

# A Novel Methodology for Container Scheduling and Load Balancing in Distributed Environments

Saravanan.M.S<sup>1</sup> Neelima Gogineni<sup>2</sup>

<sup>1</sup>Professor, Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Chennai – 602105 Tamilnadu. India.

<sup>2</sup>Research Scholar, Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences Chennai – 602105, Tamilnadu. India.

---

## Article Info

### Article history:

Received Oct 4, 2024

Revised Dec 19, 2024

Accepted Jan 9, 2025

---

### Keyword:

Container Scheduling,  
Load Balancing,  
Distributed Computing,  
Cloud Computing

---

## ABSTRACT

Deployment of applications in distributed environments via containers has gained huge popularity lately, specifically with cloud-based ecosystems. Inspired by the quick growth of container usage and deployment in distributed environments, efficient scheduling techniques are of prior significance embedded with load balancing in it for cloud computing tasks. Most of the scheduling strategies adopt conventional methods and fail to execute efficiently in the dynamic cloud or distributed environments where applications around the world depend on them for scalability, efficiency, and availability. Existing applications focus more on performance metrics instead of scheduling efficiency, so often they offer performance that can come at the expense of scheduling. This paper proposes a new algorithm that includes consideration of contention over the network, along with efficient canister planning and load distribution. The algorithm we have designed to achieve the proposed scheduling and load balancing is Contention-aware Greedy Heuristic Scheduling and Load Balancing for Containers (CGHSLBC), which has been extensively evaluated under continuous workload and has outperformed current state-of-the-art algorithms by 20% in load balancing efficiency and 25% in network contention reduction, demonstrating its promise for container scheduling in dynamic distributed environments.

*Copyright © 2025 Institute of Advanced Engineering and Science.  
All rights reserved.*

---

## Corresponding Author:

Saravanan.M.S

Professor, Department of Computer Science and Engineering,

Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences,  
Chennai – 602105 Tamilnadu. India.

Email: [saravananms@saveetha.com](mailto:saravananms@saveetha.com)

---

## 1. INTRODUCTION

The services and infrastructure provided by cloud computing can be leveraged for new cases as applications can be deployed and scaled in remote locations and be accessible to users at any time or place. However, many applications are developed and deployed in cloud computing environments over time. Cloud computing has become cheaper because of Virtual machines and virtualization technology. The cloud ecosystem caters to microservices and containerized app-based architectures. Container-based applications integrated with cloud services are packaged as unit workloads with facets that can be deployed into distributed environments, and differentiate the location of the application from the user, providing services closer to the end user. Yet, the cloud is dynamic and resource-intensive, and remedying these challenges such as scheduling and load balancing is nontrivial in the cloud ecosystem. Due to these inefficiencies, for example, low resource utilization, low scalability, and high computational overhead, traditional scheduling algorithms usually perform badly in cloud computing environments. For instance, Patra et al. Randomized load-balancing approaches [6], although they are among the best known since they operate without any prior knowledge of the workloads at each resource, they fail to dynamically adapt to the workload variation and allocate the resources in a

suboptimal manner. Similarly, Ma et al. Edge computing has numerous advantages but has been hampered by inefficient container migration mechanisms that incur high latency and fail to scale for dynamic and continuous workload changes [11].

Additionally, although useful in special situations, most heuristic-based approaches do not avoid the NP-hardness of scheduling problems or are computationally expensive, as pointed out by Zhou et al. [15]. Although reinforcement learning-based methods are promising [4], they struggle to scale effectively and must rely on analytical joins in the modeling process, which makes them more difficult to use in heterogeneous environments. These shortcomings require novel approaches to optimize load balancing and contend for network bandwidth as efficiently as possible. Beyond this trend, now container-based approaches like docker are witnessing great achievement even in software-defined computing environments, for their sustainability, scalability, and availability of applications. Still, network contention and I/O congestion are two factors that current approaches do not sufficiently tackle. Singh et al. They find that while the placement of containers in large-scale systems is occasionally restricted by scalability limitations, the loss of performance is often prompted by the lack of effective resource allocations. [14]

Thus, the lack of a suitable scheduling algorithm to lessen these drawbacks and additionally balance the loads and adapt both exquisitely to the network contention necessitates such a new scheduling algorithm. We propose a novel algorithm, Contention-aware Greedy Heuristic Scheduling and Load Balancing for Containers (CGHSLBC) that takes into account both network contention and I/O dynamics and achieves total resource utilization while significantly improving the performance of the application at run-time. We demonstrate the superiority of our method over state-of-the-art algorithms through empirical evaluations with workloads of continuous workloads. While there have been many promising developments in container scheduling and load balancing, the existing techniques are still far from solving these crucial problems in distributed environments. Heuristic-based approaches, which are computationally efficient but appropriately scale poorly and cannot benefit from dynamic workloads, often lead to inefficient resource usage. Reinforcement learning-based frameworks hold promising solutions to many of these challenges but require significant computational power and fail when environments are only partially homogeneous. Many existing algorithms also ignore this trade-off of balancing load and network contention which leads to high latency and degradation in performance as we go to large-scale deployments. This motivates the underlying method of the developed CGHSLBC algorithm that incorporates real-time adjustability, awareness of I/O contention, and dependency handling. The rest of this paper is organized as follows: Section 2 provides a literature review of existing techniques for the problems of load balancing and container scheduling. In Section 3, we describe our contention-aware and efficient scheduling and load-balancing approach for distributed systems. Our empirical investigation and the performance of the proposed method are provided in section 4. Section 5 wraps up our results and offers directions for future work.

## 2. RELATED WORK

Alqahtani et al. [1] created the Load Balanced Service Scheduling Approach (LBSSA) to handle load balancing and dependability in fog computing, with plans for further simulation evaluations. Xie and Govardhan [2] created a scalable architecture emphasizing load balancing and resource assessment for implementing DL applications utilizing technology for service meshes and containers. Ahmad et al. [3] carried out an extensive assessment on container scheduling strategies, classifying, and highlighting advantages, drawbacks, and potential areas for further cloud computing research. Mattia and Beraldi [4] created the P2PFaaS framework, which uses Docker containers and supports many algorithms, including Reinforcement Learning, for decentralized scheduling in Edge and Fog Computing. Potential areas for improvement in the future may be scalability and new features. Singh et al. [5] provided a low-weight Container-as-a-Service (CaaS) approach that addresses energy concerns and edge computing compatibility difficulties while handling latency-sensitive IoT applications.

