Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH **Open Access**

HybOff: a Hybrid Offloading approach to improve load balancing in fog environments

Hamza Sulimani[1,2*], Rahaf Sulimani[1], Fahimeh Ramezani[2], Mohsen Naderpour[2], Huan Huo[3], Tony Jan[4] and Mukesh Prasad[2]

Abstract

Load balancing is crucial in distributed systems like fog computing, where efficiency is paramount. Offloading with different approaches is the key to balancing the load in distributed environments. Static offloading (SoA) falls short in heterogeneous networks, necessitating dynamic offloading to reduce latency in time-sensitive tasks. However, prevalent dynamic offloading (PoA) solutions often come with hidden costs that impact sensitive applications, including decision time, networks congested and distance offloading. This paper introduces the Hybrid Offloading (HybOff) algorithm, which substantially enhances load balancing and resource utilization in fog networks, addressing issues in both static and dynamic approaches while leveraging clustering theory. Its goal is to create an uncomplicated low-cost offloading approach that enhances IoT application performance by eliminating the consequences of hidden costs regardless of network size. Experimental results using the iFogSim simulation tool show that HybOff significantly reduces offloading messages, distance, and decision-offloading consequences. It improves load balancing by 97%, surpassing SoA (64%) and PoA (88%). Additionally, it increases system utilization by an average of 50% and enhances system performance 1.6 times and 1.4 times more than SoA and PoA, respectively. In summary, this paper tries to introduce a new offloading approach in load balancing research in fog environments.

Keywords Fog computing, Load balancing, Resource management, Offloading, Time-sensitive applications

Introduction

Central computing has emerged as a prevalent concept in various fields in the Internet era, supported by 5G access networks. Central computing systems encompass technologies that empower enterprises to collect, process, analyze, and archive data from distributed clients worldwide [13]. This concept has become so integral to the Internet that reverting to primitive, decentralized systems are no longer feasible [8, 53, 54]. In practice, cloud computing represents the tangible implementation of the central computing concept. It has gained widespread recognition as the ideal infrastructure for efficiently managing widely distributed Internet of Things (IoT) devices [3]. IoT, a telecommunication system facilitating data exchange among interconnected objects over a public network, streamlines operations with minimal human intervention [6]. As a fundamental framework, IoT enables cloud computing to interact with the environment, facilitating the widespread adoption of IoT technology and the gradual growth of its data. However, it also presents implications for the efficiency of public networks [3, 28].

Numerous critical applications rely on the same public network infrastructure, designed to support all cloud-connected applications [16, 37]. When slowdowns occur

*Correspondence:
Hamza Sulimani
Hhhsulimani@uqu.edu.sa
[1] College of Computing, Umm Al-Qura University, Makkah, Saudi Arabia
[2] Australian Artificial Intelligence Institute, School of Computer Science, Faculty of Engineering and Information Technology, University of Technology Sydney, Sydney, Australia
[3] School of Computer Science, Faculty of Engineering and Information Technology, University of Technology Sydney, Sydney, Australia
[4] Centre for Artificial Intelligence Research and Optimization (AIRO), Design and Creative Technology, Torrens University, Sydney, Australia

in the public network, time-sensitive applications such as e-health, smart grids, and unmanned vehicles, which have strict timing requirements for proper functioning, are severely affected [48]. Cloud computing often needs help to consistently deliver the required level of service for these time-sensitive applications due to the unpredictable efficiency of public networks [53, 54].

To address this challenge, Cisco introduced a new layer seamlessly integrated into cloud computing, forming fog computing (FC). FC integrates storage, computing, and networking at the network's edge, reducing data transfer to the cloud, lowering latency, and enhancing efficiency [5, 17]. This technology is crucial for decentralized computing, especially in real-time IoT applications. However, the continuous growth in the number of IoT devices and their generated data, along with the unpredictable nature of distributed IoT clients, places an increasing load on fog servers [60].

