Review of Computer Engineering Research
2026 Vol. 13, No. 3, pp. 244-263
ISSN(e): 2410-9142
ISSN(p): 2412-4281
DOI: 10.18488/76.v13i3.5151

An adaptive dual-threshold chunking framework for performance optimization in fog–cloud systems

Anjuli Goel[1]
Chander Prabha[2+]

[1][2]*Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab 140401, India.*
[1]*Email: goel.anjuli@gmail.com*
[2]*Email: prabhanice@gmail.com*

(+ Corresponding author)

Article History
Received: 6 January 2026
Revised: 2 April 2026
Accepted: 30 April 2026
Published: 7 September 2026

Keywords
Content-Aware chunking
Data
Dual threshold chunking
Fog cloud data deduplication
Load distribution
Threshold.

ABSTRACT

The widespread adoption of Internet of Things (IoT) applications and distributed computing systems has led to the continuous generation of large volumes of data that require efficient processing with low latency and minimal bandwidth usage. Traditional chunking approaches, particularly single-threshold mechanisms, are not suitable for dynamic fog environments due to fluctuating data arrival rates and heterogeneous node capabilities. To overcome these challenges, this study presents a Hybrid Dual Threshold Chunking (HDTC) framework aimed at improving load distribution, reducing latency, and enhancing Quality of Service (QoS) in integrated cloud–fog architectures. The proposed model combines dual-threshold-based resource management with adaptive, content-aware chunking to achieve efficient data segmentation and balanced workload allocation across fog nodes. A simulation-based implementation is developed in which users are clustered using the K-means algorithm, and fog nodes are assigned based on proximity. The system continuously monitors node utilization through upper and lower threshold limits, enabling dynamic decision-making for task allocation and chunking. Implemented in MATLAB and tested on real-world datasets consisting of text and image files, the framework is evaluated against Dynamic Prime Chunking (DPC) and Improved Dynamic Prime Chunking (IDPC). Experimental results indicate that HDTC significantly enhances performance by reducing chunking time and overall execution time by up to 95% and 85%, respectively. Although a slight decrease in delivery ratio and throughput is observed compared to IDPC, the framework improves load balancing and responsiveness, offering a scalable and efficient solution for real-time IoT and fog-based distributed computing environments.

Contribution/Originality: This work contributes to designing and evaluating a Hybrid Dual Threshold Chunking (HDTC) architecture and framework in distributed cloud-fog environments that combine adaptive dual-threshold-based resource utilization with adaptive data chunking. The strategy offers load-sensitive and context-sensitive chunk building, leading to a significant decrease in processing latency and elapsed time, and an improved Quality of Service (QoS) for heterogeneous and dynamic IoT traffic.

1. INTRODUCTION

Because of the widespread use of centralized cloud computing and Internet of Things devices, a significant volume of real-time data is created at the network. Low-latency processing and instantaneous responsiveness-based applications, such as smart health systems and industrial automation, pose a threat [1, 2]. By bringing cloud features to the network edge data source, fog computing is a paradigm that gets around these restrictions [3]. Communication

latencies are decreased by the localized data processing provided by fog nodes, which are made up of routers, gateways, and edge servers. This increases system responsiveness and bandwidth [4].

The effective management of fog computing environments is one of the main challenges, and the offloading of huge, continuous streams of data. IoT applications tend to generate heterogeneous data with non-uniform data arrival rates, which need to be processed or offloaded without overloading the limited resources available in fog nodes [5]. Chunking is an important pre-processing method to address the above-mentioned problem.

Chunking refers to dividing large data streams into smaller and manageable-sized "chunks" or data blocks, so that these can be buffered for efficient transmission, processed in parallel, or mapped across nodes [6, 7]. The method is aimed at minimizing computation and memory overhead, and maximizing scalability, fault tolerance, and load balancing [8].

The old-fashioned chunking techniques tend to adhere to one fixed rule - until a data stream reaches a particular size, they buffer all the data in that stream and convert it into a single chunk. That may seem easy, yet practically, this does not work in the fog of the real world. It is simply too unbendable, and this causes several problems [9]. First, the data streams provided by IoT are highly dynamic and characterized by high temporal variability. A single static threshold will result in undersized chunks during low load, leading to fragmentation and increased network overhead, or oversized chunks during high load, causing latency and processing delays [10]. Secondly, the computation capability, network bandwidth, and storage capacity of fog nodes are intrinsically heterogeneous. Inefficient usage and potential performance loss are caused by a static chunking policy's inability to accommodate the heterogeneity in resources between nodes [11].

For resolving such concerns, in this paper, an adaptive dual threshold chunking policy for fog computing situations has been suggested [12, 13]. Here, two threshold values are introduced: a Lower Threshold (LT) and an Upper Threshold (UT). The chunking module keeps tracking the incoming data stream and makes decisions using these thresholds. When the buffer size decreases below LT, the system waits for additional data to arrive before processing, unless in the case of time-critical events that invoke early chunking.

When the buffer size is above UT, the system creates a chunk instantaneously to prevent overflow or latency [14]. Data between LT and UT is handled normally so that the system can adapt accordingly to existing workloads and available resources.

The double threshold mechanism alleviates the latency versus fragmentation trade-off by incorporating both higher and lower bounds as a responsive, adaptive mechanism. It accommodates context-sensitive chunking, dynamically varying in real-time, improving load balancing among fog nodes, reducing processing latency, and enhancing overall Quality of Service (QoS) in distributed fog topology [15-17].

Developing an effective chunking algorithm for a fog computing environment and discussing ongoing research problems in content-aware chunking are the primary goals of this work. This paper makes the following contributions.

- Tabular study for the recent literature work.
- Designing an architectural framework of HDTC in a fog computing environment.
- Implementing a hybrid dual threshold chunking (HDTC) process, ensuring the properties of both chunking and dual threshold to enhance the system's performance. Thus, improving the overall throughput.
- Evaluation of HDTC based on QoS parameters.
- Exploring future research directions in content-aware chunking for fog nodes to facilitate advancements of the state-of-the-art operational fog environment.

Although there is an increasing amount of implementation of data chunking and fog computing, there are still a number of core constraints that are yet to be tackled in scalable fog-cloud settings.

1.1. Research Gap and Problem Statement

Although there is a substantial amount of studies on content-defined and static chunking methods, the available approaches cannot account for real-time usage of fog nodes, variability of workloads, and heterogeneity of resources in the fog cloud system. Consequently, under dynamic IoT data streams, chunking decisions are often decoupled from the load conditions of the system, leading to higher latency and poor resource utilization. This highlights a significant research gap in developing a chunking framework that can simultaneously respond to data properties and node load in the fog environment.

The problem that the present paper will address is based on the design of a load-aware and adaptive chunking mechanism that minimizes latency and processing overheads, guarantees efficient resource use, and enhances Quality of Service (QoS) in distributed fog-cloud computing systems.

1.2. Research Objectives

The specific objectives of this study are as follows:

- To implement a load-sensitive chunking system that could react to instant changes in the rate of data arrival and the utilization level of a fog node.
- To measure the effectiveness of the suggested HDTC framework with the help of major QoS indicators, such as chunking time, processing time, throughput, and chunk delivery ratio.
- To compare the effectiveness of HDTC with any existing technique of chunking in conditions of heterogeneous and dynamic workloads.

The remainder of the paper is structured as follows. The relevant work is covered in Section 2, and in Section 3, the architectural framework and guiding principles for hybrid chunking with fog computing are presented. Section 4 discusses the suggested hybrid dual-threshold chunking algorithms.

Section 5 presents the experimental section showing the performance analysis of the proposed framework. Section 6 depicts the results. The discussion is presented in Section 7 and finally, Section 8 presents the conclusion and future work.

2. RELATED WORK

From 2015 to 2025, people really pushed the boundaries of chunking techniques to meet the huge demand for smarter data storage and transmission in cloud and fog computing [18, 19]. The core of data deduplication is Chunking. It assists in tightening the belt of storage, reduces bandwidth, and simply makes all things easier to operate [20]. Previously, much was done based on content-defined chunking (CDC) using Rabin fingerprints and other tricks to ensure that deduplication was more effective [21].

At some point, the discipline changed. Scientists began deploying faster, less tolerant solutions, such as FastCDC, hybrid designs, and chunking designs designed to be used in fog computing, even with encryption to ensure data confidentiality [22-24]. More recently, attention is moving to eke out all the performance with hardware acceleration and fog-oriented deduplication that bring real-time and energy-efficient processing to where the data is created [25, 26].

Nevertheless, things do not go well all along. Finding the appropriate chunk size, maintaining costs of computation low, and security locked in is an eternal battle, particularly when fog-cloud systems may be everywhere regarding hardware and configuration [27]. Table 1 aligns the key techniques of chunking, giving their techniques, mode of operation, strengths, and weaknesses.

Table 1. Comparative analysis of various chunking methods in a cloud-fog computing environment.