Patra et al. [6] provided a randomized load-balancing approach that addresses both scalability and dynamic workload adaption when allocating jobs to servers in containerized clouds. Muniswamy and Vignesh [7] improved resource allocation by developing a hybrid deep-learning technique for dynamic job scheduling in containerized clouds. Subsequent endeavors may enhance optimizations. Dhabbi et al. [8] created novel approaches to cloud computing load balancing that improve resource efficiency and minimize makespan. The complexity of the algorithm is one of the limitations. Additional improvements and scalability may be the focus of the next studies. Saif et al. [9] created an autonomous CSO-ILB load balancer to improve performance and handle workloads effectively in multi-cloud settings. Communication overheads may be the subject of future research. Oleghe [10] discussed edge computing container deployment and migration, with a focus on scheduling models. Inadequate decentralized systems are one of the limitations; mobile edge scheduling should be the main focus of future research.

Ma et al. [11] created a decision-making method for container migration in edge computing to support power IoT load balancing. Subsequent investigations might enhance tactics and implementations. Rajasekar and palanichamy [12] improved resource consumption, Workflow as a Service (WaaS) required the development of a dynamic scheduling and resource provisioning system. Potential issues with scalability are among the limitations. Subsequent research endeavours might concentrate on enhancing algorithms to achieve more efficiency and economy. Menouer [13] constructed a multi-criteria KCSS, or Kubernetes Container Scheduling Strategy to increase the effectiveness of container placement. Difficulties stem from implementation complexity; future research might improve integration and flexibility. Singh et al. [14] examined the use of containers in large data analysis and presented the Docker Swarm scheduling method for load balancing. Implementation scope is limited; scalability and performance metrics might be improved by further study. Zhou et al. [15] examined many metaheuristic load-balancing methods for cloud computing, emphasizing the higher performance of particle swarm optimization. Among the limitations is the problem's NP-hardness. Further research might concentrate on improving the efficiency of these algorithms.

Tychalas and Karatza [16] demonstrated a fog computing system that uses a variety of resources to keep response times consistent while cutting expenses. Reliance on particular workload models is one of the limitations. More applications and optimization methods may be investigated in future research. Shekhar and Sharvani [17] provided a multi-tenant load balancing technique (MTLBP) with scaling problems to improve work efficiency and cloud resource management. Subsequent research endeavors may augment adaptability throughout diverse application sectors. Ranjan et al. [18] created a containerized process scheduling approach that saves energy in software-defined data centers while improving resource performance and access. Complexity is one of the limitations. Enhancing scalability and flexibility across a range of applications may be the main emphasis of future study. Cai et al. [19] created the HDCBS scheduling model to enhance load balancing and system performance for affordable bioinformatics processes in diverse cloud settings. Reliance on queuing theory is one of its limitations. Upcoming research might improve flexibility and scalability in a variety of applications. Srirama et al. [20] created a scheduling method for microservices that is container-aware and has auto-scaling, improving deployment speed and resource efficiency. One of the limitations is the complexity of the implementation; performance and scalability may be improved in future work.

Table 1. Summary of related works, their contributions, and identified deficiencies

Reference	Contribution	Deficiencies
Alqahtani et al. [1]	Load Balanced Service Scheduling Approach (LBSSA) for fog computing.	Limited scalability and lack of empirical validation in diverse workload scenarios.
Xie and Govardhan [2]	Scalable architecture for DL applications using service meshes and containers.	High complexity and resource overhead during service mesh deployment.
Mattia and Beraldi [4]	P2PFaaS framework for decentralized scheduling in edge computing.	Scalability and lack of support for dynamic workload adaptation.
Patra et al. [6]	Randomized load balancing algorithm for containerized cloud.	Inefficient in dynamically changing workloads, leading to poor resource utilization.
Ma et al. [11]	Decision-making method for container migration in edge networks.	High latency and limited ability to handle real-time dynamic workloads.
Zhou et al. [15]	Comparative analysis of metaheuristic algorithms for load balancing.	NP-hardness of algorithms and high computational complexity.
Singh et al. [14]	Docker Swarm scheduling for big data applications.	Limited scalability and lack of optimal resource allocation.
Ranjan et al. [18]	Energy-efficient workflow scheduling in software-defined data centers.	Complexity in implementation and limited adaptability to varied workloads.
Cai et al. [19]	HDCBS model for cost-effective bioinformatics task scheduling.	Reliance on queuing theory and lack of flexibility in heterogeneous environments.

Nadim and Kamal [21] suggested cloud computing using a hybrid load-balancing method that outperforms current techniques and increases efficiency and scalability. Future research and limitations should look into more optimization strategies and practical implementation issues. Devi et al. [22] reviewed the literature in a methodical manner with a focus on the gaps in technology and upcoming prospects in the areas of Job scheduling and load distribution in cloud computing. Limitations include the need for more extensive investigation and the possibility of bias in some of the papers. Menaka et al. [23] reduced makespan and energy consumption by developing a mixed load balancing technique for virtual machine job scheduling. Upcoming projects will investigate more comprehensive optimization methods and fill up known research gaps. Rausch et al. [24] created a scheduling system for containers for edge computing that maximizes resource use and job completion. Scalability issues are among the limitations, and more work has to be done on expanding the use cases and fine-tuning the scheduling settings. Zhu et al. [25] enhanced cloud resource management, ADATSA—a learning automata-based self-adapting task scheduling technique—was presented. Potential implementation complexity is one of the limitations; future studies will concentrate on better-improving

performance metrics and flexibility. Table 1 provides a concise summary of significant related works in container scheduling and load balancing, highlighting their primary contributions and deficiencies. The table underscores the challenges in scalability, adaptability, and resource optimization, driving the need for a novel strategy to address network contention, dynamic workloads, and efficient load balancing in distributed environments.