These factors drive researchers in FC to enhance the resource management system, particularly the load balancing (LB) system. A wide range of LB in cloud systems have been proposed. However, the diverse structures of FC have led researchers to introduce a different type of LB algorithm other than the central system, cloud computing. LB aims to allocate incoming tasks among servers with limited resources to prevent overloading or underutilizing fog resources. Effective LB management is vital to maintaining a stable computing environment and improving network availability and flexibility especially for time-sensitive applications [7].

A steering algorithm is required to direct user requests to the most suitable fog server based on application requirements to achieve effective fog load balancing. Offloading is the primary mechanism for relieving overloaded servers, thus balancing load in a distributed system [18]. A well-designed resource allocation policy is essential for creating an effective offloading strategy to balance load [34, 35]. In general, there are two fundamental approaches to offloading: static and dynamic [48]. Most recent offloading algorithms favour the dynamic approach due to its superior features compared to the static algorithm [52]. However, prevalent dynamic offloading or prevalent offloading approach (PoA) does have inherent drawbacks, including decision-making time, increased offloading messages, and distance-related issues [48]. These challenges result in significant network costs, often considered hidden expenses. Many articles view these costs as a trade-off for the reliability gained from dynamic approaches [51].

The motivation for this research is rooted in the pressing need to address the formidable challenges posed by large-scale networks and time-sensitive issues, which, despite various studies on the subject [4], have yet to see a comprehensive solution that considers the hidden expenses associated with these challenges. The imperative drives the impetus for this research to meet the escalating demands of time-sensitive applications in a world characterized by the continuous proliferation of IoT devices. Cisco's introduction of FC, which seamlessly integrates storage, computing, and networking at the network's edge, is a notable development [5]. With its capability to reduce data transfer to the cloud, diminish latency, and improve efficiency, FC represents a significant step forward. However, the critical need remains to establish an effective resource management system, particularly an LB system, to optimize the utilization of FC resources and establish a stable environment for time-sensitive applications. This research aims to develop a solution that simultaneously tackles the challenges of fog load balancing for large-scale networks, particularly in the context of time-sensitive applications. It introduces a novel approach, a hybrid algorithm, designed to benefit from previous solutions be selecting what suits the research goal of creating a low-cost and highly efficient solution to tackle these issues simultaneously and ensure the selection of a suitable destination server for offloading. The research questions guiding this paper include:

1. How can fog load balancing be improved to efficiently support time-sensitive applications, such as e-health and unmanned vehicles?
2. What is the impact of offloading strategies on fog load balancing, and how can the hidden expenses associated with dynamic offloading be minimized?
3. Can a hybrid load-balancing algorithm that combines the strengths of both static and dynamic offloading approaches provide a comprehensive solution to these challenges?

We introduce a hybrid load-balancing algorithm that combines the strengths of both static and dynamic offloading approaches. The proposed algorithm offers five key contributions to fog load balancing:

1. It is reintroducing static offloading by deeply understanding its drawbacks to reuse it through hybrid offloading.
2. It minimises message exchanges generated in the system to satisfy the offloading requirements. Even though these messages are essential, the proposed solution utilises many techniques to keep this number at the bottom.
3. It reduces decision-making time for offloading. This time is one of the requirements of dynamic offloading, which most state-of-the-art algorithms try to trade off to keep it down. The proposed solution engages the reuse of the static technique to solve this problem.

4. It encourages servers to handle time-sensitive applications locally, eliminating the need for global allocation. The proposed solution is designed to allow this application to be executed locally to avoid the complicated offloading costs.
5. It efficiently manages networks of all sizes using a cell-based approach, reducing latency, alleviating network congestion, and enhancing LB. As most distributed computing systems, such as fog computing, can expand dramatically and make the offloading cost more costly and cannot be ignored, our solution to tackle this problem is considered an excellent contribution to the field.
6. Comprehensive experiments evaluate our algorithm from various perspectives, illustrating its superiority over other state-of-the-art fog LB algorithms in extensive studies.