Technique name & Ref No	Methodology	Evaluation metric	Advantages	Limitations
Bit String Content-aware Chunking (BCCS) [28]	Variable-length CDC using Rabin fingerprints to detect chunk boundaries based on content	Deduplication ratio, CPU energy per byte, storage overhead	Saves CPU resources and memory	Chunk-size variance; CPU-intensive fingerprinting
Asymmetric Extremum (AE) [29]	Content-based CDC using extremum markers in a sliding window to set chunk boundaries	Throughput, bandwidth efficiency, chunk-size variance	High throughput; low variance; bandwidth-efficient	Limited handling of complex boundary shifts
Fast and efficient CDC [22]	Compared to other CDC methods, this gear-based approach is quicker and more effective. FastCDC accelerates chunking by combining the three popular methods of normalized chunking, subminimum chunk cut-point skipping, and enhancing the hash-based chunking judgement.	Chunk size distribution; chunking throughput; CPU load	Improved chunking speed and the issue of the average chunk size	More complex to implement, and CPU overhead
Rapid asymmetric maximum (RAM) [21]	This technique employs two windows: one fixed-size and one variable-size, positioned differently by RAM. The variable window follows the fixed window, and the chunk boundary is determined by the position of the highest-valued byte. This ensures the cut point is at the chunk's end, reducing comparisons due to the statistical properties of the arrangement.	Chunking and the number of saved bytes per second	Reduce the cost of computing and increase throughput	NA
Minimal incremental interval (MII) [30]	One unique content-defined chunking technique used in this study is the identification of the stepped data during the backup process. Both at the source and backup, the data is divided into multiple discrete, varying-length chunks. After that, an analysis is conducted to determine the differences between the backup and source data.	Chunk throughput and chunk size variance	Resistant against byte shifting problem	NA
Bytes Frequency-Based Chunking (BFBC) [31]	The BFBC technique drastically lowers the number of calculations by employing multiple dynamic optimum parameter divisors (D) with the optimal threshold value. Additionally, it reduces the chunk-size variance and chunking throughput by utilizing BFBC's multi-operational nature	Chunking throughput and hashing throughput	Optimized storage capacity, reduces hash collision	The size of the hash table and computation cost will increase if the dataset is large
Dynamic Prime Chunking (DPC) [32]	By dynamically altering the window size inside the prime value based on the minimum and maximum chunk size criteria, DPC aims to reduce chunk duplication in cloud storage. Large chunk volatility in the deduplication system is avoided, and high	Chunking throughput and Data elimination ratio (DER)	Reduces processing time and a high ratio of DR	Able to find the redundant chunks up to some extent only

Technique name & Ref No	Methodology	Evaluation metric	Advantages	Limitations
	throughput is provided by DPC			
FogDedupe – Fog-Centric Chunking with Partial Hash & Homomorphic Encryption [33]	Source-level chunking using fixed 1 MB blocks; partial hash sent to fog node; fog checks redundancy via distributed index; ciphertext stored in cloud with homomorphic encryption	Storage overhead reduction; deduplication latency; security against confirmation-of-file attacks	27 % storage saved; 12 % latency reduction	Large fixed chunk size reduces fine-grained dedup; partial hash may raise false positives; encryption adds overhead
Hybrid Smart Chunker (OPC) [34]	Hybrid file-level for <2 KB, CDC for larger; "Optimus Prime Chunking" using prime-breakpoints without rolling hashes	Chunk processing time, throughput, chunk-size variance	17% faster chunking; constant average chunk size; >3.3× throughput	No hash/rolling window may reduce dedup precision
SeqCDC – Vectorized Sequence-Based Content-Defined Chunking [26]	CDC, without rolling hashes, uses lightweight boundary detection, content-defined skipping, and SSE/AVX vector acceleration for chunk boundary detection	Chunking throughput; deduplication ratio; CPU utilization	15× faster than scalar CDC; 20–35 % faster than other vector-accelerated chunkers; maintains deduplication effectiveness	Requires SSE/AVX support; may not generalize to all CPU architectures; implementation complexity

Dynamic Prime Chunking (DPC) [32] is an innovative content-based chunking tool that enhances the efficiency of data deduplication, particularly in cloud storage systems. DPC makes use of a hash-free method to dynamically size the window in prime terms within a user-specified minimum and maximum size. Instead of the computationally expensive hash computations of older methods, it scans the stream of data using two variable-length windows (one fixed, one dynamic) to determine the edges of chunks by detecting local maxima in the chunk, instead of the hash computations required by older algorithms. This process considerably minimizes the variation in the chunks by reducing processing time without generating too large or skewed chunks that would adversely affect deduplication effectiveness. It also enhances throughput. Prime number use of window size enhances the detection of data block duplications as it provides borders between chunks that are more distributed equally and based on the content in the data. The window-based and hash-free design of DPC improves throughput and Bytes Saved per Second (BSPS) and decreases the calculation overhead. Its great weaknesses, however, lie in its time lag reaction (because it fails to consider the potential consumption of resources at one given time) and its reduced capability to identify similar pieces. Moreover, the proper parameters adjustment, such as prime numbers, minimum and maximum window sizes, which most likely have been optimized in accordance with the dataset and storage setup, is a key factor in the success of DPC.

2.1. Critical Analysis of Existing Chunking Approaches

In spite of the fact that current chunking methods, including DPC, FastCDC, RAM, and hybrid content-defined ones, have shown improvements in deduplication efficiency and throughput, they have severe limitations in dynamic fog-cloud settings. The majority of these methods are mostly content-based and fail to provide real-time use of the fog node, variation in workload, and heterogeneity of resources in their chunking decisions. Consequently, the formation of chunks does not change with the system's conditions, causing processing stalls, uneven load allocation, and higher latency in cases of bursty IoT traffic.

Moreover, traditional chunking algorithms are optimized for deduplication ratio or computational speed but are not focused on adaptive workload balancing among distributed fog nodes. Where the capacity of nodes continuously varies with their load, and the rate of arrival and departure of tasks is very dynamic (i.e., in a continually changing fog environment), it is possible that, unless load-aware mechanisms exist, nodes can be overutilized or underutilized.

The suggested Hybrid Dual Threshold Chunking (HDTC) system addresses these drawbacks by incorporating dual-threshold-regulated load regulation with adaptive chunk boundary development. Unlike traditional methods, HDTC responds dynamically to real-time fog node utilization to allocate tasks to the core and minimize latency. With system-awareness embedded in the chunking process, HDTC can adapt chunking into a performance-based, resource-aware process that effectively supports heterogeneous fog-cloud infrastructures.

3. HYBRID METHOD

Chunking and the dual threshold policy are combined in the hybrid approach. This section incorporates the basic model working of HDTC and its flow chart.

3.1. Basic Model of HDTC

In order to model a fog computing network in a specific two-dimensional space, HDTC builds a simulation environment. To represent end devices or clients in the network, it creates a uniformly random spatial distribution of users. These users are then grouped using a k-means clustering algorithm, with a fog node at the centroid serving each cluster [35]. Effective fog node placement and resource management in edge computing scenarios are made possible by this configuration, which makes it easier to analyze and visualize user-to-fog node assignments.

In some situations, fog computing is better than cloud computing because it allows data processing and analysis to take place nearer to the data source, like user terminals, sensors, or Internet of Things devices. Since latency is greatly decreased by this closeness, fog computing is perfect for real-time or applications that require quick decisions, like industrial automation, autonomous cars, and healthcare monitoring [36]. Furthermore, fog computing saves bandwidth and lessens network congestion by minimizing the need to send massive amounts of raw data to remote cloud servers. Besides that, fog computing enhances the resilience and reliability of systems, since it can continue its operation even when internet connectivity is patchy or there is no connectivity at all, which is quite handy in remote or mobile locations. All things considered, fog computing provides a scalable, economical, and effective way to meet the increasing demands of distributed computing systems in situations where cloud-only solutions might not be sufficient [37].

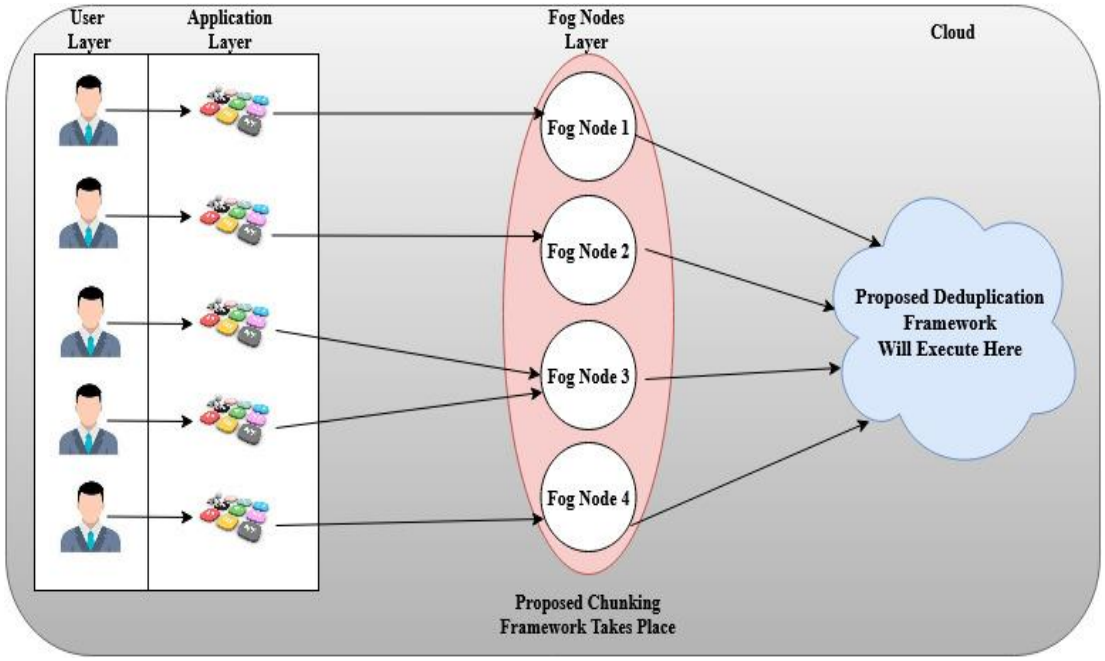

Figure 1. Basic working of the proposed framework.