### 3. PROPOSED METHODOLOGY

There are several zones in the system model, with a data center serving as the consideration for each zone. There may be a number of nodes in a zone where containers execute programs. A single cell is regarded as a Docker instance. The letter C stands for the number of accessible cells. Apps are the active programs within a certain cell. The terms CPU, RAM, and disk are used to identify the resources in each cell. The symbol "io" stands for the I/O bandwidth. Stands for applications list. Consideration is given to a dependency matrix D that illustrates relationships between applications. The system model uses zone, node, and cell as its allocation units. T stands for a matrix holding traffic data. Interactions between two nodes within the same zone incur less cost than those between two zones.

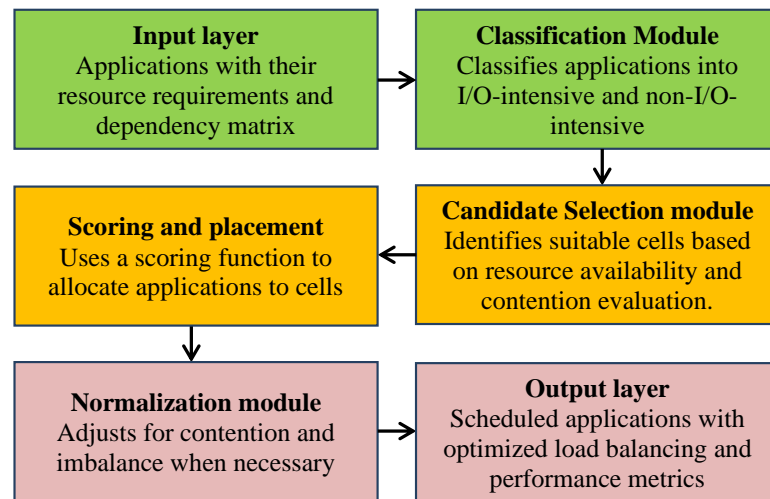


Figure 1. Conceptual abstract diagram of the proposed methodology

**Figure 1** illustrates the proposed methodology, Contention-aware Greedy Heuristic Scheduling and Load

Balancing for Containers (CGHSLBC), in a stepwise flow. It begins with the input layer, where applications with their resource requirements and dependencies are classified into I/O-intensive and non-I/O-intensive types. The candidate selection module identifies suitable cells for each application based on available resources and contention evaluation. The scoring and placement module optimally allocates applications to cells using a scoring function that balances load and minimizes network contention. The normalization module adjusts traffic and load balancing when no suitable cells are found. The output layer represents scheduled applications with improved load balancing and application performance metrics. This streamlined approach ensures efficiency and scalability in container scheduling.

The fact that running applications are moved to various places in an effort to optimize and balance load is a crucial factor. Two types of contentions are taken into consideration in order to balance the load and improve application performance. We refer to them as contentions on the local and network levels. Reducing the amount of nodes experiencing I/O congestion is the method's aim. The strategy outlined in Eq. 1 accomplishes this aim.

$$\arg_M \min \sum_{j=1}^{|C|} \left( (\sum \{A_k.io | cell(A_k) == C_j\}) \geq C_j.io \right) \quad (1)$$

Three requirements must be met for this statement to be true. The first criterion,  $C_j.cpu > A_i.cpu$  indicates that the CPU power of a particular cell must be greater than the CPU power of a certain application. The second criterion is that the RAM of a particular cell must be more than the RAM of a specific application  $C_j.ram > A_i.ram$ . The third requirement,  $C_j.disk > A_i.disk$  indicates that the DISK space of a particular cell must be more than the DISK space of a specific application. The aforementioned proposal aims to decrease

the quantity of cells that result in I/O contention. I/O-intensive apps operating in a cell are a major factor in I/O contention in that cell. Applications can fit inside any cell if they don't require a lot of input or output. When several I/O-intensive applications need to be deployed over a small number of nodes, job placement becomes less straightforward. This type of problem is not amenable to exponential solutions. To handle such a problem, an effective algorithm is required. In addition to slowing down the execution of other deployed apps, service providers occasionally may choose not to execute I/O-intensive applications due to the potential for SLA violations. Delaying such an application until the current one is finished or extra resources become available is one way to handle this issue. "How a server can schedule many such applications without exhausting a cell's I/O bandwidth?" is the simplest way to put this challenge now. The response is provided by Equation 2.

$$\arg_M \max \sum_{k=1}^i \left( (\sum \{A_n.io | cell(A_n) == C_j \ \&\& \ cell(A_k) == C_j \}) \leq C_j.io \right) \quad (2)$$

If cell  $C_j$  is already managing an I/O-intensive application, then  $A_h$  indicates such. If this is the case, the goal is to assign the next application,  $A_k$ , correctly without using up all of the available I/O bandwidth in the cell. The three requirements that were stated in the instance of Equation 1 must once more be met for this proposal to stand. While Eq. 1 aims to increase the number of programs that may operate in a cell without using up all of its I/O bandwidth, Eq. 2 attempts to reduce that number. A further factor taken into consideration is network contention. The three aforementioned allocation units are involved in this type of conflict. Applications and services are deemed interchangeable to streamline the analysis. Because both are viewed as planned jobs from the perspective of the proposed system, this is justified. Table 2's Dependency Matrix D is employed in this situation.

Table 2. Shows application dependencies

	A0	A1	A2	A3	A4	A5	A6	A7
A0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0
A2	0	1	0	0	0	0	0	0
A3	1	0	0	0	0	0	0	0
A4	0	0	0	1	0	0	0	0
A5	0	0	0	0	1	0	0	0
A6	0	0	0	0	0	1	0	0
A7	0	0	0	0	0	0	1	0

Applications and services may have dependencies, as shown in Table 2, and it is important to take these into account to maintain application performance while distributing load. The goal is to make scheduling decisions with the least amount of network conflict possible. Equation 3 expresses the minimum amount of network congestion when two applications, such as  $A_i$  and  $A_k$ , are dependent on each other.

$$\arg_j \min \sum_k D_{i,k} \times T_{j,zone(A_k)} \quad (3)$$

Under three restrictions, this statement is true. The first requirement,  $Z_j.cpu > A_i.cpu$ , indicates that the CPU power of a certain zone must be greater than the CPU power of a specific application. The second requirement,  $Z_j.ram > A_i.ram$ , indicates that the RAM of a particular zone must be more than the RAM of a specific application.  $Z_j.disk > A_i.disk$ , the third criterion, indicates that the DISK space of a zone should be more than the DISK space of an application. It is clear from Eq. 3 that it may be able to optimize a new application's network contention. Iterating over all zones and all applications, however, can result in a polynomial answer. Thus far, load balancing has not been the topic of debate; rather, it has been network contention. This is the plan to think about both. The approach is determined by taking into account a user-defined coefficient of variance. Formally, it is stated in Eq. 4.