Our work marks the implementation of the true essence of hybrid offloading, merging static and dynamic offloading behaviour. Additionally, the proposed algorithm incorporates various features, including a central-distributed control system, fog server clustering, and prioritization of critical applications while also addressing hidden expenses such as distance-based offloading, decision messages, and network congestion. In comparison to the static offloading approach (SoA) and the PoAs, the experiments demonstrate that the proposed algorithm enhances LB by 52.1% and 38.2%, improves system performance by 60% and 38.8%, and increases the system utilization ratio by 62.4% and 42.7% compared to SoA and PoA, respectively.

The rest of this paper is organized as follows. The next Section presents the literature review. "Hybrid Approach to Enhance Load Balancing" section describes the proposed algorithm in detail. "Experiments and Results" section shows the experiments and results, followed by the discussion and conclusion in "Discussion" and "Conclusion" sections.

Literature review

In this Section, the literature review explains the foundational concept of FC systems and the LB strategies devised to enhance offloading.

Fog computing

FC, a pivotal concept in the realm of distributed computing, is engineered to support Internet of Things (IoT) applications efficiently, especially those demanding real-time responses [44]. As a complement to traditional cloud computing, it aspires to leverage edge resources strategically positioned closer to end-users [12]. The core objective is reducing reliance on remote cloud data centers, reducing latency, and decreasing network bandwidth requirements. Embracing FC presents various innovative advantages, including cost savings in cloud operations and fortified system stability [7].

However, the continuous proliferation of IoT devices and the surge in data generation has strained FC's capacity to meet performance expectations [3]. This strain is particularly acute in specialized applications, especially time-sensitive ones. Varied growth rates in user density across different regions have resulted in an uneven distribution of workloads, causing some fog servers to become overloaded while others remain underutilized [23]. This imbalance leads to resource wastage and misalignment within the fog layer [19, 51]. To tackle these challenges, researchers have explored dynamic offloading as a potential solution [31, 32, 36, 42, 56, 57, 60]. Notwithstanding the merits of FC, due to inherent resource limitations within the fog layer, certain applications necessitate offloading to the cloud, emphasizing the enduring significance of web-based computing applications [1].

To better comprehend the structure of computing networks in the proposed system, Fig. 1 illustrates the three interconnected layers. Cellular or WiFi networks are wireless links connecting fog servers to client servers in the IoT edge layer [24]. The Internet serves as the primary medium connecting the fog layer and the cloud [34, 35]. Within the fog layer, tasks are managed by surrounding fog servers, with results forwarded to the source server if necessary. The cloud layer is dedicated to specific purposes, such as heavy processing or data archiving. This research focuses on applications predominantly processed within the fog layer [31, 32].

All user-sent applications adhere to a standard operational algorithm, as outlined by Mukherjee, Shu et al. [44]:

1. Edge servers receive application requests from end-users.
2. Received applications are decomposed into a set of sub-tasks for distribution.

Heavy fog servers either redirect the sub-tasks to idle fog servers for processing or add them to their processing queues. The processing results are subsequently sent back to the original server.

Challenges in task offloading in fog computing

While FC is classified as an evolved extension of the cloud computing system to handle IoT-related problems and shortcomings at the network edge, in FC, processing nodes are distributed and heterogeneous. Furthermore, the services based on fog technology must work with various aspects of the restricted environment. Therefore,