The basic operation of the suggested hybrid model is depicted in Figure 1. A multi-layered architecture intended to implement a suggested chunking and deduplication framework in a cloud and fog computing environment is depicted in the provided diagram. The four different layers of the architecture are the User Layer, Application Layer, Fog Nodes Layer, and Cloud Layer. Different end users generating data through their devices form the User Layer. The data first travels to the Application Layer, where it is processed by various programs and prepared for the next phase. After that, the data is moved to the Fog Nodes Layer, which is essentially a collection of fog nodes (Fog Node 1, Fog Node 2, Fog Node 3, and Fog Node 4). This is where the chunking framework comes in, where the data is divided into chunks before getting it to the cloud. By the time the information gets to the Cloud Layer, it has already been chunked and processed, and the deduplication model comes into action. It identifies and eliminates any redundant information; thus, you will not have to waste space, and you will not have to deal with redundancies. This hybrid system, which combines cloud computing with fog computing, keeps everything under control and enables the system to utilize the resources more wisely.

3.1.1. System Initialization and User Placement

- The simulation environment is initialized by defining a two-dimensional network area in which users are uniformly distributed. Let N_{users}denote the total number of users deployed in the network. The number of fog nodes $N_{\text{fog}}$is determined as a function of the user population and is computed as one-tenth of the total number of users, as given in (1). For the considered simulation scenario, $N_{\text{users}} = 50$, which results in $N_{\text{fog}} = 5$.
- Number of Fog Nodes:

$$N_{\text{fog}} = \frac{N_{users}}{10} \quad (1)$$

Where $N_{\text{users}} = 50 \implies N_{\text{fog}} = 5$.

User locations are generated randomly within the simulation area using a uniform distribution. The coordinates of the i^{th}user are calculated along the x-axis and y-axis according to (2), where $A_x$and $A_y$represent the maximum dimensions of the network area. In this study, both $A_x$and $A_y$are set to 1000 units, defining a square deployment region.

- Random User Placement:

$$x_i = A_x \cdot \text{rand}(), \; y_i = A_y \cdot \text{rand}()$$
$$\text{for i} = 1, 2, \ldots, \text{Nusers}, \quad \text{and} \quad \text{Ax} = \text{Ay} = 1000 \quad (2)$$

The position of each user is represented by the coordinate pair (x_i, y_i), where $i = 1,2, \ldots, N_{\text{users}}$. These spatial coordinates are stored for subsequent processing, including fog node association, workload distribution, and visualization of the network topology.

3.1.2. User Clustering and Fog Node Localization

To efficiently associate users with fog nodes, K-means clustering is applied to partition the set of users into multiple clusters, where each cluster is served by a single fog node. The number of clusters is set equal to the number of fog nodes, N_{fog}. Clustering is performed based on the spatial proximity of users in the two-dimensional network area.

The Euclidean distance metric is used to measure the distance between a user and a cluster centroid during the clustering process. The distance between the i^{th}user located at $\mathbf{x}_i = (x_i, y_i)$and the centroid of the j^{th}cluster $\mathbf{c}_j = (c_{jx}, c_{jy})$is computed using (3).

- Euclidean Distance (for clustering):

$$d(x_i, c_j) = \sqrt{(x_i - c_{jx})^2 + (y_i - c_{jy})^2} \quad (3)$$

The centroid of each cluster is updated as the arithmetic mean of the positions of all users assigned to that cluster. Specifically, the centroid $\mathbf{c}_j$ of the j^{th} cluster is calculated according to (4), where S_j denotes the set of users associated with cluster j, and $|S_j|$ represents the total number of users in that cluster.

- Centroid Calculation:

$$\mathrm{cj} = \frac{1}{|S_j|} \sum_{x_i \varepsilon S_j} x_i \quad (4)$$

During each iteration of the K-means algorithm, users are assigned to the cluster whose centroid yields the minimum Euclidean distance. The cluster assignment for the i^{th} user is determined using (5), ensuring that each user is associated with the nearest fog node.

- Cluster Assignment:

$$\mathrm{idx_i} = \arg\min_j d(x_i, c_j) \quad (5)$$

Upon convergence of the clustering process, the resulting centroids (c_{jx}, c_{jy}) correspond to the optimal fog node locations, as they minimize the intra-cluster distances between users and their serving fog nodes. This clustering-based localization enables efficient user–fog association and reduces communication latency within the fog computing environment.

3.2. Visualization and Data Structuring

The simulation displays user positions, cluster memberships, and fog node locations.

- Visualization Mapping:
- Users: (x_i, y_i).
- Fog Nodes: (c_{jx}, c_{jy}).
- Cluster Membership: $\mathrm{idx_i}$.

Figure 2 shows the graphical representation of user clustering and fog node localization, and Figure 3 represents the allocation of users to the fog node.

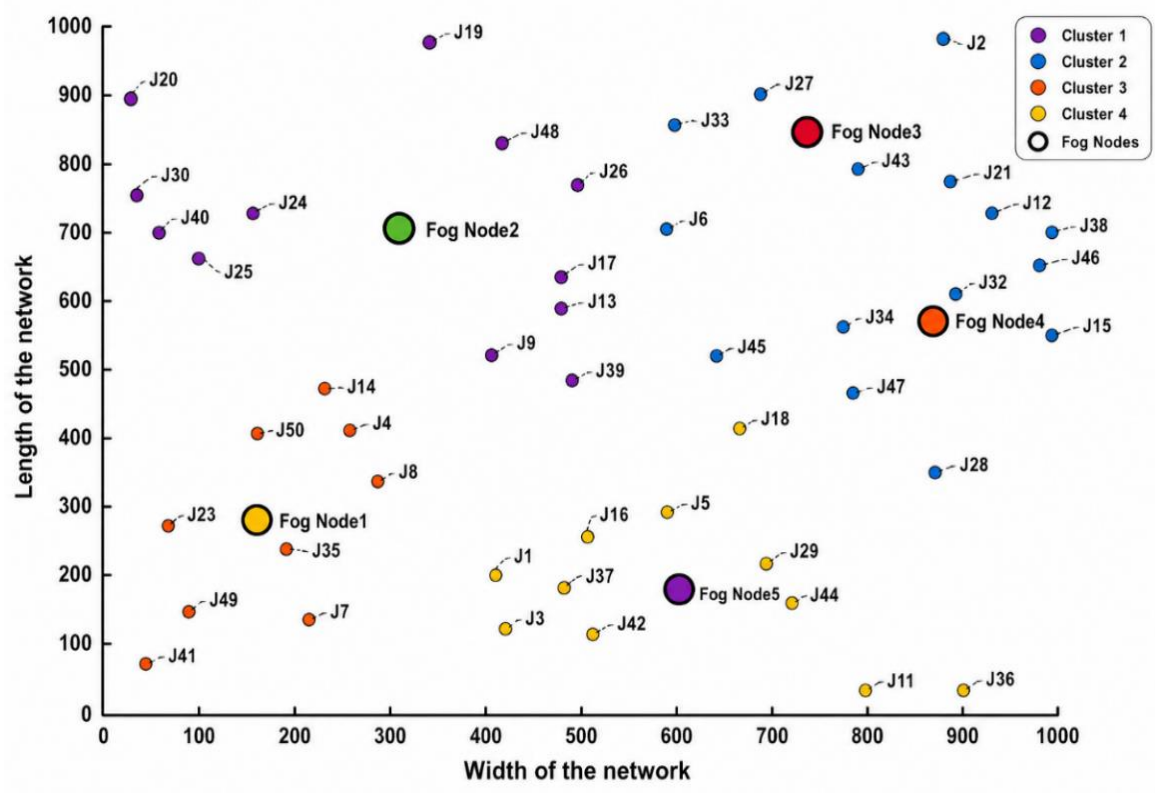

Figure 2. The graphical representation of user clustering and Fog node localization.

