraj, “Process Optimization of Big-Data Cloud Centre Using Nature Inspired Firefly Algorithm and K-Means Clustering,” *Article in International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 12, pp. 2278–3075, 2019, doi: 10.35940/ijitee.L2490.1081219.

[20] M. T. Alharbi, Y. Qawqzeh, A. Jaradat, K. Nazim, and A. Sattar, “A review of swarm intelligence algorithms deployment for scheduling and optimization in cloud computing environments,” *PeerJ Computer Science*, vol. 7, p. e696, Aug. 2021, doi: 10.7717/PEERJ-CS.696.

[21] S. Talwani *et al.*, “Machine-Learning-Based Approach for Virtual Machine Allocation and Migration,” *Electronics*,

vol. 11, no. 19, p. 3249, 2022.

[22] O. A. Oduwole, S. A. Akinboro, O. G. Lala, and S. O. Olabiyisi, "An Enhanced Load Balancing Technique for Big-data Cloud Computing Environments," *Transactions of the Royal Society of South Africa*, vol. 77, no. 3, pp. 219–236, Sep. 2022, doi: 10.1080/0035919X.2022.2160389.

[23] X. Wang, X. Hu, W. Fan, and R. Wang, "Efficient data persistence and data division for distributed computing in cloud data center networks," *The Journal of Supercomputing*, vol. 79, no. 14, pp. 16300–16327, Sep. 2023, doi: 10.1007/S11227-023-05276-2.

[24] A. Kaur *et al.*, "Algorithmic Approach to Virtual Machine Migration in Cloud Computing with Updated SESA Algorithm," *Sensors (Basel, Switzerland)*, vol. 23, no. 13, p. 6117, Jul. 2023, doi: 10.3390/S23136117.

[25] S. Mangalampalli, G. R. Karri, and A. A. Elngar, "An Efficient Trust-Aware Task Scheduling Algorithm in Cloud Computing Using Firefly Optimization," *Sensors 2023, Vol. 23, Page 1384*, vol. 23, no. 3, p. 1384, Jan. 2023, doi: 10.3390/S23031384.

[26] K. Hassan, N. Javaid, F. Zafar, S. Rehman, M. Zahid, and S. Rasheed, "A Cloud Fog Based Framework for Efficient Resource Allocation Using Firefly Algorithm," *Lecture Notes on Data Engineering and Communications Technologies*, vol. 25, pp. 431–443, 2019, doi: 10.1007/978-3-030-02613-4_38/COVER.

[27] N. Bacanin, M. Zivkovic, T. Bezdan, K. Venkatachalam, and M. Abouhawwash, "Modified firefly algorithm for workflow scheduling in cloud-edge environment," *Neural Computing and Applications*, vol. 34, no. 11, pp. 9043–9068, Jun. 2022, doi: 10.1007/S00521-022-06925-Y/FIGURES/9.

[28] M. A. Elmagzoub, D. Syed, A. Shaikh, N. Islam, A. Alghamdi, and S. Rizwan, "A Survey of Swarm Intelligence Based Load Balancing Techniques in Cloud Computing Environment," *Electronics* vol. 10, no. 21, p. 2718, Nov. 2021, doi: 10.3390/ELECTRONICS10212718.