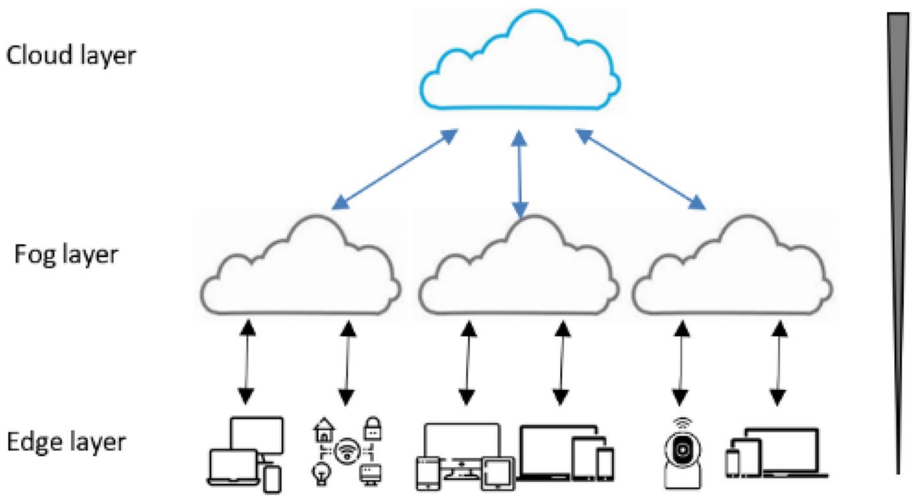


Fig. 1 Fog computing architecture [52]

discovering the challenges of task offloading in FC is essential [43]:

Network dynamics challenges

Dynamic network conditions IoT networks are fast-varying access networks that produce dynamic network and traffic conditions. This behaviour is a substantial challenge that adds extra complexity during the task offloading problem. Offloading prediction can be conjoined with a resource allocation mechanism at the fog level since the amount of resources needed for the task execution is directly proportional to the network traffic that will end up at the edge of the network.

Dynamic user behaviour In task offloading, another level of restriction is added by the unexpected behaviour of the users, which is difficult to foresee. Hence, data analytics and machine learning techniques should be used to assess the users' behaviour and the rate of task generation.

Resource allocation challenges

Task offloading is extremely impacted by the resource allocation mechanisms determining where and how the offloaded tasks will be executed in a distant device. Therefore, the resource allocation and task offloading decisions are connected to be addressed together.

Partitioning decision The first and core of the task offloading problem is deciding which task to offload. The offloading algorithm contains an intelligent mechanism designed to decide whether to execute the generated task locally or to be offloaded to another device. Some associated costs are due to this partition decision of the tasks, such as energy consumption, task execution, and transmission delay. A flawed partitioning decision may cause performance bottlenecks.

Resource availability The availability of the system resources is crucial to enhance the performance of an application. Although the cloud has a massive amount of the system, using these resources significantly delays the overall system. Consequently, utilizing the edge resources is a crucial challenge requiring an efficient management mechanism and resource allocation to ensure performance requirements.

Task management At the fog layer, one of the core benefits of Edge Computing is that its infrastructure is usually spread over multiple geographical sites, which gives the system minimal execution time. However, a meticulous strategy of the task management control modules is required at the Edge [40].

The research problem

According to our observation, we have noticed that all the PoAs use the present system state theme, in which the heavy servers read the environment (gathering attributes) to give an offloading action to redirect the excess tasks to the target server. This is repeated several times when there is a necessity for extra resources. Obviously, this theme generates a high volume of exchanged messages with the peer servers; we can call it *decision messages*. They seek to explore unused resources to cover the shortage in the affected areas without any intention to increase the number of served servers [51].

As illustrated in Fig. 2, an infinite series of interaction and offloading processes among servers will occur unabated due to the unchanging quantity or quality of physical resources in the field. These processes make the network situation seem to get worse over time. Keeping the *networks congested* will escalate the latency in the system [11], which is a decline in the main objective of FC.

However, expanding the network, either partially or entirely, appears to be the logical solution to support the affected areas. Moreover, "serve more clients, earn more profit" is the goal of most network operators [29]. Expanding their coverage can increase their number of clients. Increasing coverage is an excellent metric by which to evaluate any network as an increase in network cover is an increase in rank. This behaviour increases the number of servers in the fog layer consequently entering the network in the state known as *network oversizing* in case the expanding has not a sufficient planning. Moreover, increasing the number of fog servers will increase system availability. However, it can negatively impact dynamic offloading. Unfortunately, dynamic offloading in this type of network may offload some tasks to remote servers because most algorithms have no limits on distance. The system outcomes will be the worst if the task is time sensitive. Consequently, this adds an extra burden on the network bandwidth and total execution time due to messages travelling among remote servers. Hence, dynamic offloading is affected again by *distant offloading*, networks congested, and offloading decisions, which makes it less effective. Although all research approaches pursue enhancing offloading strategies, they ignore these hidden costs.