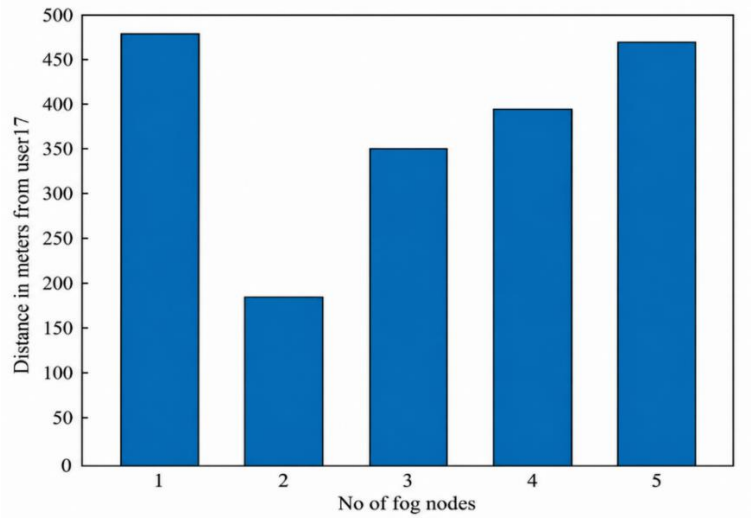

Figure 3. Allocation of a user to the Fog node.

3.3. Flowchart

The framework of HDTC combines chunking with the dual threshold technique. The working of hybrid chunking with two different levels of thresholds has been explained in this section. Implementing chunking enacts the properties of chunking, such as data segmentation, redundancy elimination, fault tolerance, scalability, and compression friendliness, while dual threshold ensures adaptive control, load balancing, storage optimisation, data flow regulation, and reduces elapsed time. The flowchart for the basic steps of HDTC is shown in Figure 4.

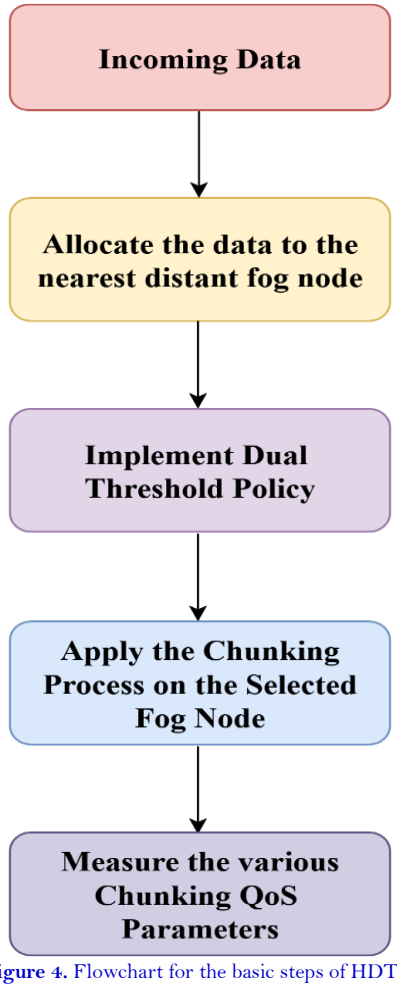

Figure 4. Flowchart for the basic steps of HDTC.

4. PROPOSED WORK

There are two components to the proposed work, HDTC: the dual threshold policy will be used to determine which fog node is best for communication. The optimal fog node means that the node should not be overrunning or overusing resource utilization. The second part involves the chunking process, where data will be converted into chunks.

4.1. Dual Threshold Policy

Dual threshold-based resource utilization in fog computing is a dynamic load balancing strategy that uses two adaptive thresholds, upper and lower, derived from the average flow rate of tasks [38]. A fog node is classified as underutilized, overutilized, or neutral by comparing its current utilization against these thresholds. Tasks are either allocated to underutilized nodes or migrated from overutilized ones to maintain optimal system performance, minimize latency, and ensure energy-efficient task distribution, often considering node proximity for localized task offloading [39, 40]. Each fog node is categorized into one of three states.

$$\text{State}_i(t) = \begin{cases} Underloaded, & if\ Ui(t) < Tl \\ Optimal, if\ Tl \leq Ui(t) \geq Tu \\ Overloaded, & if\ Ui(t) > Tu \end{cases}$$

Where,

Ui(t) = Current resource utilization of fog node i at time t.

Tl = Lower Threshold.

Tu = Upper Threshold.

The average current utilization Ui of the fog network is calculated by analyzing the flow rates of all active fog nodes. Each node maintains a value representing how much workload it is currently handling. By averaging these values across all nodes, the system obtains a single representative value that reflects the overall load status of the network at that moment. This value is then used to determine whether a node is underutilized in Equation 6, overutilized in Equation 7, or operating normally based on predefined thresholds. Figure 5 presents a dual threshold policy. If the node is overutilized, then there is a need to identify a less-utilized node. If the node is underutilized, then the task is allocated to that node only, and thus, there is no need for migration to other fog nodes. If the node is neither overutilized nor underutilized, then the system is running in its optimal conditions. Pre -defined upper and lower thresholds are given in Equations 6 and 7, respectively.

$$Tl = \text{Average flow rate} - \text{average flow rate} * \theta / 100 \quad (6)$$

$$Tu = \text{Average flow rate} + \text{average flow rate} * \theta / 100 \quad (7)$$

Where the value of θ is 20%, which is within the margin of 10%-30%, if 20% of the time, the fog node may be busy [38]. It was taken to ensure a balance between underutilized and overutilized systems.

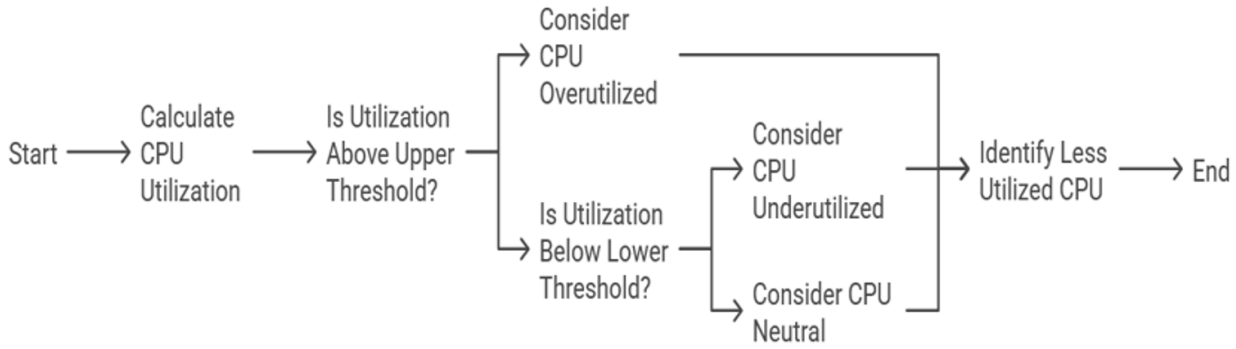

Figure 5. Flowchart for the dual threshold policy.

After the identification of an overutilized node, it is necessary to allocate the task to nearby underutilized nodes. So, it checks the threshold of all nodes in the vicinity, approximately 30% from the current location node. Then, the task is migrated to an underutilized node. This dual threshold approach effectively balances the computational load among fog nodes, minimizes service latency, and optimizes energy consumption. Figure 6 clearly shows the flowchart for the complete proposed dual threshold policy.

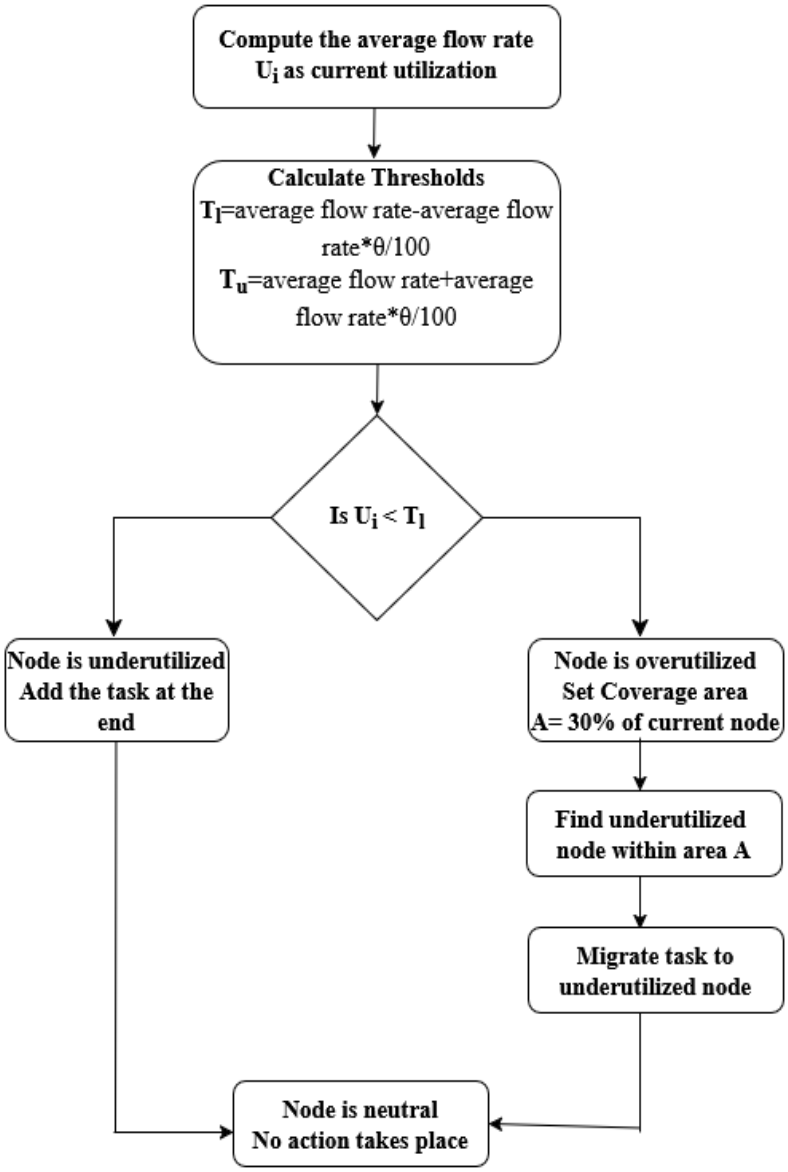

Figure 6. Dual Threshold Policy Flowchart for the HDTC.