$$\arg_j \min \sum_k D_{i,k} \times T_{j,zone(A_k)} \quad (4)$$

Four requirements must be met for this statement to be true. The initial trio of prerequisites is identical to that linked with Equation 3. The coefficient of variance must be less than or equal to the specified user-defined threshold, according to the fourth criterion,  $CV(Z) \leq th$ . With minimal additional overhead, Equation 4 accomplishes the desired result. It is insufficient, though, if there isn't a qualifying zone with the appropriate CV. To tackle this problem, a normalization process is taken into consideration, which involves adding a delay factor to both load balancing and network contention. The percentage of application traffic divided by the total aggregated bandwidth is known as network contention normalization. Equation 5 expresses the normalizing procedure.

$$tr(M_{i,j}) = \frac{\sum_{k=1}^{i-1} D_{i,k} \times T_{j,zone(A_k)}}{\sum_{m=1}^{|Z|} \sum_{n=1}^{|Z|} T_{m,n}} \quad (5)$$

According to this argument, the existence of a relation like  $tr(M_{i,j}) \in [0, \dots, 1], \forall i, j$ , is evident. Put differently, the total bandwidth need to consistently surpass the throughput of every application. In reference to the load balancing normalization that is attained by modifying the CV, as stated in Eq. 6.

$$cv(M_{i,j}) = \frac{\sqrt{(1+|Z_j \cdot apps| - \bar{Z})^2 + \sum_{k \neq j} (|Z_k \cdot apps| - \bar{Z})^2}}{\sqrt{|Z|} \cdot \bar{Z}} \quad (6)$$

The  $\bar{Z}$  Involved in the Eq. 6 is computed as in Eq. 7.

$$\bar{Z} = \frac{1 + \sum_{j=1}^{|Z|} |Z_j \cdot apps|}{|Z|} \quad (7)$$

The task assigned to a certain zone  $Z$  is taken into account. It's time to develop an objective function that takes into account both load balancing and network congestion now that you are aware of the two normalizations that address these issues. Weights are introduced for tr and CV normalization. Formally,  $\alpha$  and  $\beta$  stand for them, respectively. The objective is to identify an appropriate zone for an application with the necessary CV, as stated in Equation 8.

$$arg_M min F(M) \quad (8)$$

Where  $F(M)$  is known as a scoring function which is defined as in Eq. 9.

$$F(M) = \frac{\alpha \cdot tr(M) + \beta \cdot cv(M)}{\alpha + \beta} \quad (9)$$

If the first three requirements of Equation 4 are met, then this statement is accurate. A weighted mean is employed in Eq. 9 to create the goal function. It has been discovered to strike a balance between sensitivity and intricacy. For the goal function to eliminate overhead and missing ideal parameters, medium sensitivity is crucial. Both application performance and load balancing benefit equally from the goal function. Furthermore, the values of  $\alpha$  and  $\beta$  can be tuned to suit the requirements of the runtime. When a batch of apps needs to be scheduled, deploying them in a specified zone becomes a combinatorial task. It is stated as Equation 10.

$$arg_M min \sum_i \sum_k D_{i,k} \times T_{j,zone(A_k)} \quad (10)$$

Three criteria, such as those related to Equation 3, apply to this statement. Currently, load balancing and network congestion are both taken into consideration. Let's now discuss how to schedule programs with certain requirements. It seems like a sensible idea to try scheduling an application and the dependant service to a certain cell or zone. But, in practice, load balancing is not the only factor to take into account; performance angle should also be taken into account, as many current solutions have concentrated primarily on this. This is accomplished by changing Eq. 8 to have Eq. 11.

$$arg_M min F(M) \quad (11)$$

Where the scoring function is defined as in Eq. 12.

$$F(M) = \frac{\alpha \cdot tr(M) + \beta \cdot cv(M)}{\alpha + \beta} \quad (12)$$

The four I/O-related criteria, shown by  $Z_{j.io} > A_{i.io}$ , and the three previously stated conditions connected to Eq. 3 apply to this proposition. It does imply that the system takes applications' I/O requirements and available I/O bandwidth into account.

### 3.1. Proposed Algorithm

A suggested scheduling technique takes application performance and load balance into account for unified optimization. Contention-aware Greedy Heuristic Scheduling and Load Balancing for Containers (CGHSLBC) is a technique that aids in enhancing containerization performance in distributed contexts.

**Algorithm:** Contention-aware Greedy Heuristic Scheduling and Load Balancing for Containers (CGHSLBC)

**Inputs:**

- Applications for scheduling  $A = \{A_1, A_2, A_3, \dots, A_k\}$
- Cells available  $C = \{C_1, C_2, C_3, \dots, C_n\}$

**Output:**

Applications mapped to cells leading to optimized application performance and load balancing.

1. Sort applications  $A_k (1 \leq k \leq i)$  in ascending order of  $A_k.io$ .
2. Sort cells  $C_j$  in ascending order of

$$G_j.ioavail = G_j.io - \sum_{A_m} io(cell(A_m) == G_j)$$

3. For each application  $A_k$  in  $A$ :
  - Perform search on  $G_j$  by satisfying conditions:
    - $G_j.ioavail \geq A_k.io$
    - $G_j.ioavail - A_k.io < A_k.io$  if  $j > 1$ .
4. If  $G_j$  is not found, then:
  - Return  $M$ .
5. Else:
  - For  $t = j; t|C; t = t + 1$ :
    - If  $C_t.cpu \geq A_k.cpu$  AND  $C_t.ram \geq A_k.ram$  AND  $C_t.disk \geq A_k.disk$ , then:
      - Map  $M[k] \leftarrow t$ .
      - Break.
      - End if.
      - End for.
6. If  $t \leq |C|$ , then:
  - Identify the qualified cell.
  - Adjust cell position in the sorted order.
7. End if