However, the expansion decision may not be suitable if there is uncertainty about the full utilization of all fog servers, especially considering the varying efficiency of LB algorithms. Therefore, finding an efficient offloading strategy with a low-cost is the key for the dilemma of the research.

Related works

The offloading technique is a pivotal solution for LB aimed at conserving computing and storage resources,

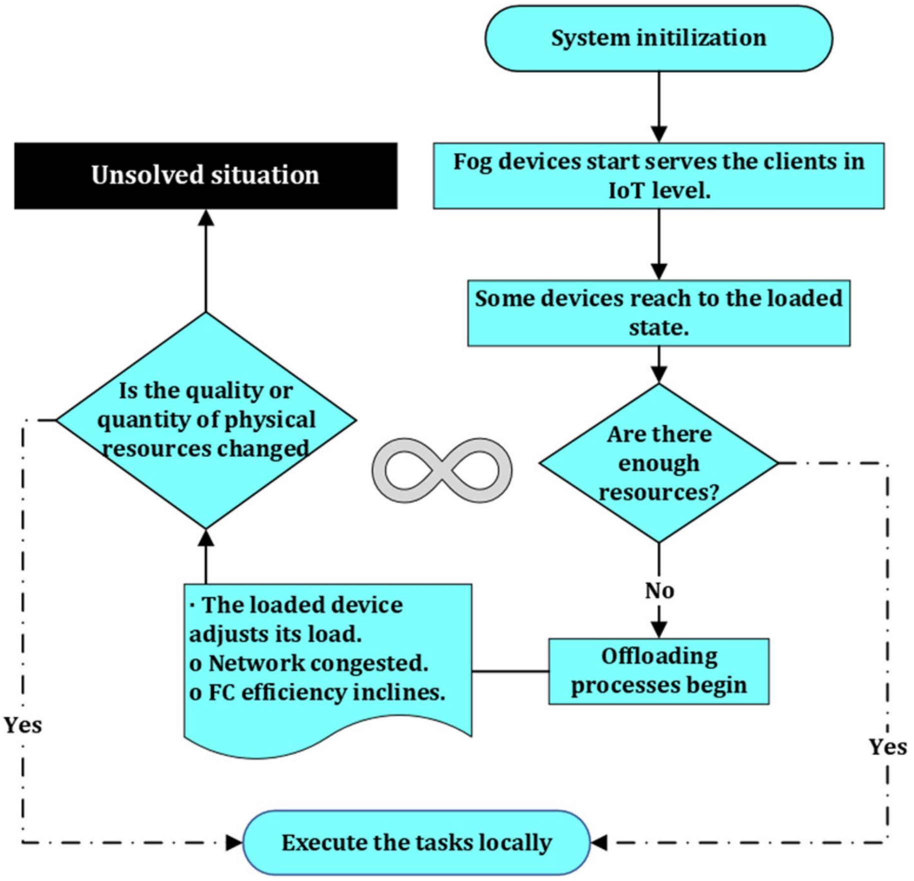

System initilization
Fog devices start serves the clients in IoT level.
Some devices reach to the loaded state.
Are there enough resources?
No
Yes
Offloading processes begin
· The loaded device adjusts its load.
o Network congested.
o FC efficiency inclines.
Is the quality or quantity of physical resources changed
Yes
Unsolved situation
Execute the tasks locally

Fig. 2 Prevalent offloading process flowchart

particularly in decentralized systems [56, 57]. A plethora of research efforts are dedicated to minimizing inefficiencies. However, prevalent task offloading schemes have unavoidable hidden costs due to their specific requirements. These costs include offloading decisions, distance offloading, and network congestion [48]. Conversely, low-cost static offloading encounters numerous challenges, such as reliability concerns, that make it inefficient for use. This Section delves into relevant publications and prior works that validate the algorithm's novelty, successfully addressing many of these obstacles.