4.2. Chunking Process

Chunking is the process of dividing large data streams or files into smaller, more manageable pieces. This localized processing reduces the load on the cloud by decreasing latency and improving bandwidth use. By processing only pertinent or smaller portions of data before sending it to the cloud, chunking in fog computing facilitates real-time analytics, enhances data deduplication, and promotes effective storage and transmission.

The presented algorithm dynamically divides a data stream into manageable chunks based on a self-adjusting window known as the *dynamic window (dw)*. This method is beneficial in adaptive fog systems where data packet sizes vary, and optimal transmission or processing efficiency is required. Figure 7 illustrates the chunking flowchart for the HDTC. And this flow chart explains the common approach to content-defined chunking, the sliding window-

based chunking methodology. When chunking, boundaries are established based on the substance of the data rather than preset sizes. Better deduplication and storage optimization occur.

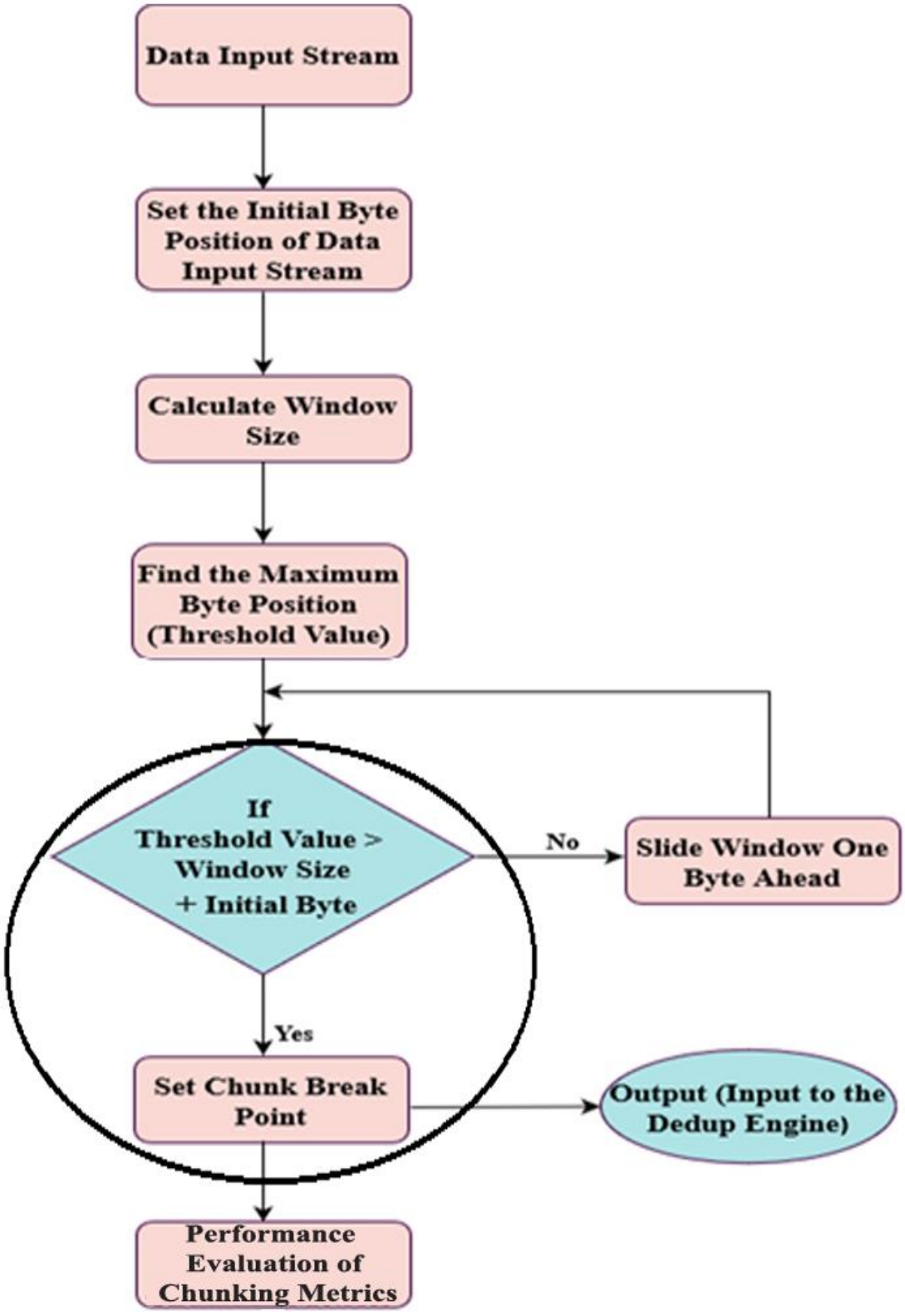

Figure 7. Flowchart for the HDTC.

A sliding window-based process of chunking to achieve optimal data deduplication in fog or cloud computing is depicted in the flowchart above. The procedure is activated through a stream of continuous data input and an initial

byte position to determine the beginning of the analysis. The number of bytes to be processed in each iteration is calculated by determining a window size (ws). The algorithm detects the highest byte position, also known as the threshold value, within this window, which is a condition for detecting possible chunk boundaries. To check if the threshold value is greater than the window size plus the first byte, a conditional test is performed. The window shifts one byte ahead, and the process remains if the condition is not met. When checked out, the system will drop a chunk break point at the point it demands to be. That new piece goes directly to the deduplication engine, which deletes any duplicated data. The chunk breakpoint cb is determined using (8), where it is set as one position beyond the detected data window boundary dw_1. This ensures that chunk segmentation starts immediately after the window delimiter, avoiding overlap between consecutive chunks and enabling efficient data processing. Lastly, the original byte position sequence continues as:

$$cb = dw1 + 1 \qquad (8)$$

Where, cb is the chunk break point.

Figure 8 provides a clear concept of setting the chunk break point. The remaining length of data is considered the last chunk.

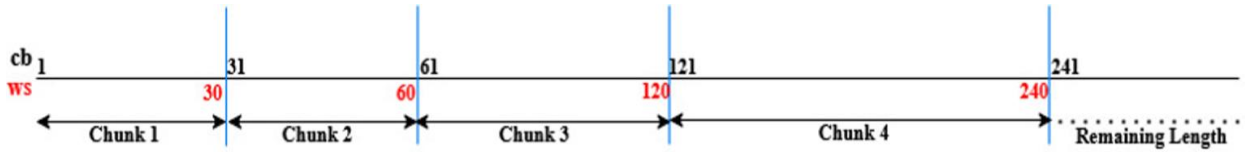

Figure 8. Sets the chunk break point.

Then the system examines factors such as chip size and the amount of information downsized to determine the effectiveness of chunking. Specifically, in fog-based systems, this approach to computing chunk boundaries based on real data, rather than fixed ones, enhances deduplication and simplifies data storage management. The improved dynamic chunking algorithm is presented below.

Algorithm 1:

IDPC (data, bits_left)

Dynamic window (dw)=(1024/100)*prime chunk size/(2.71828-1)

chunks_created=0

cb=1 // initial max

dw1=dw

While bits_left>0

Chunk value=dw1

cb=1+dw1; // chunk breakpoint setting

dw1=dw1+dw1

chunks_created=chunks_created+1

dynamic_chunks.chunks_created=chunk value

Else

chunk value=bits_left

chunks_created=chunks_created+1

dynamic_chunks.chunks_created=chunk value

bits_left=-1

end

return bits_left

end

end function.

5. EXPERIMENTAL SETUP

The MATLAB tool, namely MATLAB R2020a, has been used to implement our proposed work. MathWorks created MATLAB, an interactive platform, numerical computing environment, and high-level programming language. For technical computing tasks including numerical analysis, visualization, algorithm prototyping, simulation, and the implementation of scientific and engineering systems, it is extensively utilized in both academia and industry. The following configuration describes the machine used for experimentation: Windows 11 and an Intel Core i5-1035G1 CPU with 4 cores, 8 logical processors, and 16GB of RAM operating at 1.00GHz and 1190MHz. Our distributive communication tests were conducted through a series of experiments involving the distribution of chunks and throughput using the system. First, we examined the duration of data processing time and its performance with chunks. In this, we used real-world data such as text files and images. The textual files had the format of a text file. Regarding images, we experimented with the .png, .jpg, .jpeg, and .bmp files. The details of the dataset are shown in Table 2.

Table 2. Details of the dataset.