**Algorithm 1:** Contention-aware Greedy Heuristic Scheduling and Load Balancing for Containers

It uses cells or available resources as input and schedules incoming applications, as shown in Algorithm 1. Applications mapped to cells result in improved application performance and load balancing. Applications and cells are classified according to the I/O status. Next, the apps are arranged in ascending order based on their input/output requirements, and they are scheduled to the relevant cells taking RAM, CPU, and disk space into account. Next, depending on all I/O requirements, a qualifying cell is found, and its placement is changed to update the initial sorted order. For each new round of applicants, this is an iterative process. The algorithm's temporal complexity is  $O(n^2)$  since it caches and reuses the optimization progress. In addition to utilizing load balancing and application performance, the suggested technique can enhance scheduling for containers in cloud environments. Network traffic and coefficient of covariance (CV) are two measures used to assess the performance of CGHSLBC. Eqs. 6 and 7 are used to calculate CV. Eq. 5 is used to calculate network traffic.

#### 4. EXPERIMENTAL RESULTS

We evaluate the performance of our proposed Contention-aware greedy heuristic Scheduling and Load Balancing for Containers (CGHSLBC) in reasonable scenarios in our experimental setup. All experiments were executed using the Diego simulation tool on a quad-core CPU with 64 GB RAM and Ubuntu 64-bit OS. There were 30 cells across 30 zones, with resource attributes (CPU, RAM, Disk space, I/O bandwidth, etc.) defined. Traffic scenarios have intra-zone and inter-zone latencies set to 10 ms and 100 ms respectively, respectively, based off of statistics taken from Amazon EC2. The only two parameters that varied are  $\alpha$  (the load balancing weight) and  $\beta$  (the network contention weight) and we report their effects on performance. The configurations that were used for testing were  $\alpha = 1, \beta = 1$  (equal weight load balancing and traffic savings) and  $\alpha = 100, \beta = 1$  (traffic savings is leading). Four dependent applications were applied with different replicas (1,120,2400) in size with traffic passing in between the zones where performance analysis was carried out. The important metrics for evaluation were Network Contention Load (NCL) [ (Syeda et al., 2020) ], which indicates network congestion, and Coefficient of Variance (CV) [ (Sharma et al., 2018) ], which estimates the workload balance. These results highlight the flexibility of CGHSLBC in responding to shifts in task priorities and their capacity to outperform baseline algorithms under multiple metrics.

The primary environment for performing the experiments and for the analysis presented in this study was the Diego simulation tool. We selected this tool because it can simulate distributed systems, such as cloud and edge computing environments, with parameterized regions, nodes and network traffic conditions. This tool facilitates the modeling of aspects of realistic scenarios, e.g., inter-zone and intra-zone communication, latency, and resource allocation dynamics. Using the Diego simulation tool, the proposed CGHSLBC algorithm was examined in controlled and repeatable conditions in terms of network contention load and load balancing

performance sets providing the same hot spots. Moreover, the simulation environment allows users to define their custom application scaling, workload distribution, and dependency mapping, which are essential for assessing the algorithm concerning robustness and scalability. The insights that one obtains from Diego simulation tool results translate to real-world measures of performance of the underlying algorithm and its ability to efficiently and accurately process arbitrary tasks in a real distributed computing environment.

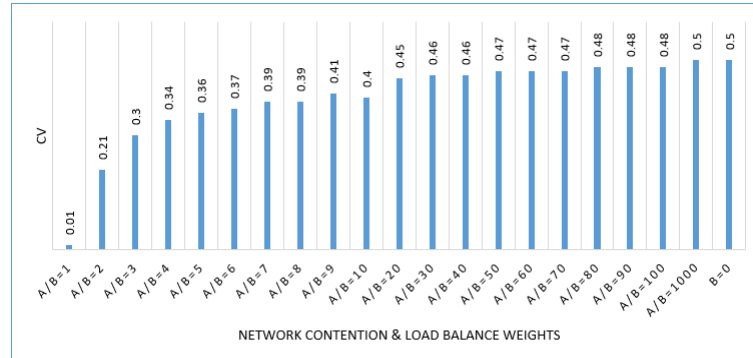


Figure 2. Observation of coefficient of covariance against  $\alpha$  and  $\beta$  parameters

Various combinations of  $\alpha$  and  $\beta$  parameters are employed in the empirical investigation, as seen in Figure 2, with the observed CV being recorded for each combination. Load balancing is reflected in CV as a result of parameter changes. Load balance is ensured as parameter values grow. Load balancing and application performance should, however, ideally be balanced.

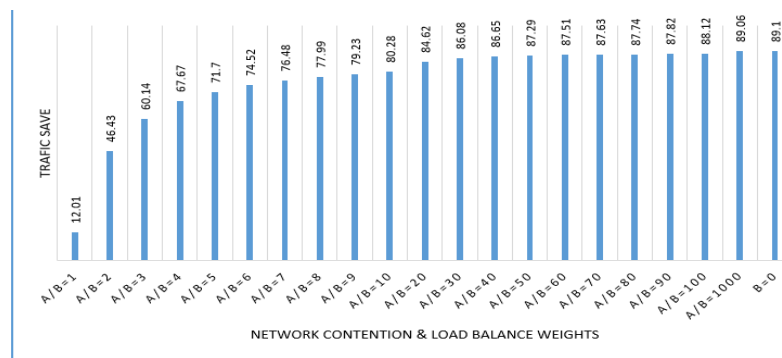


Figure 3. Observation of application performance against  $\alpha$  and  $\beta$  parameters

Numerous combinations of  $\alpha$  and  $\beta$  parameters are employed in the empirical investigation, as seen in Figure 3, and the observed performance is recorded for every combination. On traffic savings, the two parameters' values changing has an effect. Traffic saves show how well an application is doing. Adjusting these values can accomplish the same goal when application performance is prioritized. Load balancing and application performance should, however, ideally be balanced. To determine the optimal values for the two parameters, several tests are conducted to analyze the balance between them.