In dynamic offloading, overloaded servers continuously gather data from other fog servers to distribute incoming tasks among the active servers [55]. Once the system evaluates and processes this collected data, it makes an offloading decision, typically referred to as a 1-out-n process, where it selects the optimal target server (1) from among the available options (n) [50]. However, this decision process leads to network congestion due to the periodic exchange of critical messages known as decision messages [34, 35, 47]. In addition to network congestion, it also introduces high communication latency when identifying the target servers for offloading, termed decision latency [59]. While decision messages and decision latency may be minimal individually, they occur continually in affected areas, collectively impacting the effectiveness of dynamic offloading when following this approach.

On the other hand, the primary goal of most network operators is to 'serve more clients, earn more profit' [26]. Expanding their coverage can increase the number of clients they serve, making coverage expansion a valuable metric for evaluating any network, as it correlates with increased network rank [46]. While expanding the number of fog servers enhances system availability, it can have a negative impact on dynamic offloading. In such networks, dynamic offloading may offload tasks to remote servers, as many algorithms have no distance limits [26, 31, 32]. This action can result in unfavourable outcomes, particularly for time-sensitive tasks, adding a burden on network bandwidth and total execution time due to messages travelling among remote servers [42]. Therefore, distant offloading and offloading decisions hinder the effectiveness of dynamic offloading.

However, there are severe consequences if the fog system fails to deliver the expected services. Many critical applications that have recently emerged are time-sensitive, including unmanned vehicles, healthcare, and the smart grid [14, 20, 30]. These applications rely on the fog layer for proper operation, where any delay can lead to catastrophic outcomes [45]. Network congestion is another adverse effect. The conventional offloading approach increases the number of messages in the network due to present system state requirements (decision messages) and distant offloading (in some algorithms). Consequently, the network infrastructure can deteriorate rapidly [14].

Various LB algorithms and solutions have been proposed. In [25], the authors introduce an energy-efficient offloading decision mechanism and an offloading dispatcher designed to balance energy consumption and response time for fog servers serving multiple applications in the IoT. This mechanism employs energy-aware cloud-fog offloading (ECFO), which aids in selecting the optimal target server with minimal utilization from the available servers. To address the issue of distant offloading and its associated consequences, ECFO assesses the cost of offloading decisions concerning bandwidth and energy consumption. This assessment is conducted through an energy-aware module by comparing it with the cost of local server execution. The proposed algorithm is evaluated against two state-of-the-art algorithms, and the results demonstrate that ECFO outperforms the others.

In [15], the authors introduce a privacy-aware LB algorithm that employs reinforcement machine learning techniques to reduce the number of waiting tasks in the queues of fog nodes. The proposed algorithm, DDQN, does not rely on load or resource information from fog servers to determine the optimal server for offloading. Instead, it leverages Markov theory to estimate the availability of free servers. This approach significantly enhances system performance while maintaining privacy at an acceptable level. Interactive experiments demonstrate that DDQN outperforms a search-based optimization algorithm from the literature and traditional baseline approaches.

Albalawi, Alkayal et al. [2] introduced a hybrid LB algorithm called PSOSVR, which combines particle swarm optimization (PSO) with support vector regression (SVR). PSOSVR reduces response time and energy consumption while improving resource utilization (RU) and throughput. The outcomes of this proposed algorithm notably enhance various metrics, with energy consumption improving by 56%. Lu, Gu et al. [36] tackled the offloading problem in large-scale systems and multiple service clusters. Their paper compares average execution time, latency, load balancing, and energy consumption, demonstrating that the IDRQN algorithm outperforms others. Tran-Dang and Kim [56] proposed a dynamic collaborative task offloading (DCTO) algorithm to reduce execution time delays in fog systems. The algorithm has two main components: a task division technique and parallel execution. It seeks to identify the optimal target server for offloading among the servers in four layers. However, the algorithm does not prioritize sensitive applications over others.