Type of data	No. of files
.txt files	1024 files
.png, .jpg, .jpeg, and .bmp images	1000 images

Spatial user distribution, K-means-based fog node clustering, dual-threshold load regulation, and adaptive chunk formation are used in the analysis to determine the system's performance when the workload is dynamically changing. Contrary to previous research, where benchmark or static data were used, synthetic and locally generated text and image data are used to model heterogeneous and bursty IoT traffic under fog conditions in clouds. Such a controlled arrangement allows systematic experimentation of latency, processing time, throughput, and the ratio of chunk delivery. To make simulation parameters and configurations transparent and reproducible, all simulation parameters and configurations are explicitly specified.

6. RESULTS

In the results section, the performance of the proposed chunking techniques is evaluated and compared for both the HDTC and DPC approaches. The assessment is carried out using multiple datasets, including text-based files and image files, to ensure a comprehensive comparison. Furthermore, the DPC method and its enhanced variant, IDPC, are benchmarked against HDTC. The evaluation focuses on several key performance metrics to determine the efficiency and effectiveness of these approaches.

6.1. Chunk Count

The number of chunks in a dataset is the chunk count that an algorithm can generate during dataset processing. A smaller chunk count usually indicates less metadata processing, fewer indexing operations, and increased storage efficiency. The results in Figure 9 show the average number of chunks among the three methods. Both DPC and IDPC exhibit nearly the same behavior with a mean of approximately 3.6 chunks. However, HDTC has slightly more, at about 4.6. According to these values, HDTC demonstrates approximately 27.7 percent improvement in chunk detection compared to DPC and IDPC. This suggests that, in addition to faster chunking times, HDTC also performs better in recognizing meaningful chunk boundaries. Overall, it can be considered a very effective combination of speed and accuracy, making it highly suitable for working with big data.

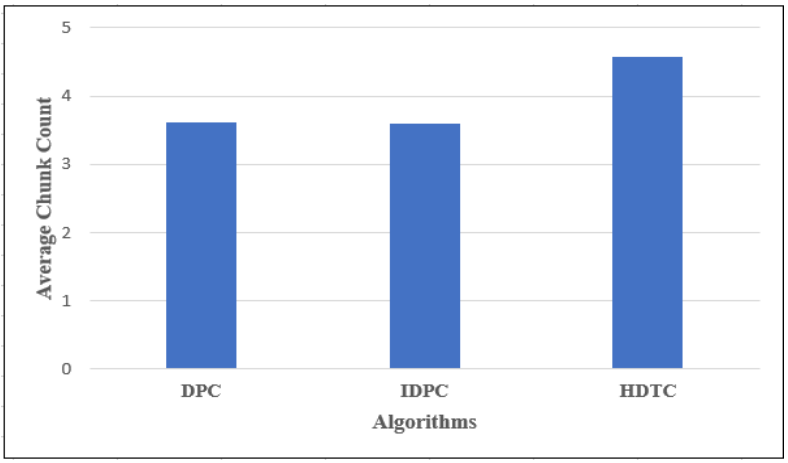

Figure 9. Comparative analysis of Chunking Count.

6.2. Chunk Time

The chunking time measure shows the duration of each algorithm in splitting the data set into smaller units. This demonstrates that it is a robust measure of the processing efficiency of the given algorithm and the applicability of the algorithm to time-sensitive or large-scale applications. Based on average chunking times depicted in Figure 10, DPC exhibits the lowest performance, at about 3.45 seconds. IDPC performs much better, with the time being no more than 1.05 seconds, approximately 69.6% better. However, HDTC provides the best result, taking only 0.15 seconds to complete the process. This makes it approximately 95.7% faster than DPC and 85.7% faster than IDPC. These results clearly demonstrate HDTC's ability to perform tasks where processing speed is a major requirement.

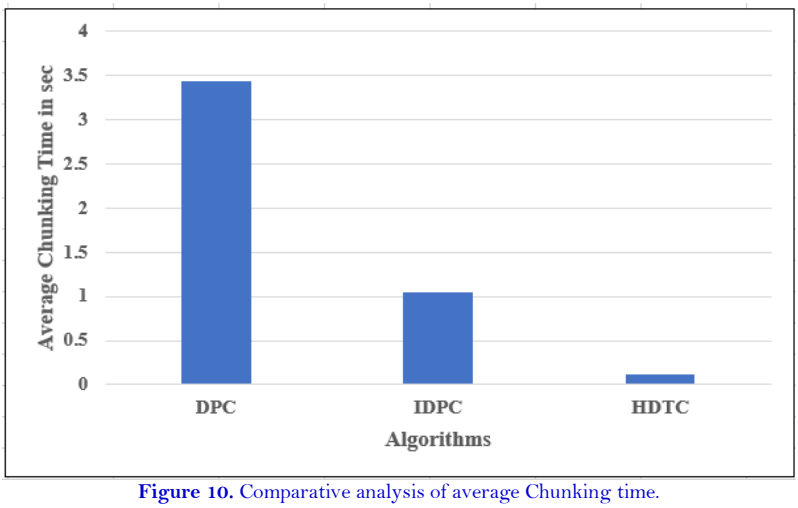

Figure 10. Comparative analysis of average Chunking time.

6.3. Average Elapsed Time Analysis

The average elapsed time is a general measure of the time it takes to complete the chunking process, including finding chunk boundaries and additional processing. This metric has been compared across three algorithms: DPC, IDPC, and HDTC, as shown in Figure 11. Results indicate that DPC is the slowest, with an average time of about 3.45 seconds, primarily due to a higher computational workload. IDPC performs better, reducing this overhead to approximately 1.05 seconds, representing an improvement of about 69.5%. HDTC demonstrates the best performance, completing the task in nearly 0.15 seconds. This makes it just over 95.7% faster than DPC and 85.7% faster than IDPC. These findings clearly show that HDTC offers superior scalability and speed, making it suitable for applications where processing time is critical.

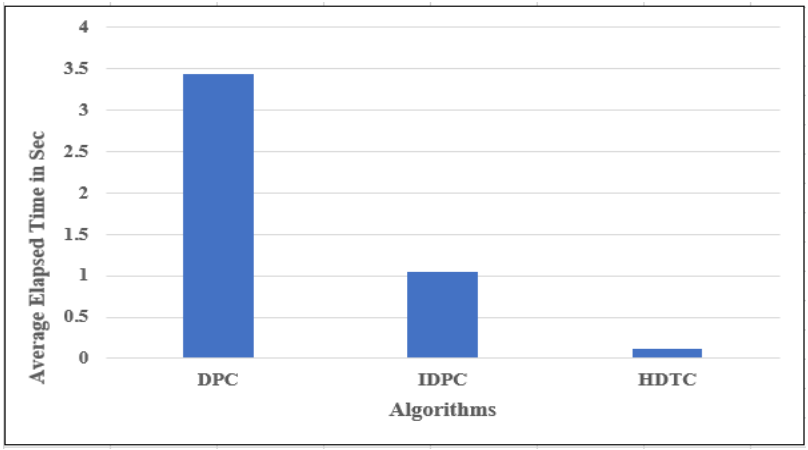

Figure 11. Comparative analysis of average Elapsed Time.

6.4. Average Processing Time Analysis

The mean parameter of processing time indicates the duration required by the chunking technique to perform data segmentation and related processes. This is a key performance factor because long delays can impact system scalability and throughput. As Figure 12 shows, the average processing time for DPC and IDPC is approximately 6.0 seconds (100%), while the innovative HDTC framework reduces this time by 25 percent to about 4.5 seconds. This improvement demonstrates that HDTC can accelerate chunk processing while maintaining system flexibility. Although DPC and IDPC have nearly identical processing times, HDTC is the clear leader because it significantly reduces latency, which is crucial in cloud and fog computing for real-time processing.

6.5. Chunk Delivery Ratio

The chunk delivery ratio (CDR) indicates the proportion of chunks successfully delivered and reassembled compared to those generated. A higher CDR reflects greater reliability, which is crucial for maintaining data integrity during transmission and de-duplication processes. Figure 13 compares the CDR of HDTC, IDPC, and DPC. Both DPC and IDPC achieve a perfect delivery ratio of 1.0, meaning all generated chunks are delivered and reconstituted completely. Conversely, HDTC has an average CDR of approximately 0.78, indicating a success rate of 78 percent, which is 22 percent lower than the other two methods. Although HDTC aims to balance improved throughput, dynamic load management, and resource efficiency, this result shows it sacrifices some reliability for these benefits. DPC and IDPC, on the other hand, are more accurate in delivering chunks.

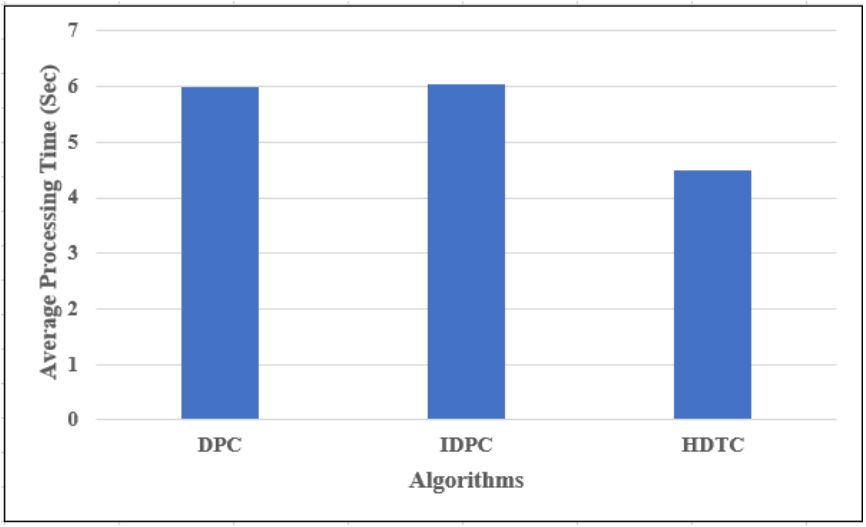

Figure 12. Comparative analysis of average Processing time.