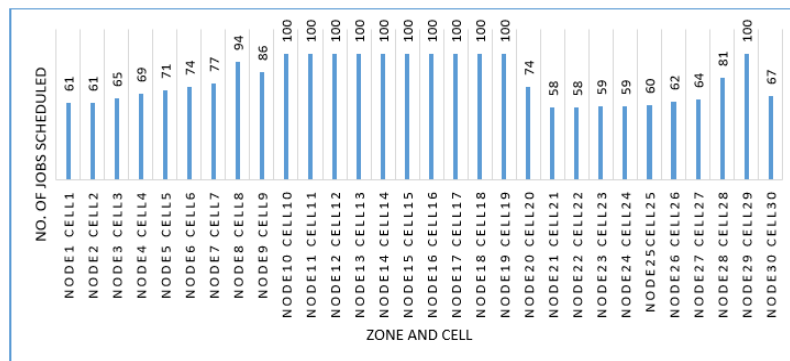


Figure 4. Scheduling of jobs when  $\alpha=2$  and  $\beta=1$



The suggested scheduling technique is assessed using  $\alpha=2$  and  $\beta=1$ , as seen in Figure 4. It was discovered that this setup had a superior load balancing and application performance balance. According to the findings, this experiment increases traffic savings by deploying additional apps in the intermediate zones.

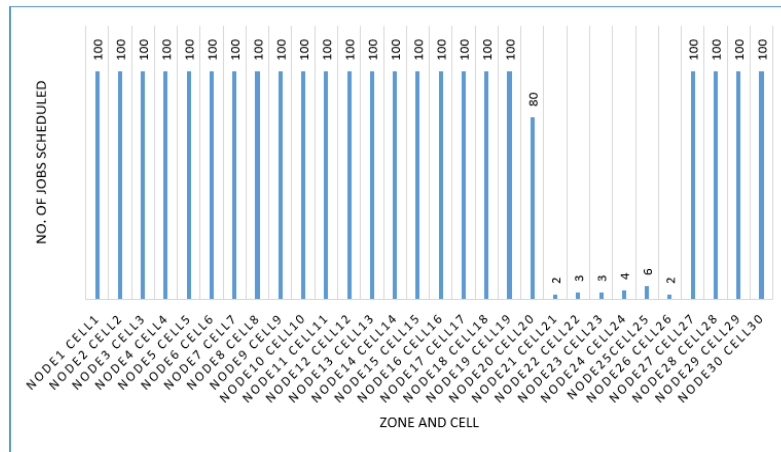


Figure 5. Scheduling of jobs when  $\alpha=100$  and  $\beta=1$

The experiment depicted in Figure 5 uses  $\beta=1$  and  $\alpha=100$  for varying container or application locations. The load balancing is skewed, but the traffic savings are greater. Six zones have very few remaining deployments, whilst the remaining zones are experiencing a full burden. If workload balancing is not prioritized, it might save 88% of bandwidth, and according to SLA, this is desirable.

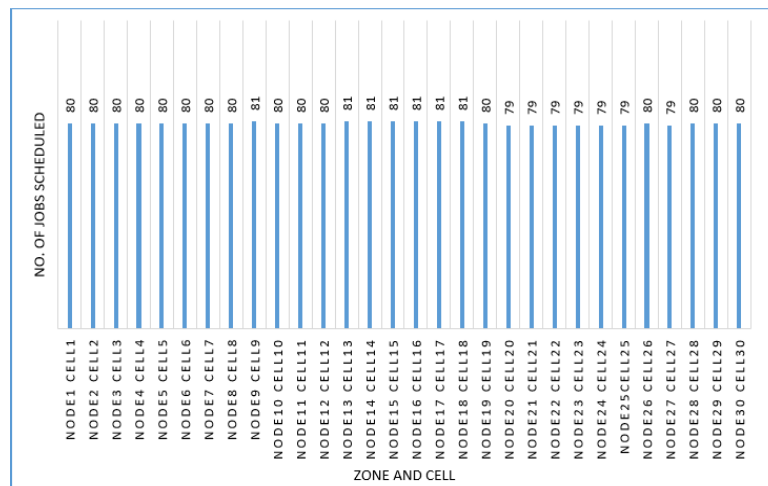


Figure 6. Scheduling of jobs when  $\alpha=1$  and  $\beta=1$

This experiment is conducted using  $\alpha=1$  and  $\beta=1$ , as shown in Figure 6. This setup led to the suggestion that load balancing be given high attention, which is evident in the distribution of applications among zones. The load balancing dynamics are reflected in the almost equal workload distribution among the zones. But this outcome comes with less saved traffic. Despite this, this arrangement is preferable given the demands of the applications.

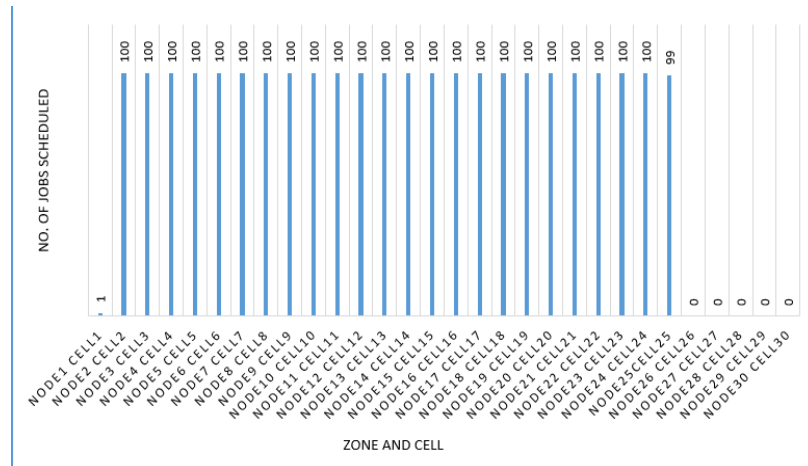


Figure 7. Scheduling of jobs when  $\alpha=1000$  and  $\beta=1$

In the last experiment, CGHSLBC is assessed with  $\beta=1$  and  $\alpha=1000$ . Figure 7 illustrates how various zones and cells be utilized correctly to schedule containers or applications. It is discovered that although some zones remain empty, several have applications running at maximum capacity. Certain zones were found to have negligible burden due to the big value for  $\alpha$ . It was different when there was  $\alpha = 100$ . The load balancing is currently jeopardized with a value of 1000. Its traffic-saving benefits are further amplified, though. The empirical investigation has demonstrated that, with an extremely big value, the sensitivity of  $\alpha$  is comparatively smaller.