In [19], a dynamic energy resource allotment (DERA) technique that combines oppositional sparrow search (OSS) with the gravitational search algorithm (GSA) is introduced. DERA aims to improve energy efficiency and overall computing cost in FC environments, focusing on LB by reducing broadband costs, duration, and energy consumption. The proposed algorithm includes four layers: terminal servers, FC, cloud computing, and applications. The fog layer's controller module coordinates these layers. The DERA algorithm outperforms the DRAM algorithm by 6.96 percent in resource management through LB in most experiments. However, DERA does not prioritize sensitive applications and follows a centralized approach, which may limit flexibility and reliability.

Hussein and Mousa [22] introduced two task-offloading algorithms using nature-inspired meta-heuristic schedulers: ant colony optimization (ACO) and particle swarm optimization (PSO). They aim to minimize task response times while considering network latency, bandwidth, and fog server loads. Comparing these algorithms with the round-robin (RR) approach in extensive experiments, the ACO-based scheduler notably improves IoT task execution times. This ACO algorithm considers completion deadlines and optimizes fog server efficiency by finding the shortest path between the source and resources. However, it maintains some aspects of traditional offloading methods, relying on a central server for decision-making and processing time determination.

Lu, Wu et al. [38] proposed a resource provisioning strategy to reduce the total mandatory cost in time-sensitive applications. The authors conducted a study in unlimited-processor and limited-processor fog nodes. Their paper introduces a heuristic algorithm that delivers exceptional performance in enhancing resource provisioning, even under challenging conditions. Li, Zhuang et al. [31] introduced a Self-Similarity-based Load Balancing (SSLB) algorithm for large-scale FC systems. The authors introduced the concept of the 'cell,' which is sized to address distance offloading issues. While SSLB exhibits excellent performance compared to other algorithms, it does not offer advantages for time-sensitive applications (TSA), which have numerous restrictions. Additionally, the algorithm enforces uniform cell sizes, leading cells to be allocated to servers that may be located at a distance.

The previous Section discussed various LB solutions summarized in Table 1. These solutions primarily aim to mitigate the impacts of dynamic offloading rather than addressing the root cause of the problem. Despite their use of innovative technologies, they often entail hidden costs that can create an inconspicuous burden.

This section deeply studies the behaviour of the related work solution to select suitable vital techniques, such as static offloading, clustering, and decentralized control systems. The proposed solution is based on collecting some of these techniques in a simplified and scientific way to obtain an innovative solution to the research dilemma.

A summary of the current literature review reveals that dynamic offloading has gained widespread acceptance in FC. However, it is beset by inherent limitations, leading to significant consequences. Existing research has predominantly concentrated on improving dynamic offloading performance and catering to time-sensitive applications. Nonetheless, a noticeable gap exists in the realm of integrated solutions that can effectively address the inherent challenges of dynamic offloading, particularly those concerning offloading decisions and distant offloading.

This work aims to bridge these gaps and propel LB capabilities to new heights within the FC environment. Achieving this goal necessitates the development of a novel offloading strategy capable of surmounting these formidable challenges, as will be explained in the next section.

Hybrid approach to enhance load balancing

In this Section, we dive deeper into the complexities of LB for FC and the innovative workload offloading solution we propose to solve the dilemma of the research. Our proposed solution aims to directly address these challenges by providing an efficient offloading strategy that combines algorithms and real-time analytics to make informed task allocation decisions. By optimizing LB at the edge, we aim to optimize resource usage, reduce latency, and provide a smooth and responsive experience for end users and servers.