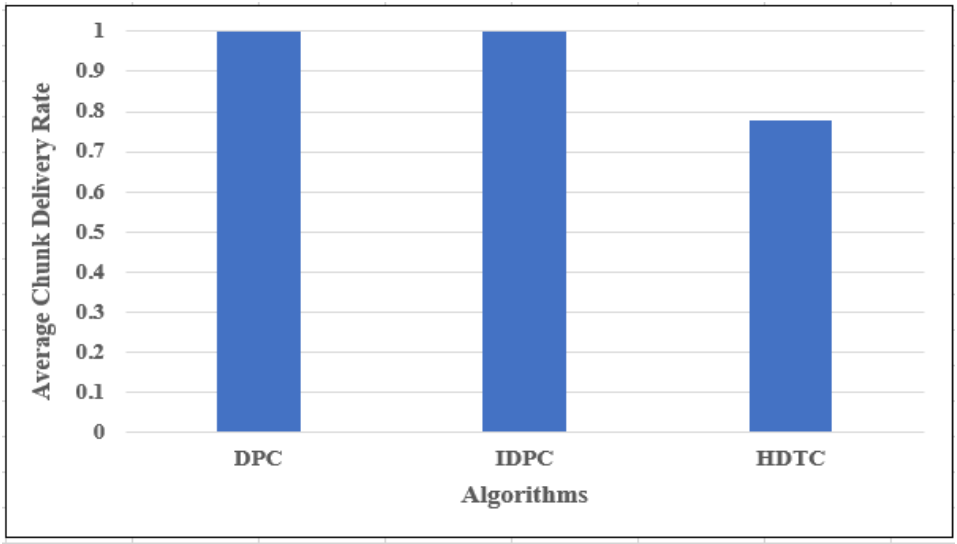

Figure 13. Comparative analysis of the average Chunk rate.

6.6. Chunk Throughput

The chunking throughput measure is used to measure the quantity of data that was processed in the process of chunking in a specific period. Better efficiency and scalability, as indicated by higher throughput, are important to systems with large or quickly moving datasets. Figure 14 shows that there is a big variation in the performance of DPC, IDPC, and HDTC, as indicated by the throughput of these three in Figure 14. DPC is the point of reference at approximately 42 Mbps. Comparatively, IDPC achieves approximately 145 Mbps, or approximately 245 percent higher than DPC, demonstrating its superior chunk processing performance. Although still approximately 48 percent lower than IDPC, the proposed HDTC registers approximately 75 Mbps, which is already a 79 percent improvement over DPC. This indicates that HDTC can be a viable option in fog and cloud systems where throughput is not a major concern, and performance and reliability are key priorities, but it has higher throughput than DPC.

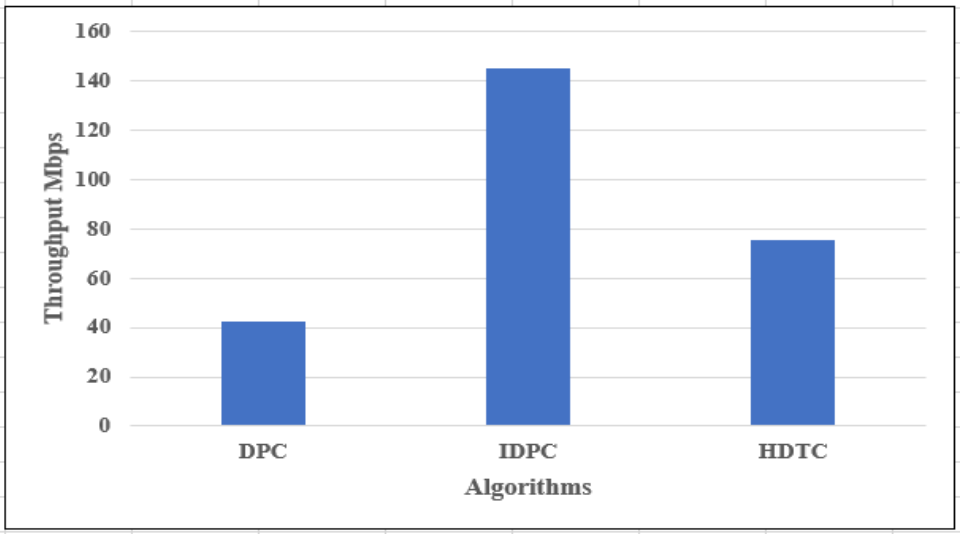

Figure 14. Comparative analysis of average Chunking throughput.

7. DISCUSSION

The effectiveness of the proposed Hybrid Dual Threshold Chunking (HDTC) model in processing data in a fog computing environment with the highest optimization is demonstrated through experimental studies. HDTC is highly reliable and flexible compared to other processes, such as Dynamic Prime Chunking (DPC) and Improved Dynamic Prime Chunking (IDPC) in terms of chunking time, elapsed time, and overall processing efficiency. Its

findings show that the dual thresholds approach is a good strategy for allocating dynamic workloads, reducing processing delays, and ensuring effective movement of tasks between fog nodes. However, there are trade-offs that are also identified through analysis. Compared with DPC and IDPC, HDTC has a higher rate of chunks, which results in increased overhead in metadata management. Additionally, HDTC exhibits slightly lower throughput and chunk delivery ratio, indicating areas that require fine-grained optimization to reduce intermittent chunk loss and improve scalability with high-speed streams. Despite these disadvantages, the method is superior to DPC in real-time Internet of Things applications, reducing chunking and elapsed time by over 96 percent. Based on this explanation, HDTC is a better option under various fog conditions, especially when combined with system load sensitivity and the ability to dynamically detect chunk boundaries. The findings clearly indicate that HDTC is a viable alternative for low-latency and real-time deduplication processes, making it suitable for current fog and cloud computing environments.

8. CONCLUSION AND FUTURE WORK

In this paper, Hybrid Dual Threshold Chunking (HDTC) was proposed as a solution to the inefficiency of traditional fixed and content-determined chunking models used in cloud fog computing. The proposed algorithm combines both dual-threshold-based load management and adaptable creation of chunk boundaries and allows making load-conscious decisions of load-available chunking depending on the usage of fog nodes and the heterogeneous nature of IoT workloads. In contrast to current methods of operation that are not system resource-dependent, HDTC harmonizes data segmentation with infrastructure dynamics to enhance system functioning in general.

The experimental analysis, conducted through simulation, proved that HDTC produces significant changes in the stimulating time of a chunking procedure, processing time, and latency, outperforming similar procedures under baseline techniques, including DPC and IDPC. Load-awareness improves workload utilization between nodes of a Fog system and enables better Quality of Service (QoS), especially in bursty and latency-sensitive IoT applications. Although there is a slight trade-off in the ratios of chunk delivery and throughput under certain circumstances, the framework remains consistent upon repeated trials and is therefore robust.

The major suggestion of this paper is that chunking of that system of fog-cloud must be handled as a resource-conscious optimization process instead of a data-driven one. Nonetheless, the evaluation being done at the moment is restricted to the simulated environment based on workload models expressed in synthetic parameters and might not reflect the entire operational complexities in the real world.

Future efforts will be carried out through real-time implementation on distributed cloud infrastructures, integration with intelligent load prediction schemes, and testing using standard benchmarking datasets to further evaluate scalability, adaptability, and practical applicability.

Funding: This study received no specific financial support.
Institutional Review Board Statement: Not applicable.
Transparency: The authors state that the manuscript is honest, truthful, and transparent, that no key aspects of the investigation have been omitted, and that any differences from the study as planned have been clarified. This study followed all writing ethics.
Competing Interests: The authors declare that they have no competing interests.
Authors' Contributions: Both authors contributed equally to the conception and design of the study. Both authors have read and agreed to the published version of the manuscript.

REFERENCES

[1] M. Rzepka, P. Boryło, M. D. Assunção, A. Lasoń, and L. Lefèvre, "SDN-based fog and cloud interplay for stream processing," *Future Generation Computer Systems*, vol. 131, pp. 1-17, 2022. https://doi.org/10.1016/j.future.2022.01.006

[2] A. Mijuskovic, A. Chiumento, R. Bemthuis, A. Aldea, and P. Havinga, "Resource management techniques for cloud/fog and edge computing: An evaluation framework and classification," *Sensors*, vol. 21, no. 5, p. 1832, 2021. https://doi.org/10.3390/s21051832

[3] B. M. Alencar, R. A. Rios, C. Santana, and C. Prazeres, "FoT-Stream: A Fog platform for data stream analytics in IoT," *Computer Communications*, vol. 164, pp. 77-87, 2020. https://doi.org/10.1016/j.comcom.2020.10.001

[4] F. Alenizi and O. Rana, "Dynamically controlling offloading thresholds in fog systems," *Sensors*, vol. 21, no. 7, p. 2512, 2021. https://doi.org/10.3390/s21072512

[5] N. Mehran, D. Kimovski, and R. Prodan, "A two-sided matching model for data stream processing in the cloud–fog continuum," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, *IEEE*, 2021, pp. 514-524.