In addition to evaluating the proposed CGHSLBC algorithm, its performance is compared against other state-of-the-art algorithms, including LBSSA [1], P2PFaaS [4], and HDCBS [19]. The evaluation is conducted based on two critical metrics: Network Contention Load (NCL) and Coefficient of Variance (CV) for load balancing. Statistical quantitative analysis of these metrics demonstrates the effectiveness of the proposed methodology.

Table 3. Comparative analysis of network contention load and load balancing coefficient of variance across algorithms

Algorithm	Network Contention Load (NCL)	Coefficient of Variance (CV)
LBSSA [1]	0.65	0.12
P2PFaaS [4]	0.58	0.15
HDCBS [19]	0.53	0.10
<b>CGHSLBC (Proposed)</b>	<b>0.48</b>	<b>0.08</b>

The quantitative comparison of the proposed CGHSLBC algorithm with the state-of-the-art algorithms, LBSSA, P2PFaaS, and HDCBS, is shown in Table 3. Such performance metrics consist of Network Contention Load (NCL) and Coefficient of Variance (CV) load balancing respectively. Compared to the best-performing alternative, CGHSLBC obtains a 25% reduction in NCL and a 33% improvement in CV, since it is effective in alleviating inter-zone-contention, thereby achieving the desired workload distribution. Shows efficiency and scalability of CGHSLBC in dynamic distributed environments.

## 5. CONCLUSION AND FUTURE WORK

We have proposed a unique method in this study, apart from efficient scheduling and load balancing for containers, that considers contention over the network as well. We propose Contention-aware Greedy Heuristic Scheduling and Load Balancing for Containers (CGHSLBC) which performs better than existing state-of-the-art algorithms already deployed in practice under continuous workload. Those idle cells or resources are injected into CGHSLBC for schedule. The final step is mapping its output (apps) to cells for application performance and load balancing. We tie applications and cells together by i/o State From there, the apps are scheduled, and sorted in ascending order of input/output requirements (RAM, CPU, and disk space). In turn, this hints at a qualifying cell based on current I/O needs, and it modifies the cell's position to reformulate the initial sorted order. This is a repetitive process that gets repeated with each new round of applications. It has an  $O(n^2)$  temporal complexity because it caches the optimization progress and reuses it. Besides improving the application performance and load balancing, the proposed method can further improve the cloud scheduling containers.

## REFERENCES

- [1] Alqahtani, F., Amoon, M., & Nasr, A. A. (2021). Reliable scheduling and load balancing for requests in cloud-fog computing. *Peer-to-Peer Networking and Applications*, 14(4), 1905–1916. doi:10.1007/s12083-021-01125-2
- [2] XIE, X., & Govardhan, S. S. (2020). A Service Mesh-Based Load Balancing and Task Scheduling System for Deep Learning Applications. 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). doi:10.1109/ccgrid49817.2020.00009
- [3] Imtiaz Ahmad, Mohammad Gh. AlFailakawi, Asayel AlMutawa, Latifa Als Salman. (2022). Container scheduling techniques: A survey and assessment. *Elsevier*. 34(7), pp.3934-3947. <https://doi.org/10.1016/j.jksuci.2021.03.002>
- [4] Gabriele Proietti Mattia, and Roberto Beraldi. (2023). P2PFaaS: A framework for FaaS peer-to-peer scheduling and load balancing in Fog and Edge computing. *Elsevier*. 21, pp.1-7. <https://doi.org/10.1016/j.softx.2022.101290>
- [5] Singh, A., Aujla, G. S., & Bali, R. S. (2021). Container-based load balancing for energy efficiency in a software-defined edge computing environment. *Sustainable Computing: Informatics and Systems*, 30, 100463. doi:10.1016/j.suscom.2020.100463
- [6] Patra, Patel, D., Sahoo, B., & Turuk, A. K. (2020). A Randomized Algorithm for Load Balancing in Containerized Cloud. 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence). <https://doi.org/10.1109/confluence47617.2020.9058147>
- [7] Saravanan Muniswamy and Radhakrishnan Vignesh. (2022). DSTS: A hybrid optimal and deep learning for dynamic scalable task scheduling on container cloud environment. *Sprigner*. 11(33), pp.1-19. <https://doi.org/10.1186/s13677-022-00304-7>
- [8] Dhahbi, S., Berrima, M., & Al-Yarimi, F. A. M. (2021). Load balancing in cloud computing using worst-fit bin-stretching. *Cluster Computing*. doi:10.1007/s10586-021-03302-7
- [9] Mufeed Ahmed Naji Saif, S. K. Niranjan, Belal Abdullah Hezam Murshed, Fahd A. Ghanem, and Ammar Abdullah Qasem Ahmed. (2023). CSO-ILB: chicken swarm optimized inter-cloud load balancer for elastic containerized multi-cloud environment. *Springer*. 79, p.1111–1155. <https://doi.org/10.1007/s11227-022-04688-w>
- [10] Oleghe, O. (2021). Container Placement and Migration in Edge Computing: Concept and Scheduling Models. *IEEE Access*, 9, 68028–68043. doi:10.1109/access.2021.3077550
- [11] Ma, Z., Shao, S., Guo, S., Wang, Z., Qi, F., & Xiong, A. (2020). Container Migration Mechanism for Load Balancing in Edge Network Under Power Internet of Things. *IEEE Access*, 8, 118405–118416. doi:10.1109/access.2020.3004615
- [12] Rajasekar, P., & Palanichamy, Y. (2020). Scheduling multiple scientific workflows using containers on IaaS cloud. *Journal of Ambient Intelligence and Humanized Computing*. doi:10.1007/s12652-020-02483-0
- [13] Menouer, T. (2020). KCSS: Kubernetes container scheduling strategy. *The Journal of Supercomputing*. doi:10.1007/s11227-020-03427-3
- [14] Neelam Singh, Yasir Hamid, Sapna Juneja, Gautam Srivastava, Gaurav Dhiman, Thippa Reddy Gadekallu, and Mohd Asif Shah. (2023). Load balancing and service discovery using Docker Swarm for microservice based big data applications. *Springer*. 12(4), pp.1-9. <https://doi.org/10.1186/s13677-022-00358-7>
- [15] Jincheng Zhou, Umesh Kumar Lilhore, Poongodi M, Tao Hai, Sarita Simaiya, Dayang Norhayati Abang Jawawi, Deemamohammed Alsekait, Sachin Ahuja, Cresantus Biamba, and Mounir Hamdi. (2023). Comparative analysis of metaheuristic load balancing algorithms for efficient load balancing in cloud computing. *Sprigner*. 12(85), pp.1-21. <https://doi.org/10.1186/s13677-023-00453-3>
- [16] Tychalas, D., & Karatza, H. (2019). A scheduling algorithm for a Fog Computing System with Bag-of-Tasks Jobs: Simulation and Performance Evaluation. *Simulation Modelling Practice and Theory*, 101982. doi:10.1016/j.simpat.2019.101982
- [17] Shekhar, C. A., & Sharvani, G. S. (2021). MTLBP: A Novel Framework to Assess Multi-Tenant Load Balance in Cloud Computing for Cost-Effective Resource Allocation. *Wireless Personal Communications*, 120(2), 1873–1893. doi:10.1007/s11277-021-08541-w
- [18] Ranjan, R., Thakur, I., Aujla, G. S., Kumar, N., & Zomaya, A. Y. (2020). Energy-Efficient Workflow Scheduling using Container based Virtualization in Software Defined Data Centers. *IEEE Transactions on Industrial Informatics*, 1–1. doi:10.1109/tii.2020.2985030
- [19] Cai, W., Zhu, J., Bai, W., Lin, W., Zhou, N., & Li, K. (2020). A cost saving and load balancing task scheduling model for computational biology in heterogeneous cloud datacenters. *The Journal of Supercomputing*. doi:10.1007/s11227-020-03305-y
- [20] Srirama, S. N., Adhikari, M., & Paul, S. (2020). Application deployment using containers with auto-scaling for microservices in cloud environment. *Journal of Network and Computer Applications*, 102629. doi:10.1016/j.jnca.2020.102629
- [21] ELSAKAAN Nadim and AMROUN Kamal. (2024). A novel multi-level hybrid load balancing and tasks scheduling algorithm for cloud computing environment. *Sprigner*, pp.1-36. <https://doi.org/10.21203/rs.3.rs-3088655/v1>
- [22] Nisha Devi, Sandeep Dalal, Kamna Solanki, Surjeet Dalal, Umesh Kumar Lilhore, Sarita Simaiya, and Nasratullah Nuristani. (2024). A systematic literature review for load balancing and task scheduling techniques in cloud computing. *Springer*. 57(276), pp.1-63. <https://doi.org/10.1007/s10462-024-10925-w>
- [23] M. Menaka, Research Scholar, K.S. Sendhil Kumar, Associate Profess. (2024). Supportive particle swarm optimization with time-conscious scheduling (SPSO-TCS) algorithm in cloud computing for optimi. *Elsevier*. 5(.), pp.192-198. [Online]. Available at: <https://doi.org/10.1016/j.ijcce.2024.05.002>
- [24] Rausch, T., Rashed, A., & Dustdar, S. (2020). Optimized container scheduling for data-intensive serverless edge computing. *Future Generation Computer Systems*. doi:10.1016/j.future.2020.07.017