[6] S. Bebortta, S. S. Tripathy, U. M. Modibbo, and I. Ali, "An optimal fog-cloud offloading framework for big data optimization in heterogeneous IoT networks," *Decision Analytics Journal*, vol. 8, p. 100295, 2023. https://doi.org/10.1016/j.dajour.2023.100295

[7] T. Pfandzelter and D. Bermbach, "Towards a benchmark for fog data processing," presented at the 2023 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2023.

[8] F. Mehdipour, B. Javadi, A. Mahanti, and G. Ramirez-Prado, *Fog computing realization for big data analytics. In R. Buyya & S. N. Srirama (Eds.), Fog and edge computing: Principles and paradigms.* Hoboken, NJ: Wiley, 2019.

[9] I. Shahid, P. Kang, P. Lama, and S. U. Khan, "Adaptive scheduling of continuous operators for IoT edge analytics," *Future Generation Computer Systems*, vol. 149, pp. 104–121, 2024.

[10] N. Chauhan and R. Agrawal, "A probabilistic deadline-aware application offloading in a multi-Queueing fog system: A max entropy framework," *Journal of Grid Computing*, vol. 22, no. 1, p. 31, 2024. https://doi.org/10.1007/s10723-024-09753-7

[11] B. Sedlak, V. C. Pujol, A. Morichetta, P. K. Donta, and S. Dustdar, "Adaptive stream processing on edge devices through active inference," *arXiv preprint arXiv:2409.17937*, 2024. https://doi.org/10.48550/arXiv.2409.17937

[12] Y. Xu, J. Wu, and H. Wu, "QoS-aware adaptive data processing in edge-fog computing," *IEEE Transactions on Services Computing*, vol. 17, no. 3, pp. 1230–1243, 2024.

[13] S. Singh, P. Kumar, and M. Conti, "Latency-aware task partitioning for heterogeneous fog systems," *IEEE Internet of Things Journal*, vol. 12, no. 4, pp. 3451–3463, 2025.

[14] K. Luo, Z. Wang, and X. Chen, "Dynamic dual-threshold resource management for edge analytics," in *Proceedings of the IEEE International Conference on Fog and Edge Computing (ICFEC)*, 2025, pp. 75–84.

[15] R. Gupta, A. Dutta, and M. Gerla, "Context-aware data stream partitioning in fog-cloud continuum, " *IEEE Transactions on Cloud Computing. Advance Online Publication*, 2025.

[16] J. Martinez, P. Rodriguez, and T. Rausch, "Intelligent buffer management for real-time IoT analytics in edge environments," *IEEE Access*, vol. 13, pp. 45231–45245, 2025.

[17] A. Goel and C. Prabha, "Leveraging data deduplication approaches in the cloud storage: A succinct review," *Procedia Computer Science*, vol. 259, pp. 1640-1651, 2025. https://doi.org/10.1016/j.procs.2025.04.119

[18] A. A. Sadri, A. M. Rahmani, M. Saberikamarposhti, and M. Hosseinzadeh, "Data reduction in fog computing and internet of things: A systematic literature survey," *Internet of Things*, vol. 20, p. 100629, 2022. https://doi.org/10.1016/j.iot.2022.100629

[19] M. Gregoriadis, L. Balduf, B. Scheuermann, and J. Pouwelse, "A thorough investigation of content-defined chunking algorithms for data deduplication," *arXiv preprint arXiv:2409.06066*, 2024. https://doi.org/10.48550/arXiv.2409.06066

[20] Y. Fu, J. Su, J. Ning, J. Wu, Y. Lu, and N. Xiao, "Distributed data deduplication for big data: A survey," *ACM Computing Surveys*, vol. 58, no. 3, pp. 1-38, 2025. https://doi.org/10.1145/3735508

[21] R. N. S. Widodo, H. Lim, and M. Atiquzzaman, "A new content-defined chunking algorithm for data deduplication in cloud storage," *Future Generation Computer Systems*, vol. 71, pp. 145-156, 2017. https://doi.org/10.1016/j.future.2017.02.013

[22] W. Xia *et al.*, "FastCDC: A fast and efficient content-defined chunking approach for data deduplication," presented at the 2016 USENIX Annual Technical Conference (USENIX ATC 16), 2016.

[23] M. S. Yoosuf and R. Anitha, "Low latency fog-centric deduplication approach to reduce IoT healthcare data redundancy," *Wireless Personal Communications*, vol. 126, no. 1, pp. 421-443, 2022. https://doi.org/10.1007/s11277-022-09752-5

[24] X. Tang, J. Chen, and L. Zhou, "FCDedup: A two-level deduplication system for encrypted data in fog computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 10, pp. 2642–2656, 2023.

[25] S. Udayashankar, A. Baba, and S. Al-Kiswany, "Accelerating data chunking in deduplication systems using vector instructions," *ACM Transactions on Storage*, vol. 22, no. 4, pp. 1-27, 2026. https://doi.org/10.1145/3797270

[26] S. Udayashankar, A. A. Mahmoud, and S. Al-Kiswany, "Vectorized sequence-based chunking for data deduplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 37, no. 4, pp. 934-947, 2026. https://doi.org/10.1109/TPDS.2026.3660793

[27] G. Said *et al.*, "Fog-assisted de-duplicated data exchange in distributed edge computing networks," *Scientific Reports*, vol. 14, no. 1, p. 20595, 2024. https://doi.org/10.1038/s41598-024-71682-y

[28] B. Zhou, S. Zhang, Y. Zhang, and J. Tan, "A bit string content aware chunking strategy for reduced CPU energy on cloud storage," *Journal of Electrical and Computer Engineering*, vol. 2015, no. 1, p. 242086, 2015. https://doi.org/10.1155/2015/242086

[29] Y. Zhang *et al.*, "AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," presented at the 2015 IEEE Conference on Computer Communications (INFOCOM), IEEE, 2015.

[30] C. Zhang *et al.*, "MII: A novel content defined chunking algorithm for finding incremental data in data synchronization," *IEEE Access*, vol. 7, pp. 86932-86945, 2019. https://doi.org/10.1109/ACCESS.2019.2926195

[31] A. S. M. Saeed and L. E. George, "Data deduplication system based on content-defined chunking using bytes pair frequency occurrence," *Symmetry*, vol. 12, no. 11, p. 1841, 2020. https://doi.org/10.3390/sym12111841

[32] M. Ellappan and S. Abirami, "Dynamic prime chunking algorithm for data deduplication in cloud storage," *KSII Transactions on Internet & Information Systems*, vol. 15, no. 4, pp. 1342–1359, 2021. https://doi.org/10.3837/tiis.2021.04.009

[33] M. S. Yoosuf, C. Muralidharan, S. Shitharth, M. Alghamdi, M. Maray, and O. B. J. Rabie, "FogDedupe: A fog-centric deduplication approach using multi-key homomorphic encryption technique," *Journal of Sensors*, vol. 2022, no. 1, p. 6759875, 2022. https://doi.org/10.1155/2022/6759875

[34] M. Ellappan and A. Murugappan, "A smart hybrid content-defined chunking algorithm for data deduplication in cloud storage," *Soft Computing*, vol. 28, no. 15, pp. 9037-9052, 2024. https://doi.org/10.1007/s00500-023-09290-7

[35] H. Zhang, Y. Li, X. Wang, and Z. Cao, "Efficient fog node placement using nature-inspired metaheuristic for IoT applications," *Cluster Computing*, vol. 27, pp. 8225–8241, 2024.

[36] A. Khalili, M. Sharifi, and F. Taleai, "Clustering of mobile IoT nodes with support for scheduling of time-sensitive applications in fog and cloud layers," *Cluster Computing*, vol. 25, pp. 3531–3559, 2022.

[37] M. Jasim and N. Siasi, "Hybrid-grained migration method for load redistribution in heavily-loaded fog nodes," *Cluster Computing*, vol. 27, no. 7, pp. 9497-9507, 2024. https://doi.org/10.1007/s10586-023-04255-9

[38] J.-P. Yang, "Dynamic threshold-based resource management for fog computing environments," *IEEE Access*, vol. 13, pp. 87898-87908, 2025. https://doi.org/10.1109/ACCESS.2025.3571394

[39] S. Rashmi *et al.*, "AI-powered VM selection: Amplifying cloud performance with dragonfly algorithm," *Heliyon*, vol. 10, no. 19, p. e37912, 2024. https://doi.org/10.1016/j.heliyon.2024.e37912

[40] R. Singh and S. Singh, "An improved VM selection and allocation hybrid algorithm using grasshopper and firefly in cloud computing," *International Journal of Embedded Systems*, vol. 17, no. 3-4, pp. 251-266, 2024. https://doi.org/10.1504/IJES.2024.144365