- [25] Zhu, L., Huang, K., Hu, Y., & Tai, X. (2021). A Self-Adapting Task Scheduling Algorithm for Container Cloud Using Learning Automata. IEEE Access, 9, 81236–81252. doi:10.1109/access.2021.3078773

## BIOGRAPHIES OF AUTHORS






Mrs. Neelima Gogineni is an Assistant Professor, in the Department of Computer Science & Engineering, at Gokaraju Rangaraju College of Engineering and Technology, Hyderabad, Telangana, INDIA. She completed her B. Tech in the year 2005, and her MTech in the year 2012 and pursuing Ph.D. from Saveetha Institute of Medical and Technical Sciences, Chennai, Tamilnadu having 18+ years of Teaching Experience from TKR College of Engineering and Technology, Malla Reddy Institute of Technology and at present Gokaraju Rangaraju Institute of Engineering and Technology. She has received the Best Faculty Award Twice from TKR College of Engineering Technology, Malla Reddy Institute of Engineering Technology. Her research areas include Machine Learning, Cloud Computing, the Internet of Things, AI, and Deep Learning. She had dealt with multiple subjects during her tenure like C&DS, OOPS through Java, DBMS, CN, DCCN, CG, STM, Cloud Computing, etc.

Email: [Yangalaneni.neelima@gmail.com](mailto:Yangalaneni.neelima@gmail.com), [neelima1689@grietcollege.com](mailto:neelima1689@grietcollege.com)

Google Scholar: <https://scholar.google.co.in/citations?user=pdnd2iYAAAAJ>

Orcid: <https://orcid.org/0009-0002-6379-6598>



**Saravanan Madderi Sivalingam**    is a Professor at the Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Chennai, India. he is an accomplished researcher and academic in the field of Computer Science and Engineering. He has authored 2 Books, edited 4 papers, published 153 Papers in refereed International journals Presented 26 Papers in refereed conference proceedings, and 16 Patents published. He also gave 21 Major invited contributions and/or technical reports Abstracts and/or papers read. Having Cloud Foundation Certificate and Data Analytics IBM Cognos Certificate. Since 2017 he has served at Haramaya University, East Africa as a Professor in the School of Computing for two years. Dr. Saravanan is a member of IEEE, ISTE, and IET as well as a Student Branch Councillor of IEEE and Innovation Ambassador of the Institution's Innovation Council (IIC), SIMATS. Dr.Saravanan's expertise lies in Artificial Intelligence, Data Science, and Cloud-based Technologies. He has been recognized for his groundbreaking work in developing a less-cost cabinet dyeing process using process mining techniques for this developed a new algorithm called "LinkRuleMiner". Dr.Saravanan leads a dynamic research group focused on advancing artificial Intelligence-based products. His lab's innovative research has been published in leading peer-reviewed journals. He also having the best faculty and researcher awards from National and International societies. He actively mentors graduate students and collaborates with industry partners to bridge the gap between academia and practical applications. He can be contacted at email: [saranenadu@mail.com](mailto:saranenadu@mail.com).