As mentioned, many challenges and difficulties persist in fog load balancing, including network congestion, distant offloading, inflation of decision time issues, which drove us to create the hybrid offloading solution. The design of this proposed algorithm adheres to the following main guiding principles to address some of the shortcomings observed in prevalent algorithms:

- Decision time: Despite the minimal impact of offloading decisions individually, they occur continuously across affected servers. To mitigate these effects, a novel offloading approach is followed, partly inspired by static offloading principles. The proposed resolution must utilise the static behaviour in selecting process of target server without consume time to pick up the destination server. At the same time, it must avoid falling in the drawbacks of static offloading. The hybrid approach plays a crucial role in minimizing the consequences of offloading decisions.

Table 1 Comparison of related work

Authors	Contributions	Offloading-in	Offloading Approach	Clust	TSA	Control System	Metrics	Pros/Cons
Jiang, Chen et al. [25]	Offloading dispatcher and an energy-efficient offloading decision mechanism	Fog nodes	Find the optimum destination server after studying the offloading cost-Dynamic	No	Yes	Decentralized	Response time, Energy consumption	The algorithm costs the system a high number of exchanged messages to explore the suitable server for offloading
Ebrahim and Hafid [15]	An LB algorithm based on Reinforcement Learning (RL), DDQN	Cloud and Fog Servers	Intelligently distributing workloads minimizes waiting delays for IoT workloads in dynamic environments with unpredictable traffic demands	Yes	No	Distributed	Latency, Waiting time, Execution time, Response time	The author does not solve the inherent issues of traditional offloading
Albalawi, Alkayal et al. [2]	PSOSVR: based on a many-objective Particle Swarm Optimization (PSO) algorithm with Support Vector Regression (SVR)	Fog nodes	Dynamic- AI	No	No	Centralized- The control unit is in the Fog layer – FSM	Response time, Energy consumption, Resource utilization, and Throughput	The authors have not given an advantage in execution for time-sensitive applications or distance offloading. The architecture has low scalability
Lu, Gu et al. [36]	DRL is based on the improved IDRQN algorithm	Cloud and Fog Servers	DRL is proposed for offloading in large-scale heterogeneous MEC with multiple service nodes and task dependencies	Yes	Yes	Decentralized	Energy consumption, load balancing, latency, and average execution	The study explores TSA and distance offloading but employs a traditional approach that may result in unnoticed delays
Tran-Dang and Kim [57]	Proposing DCTO, a dynamic collaborative task offloading algorithm	Fog nodes	Using partitioned tasks and parallel computation	No	No	Decentralized	Average of task execution delay and utilization ratio of fogs	The authors did not give an advantage for sensitive applications or distance offloading
Gowri and Baranidharan [19]	A dynamic energy resource allotment (DERA) technique	Cloud and Fog Servers	The proposed algorithm used two algorithms to find the optimum target server	No	No	Centralized- The control unit is located in the Fog layer—Controller	Broadband costs, duration, and energy consumption	The algorithm shows outstanding results, among others. However, it ignores sensitive applications and distance offloading
Hussein and Mousa [22]	Two nature-inspired meta-heuristic schedulers, namely ant colony optimization (ACO) and particle swarm optimization (PSO),	Cloud and Fog Servers	The proposed algorithm considers the network latency and the service rate of the fog nodes	No	No	Centralized- The control unit is located in the Fog layer –Fog Controller Node	Communication cost and Response time	The authors do not prioritize the critical application to be executed locally. Moreover, they ignore other critical problems

Table 1 (continued)

Authors	Contributions	Offloading-in	Offloading Approach	Clust	TSA	Control System	Metrics	Pros/Cons
Li, Zhuang et al. [31]	A self-similarity-based load balancing (SSLB) mechanism for large-scale FC	Fog nodes	address the LB challenges caused by fog's 'large-scale' characteristic through clustering	Yes	Yes	Decentralized	Execution time, clustering overhead	Even though SSLB presents many features, it faces the inherent issues of prevalent offloading, such as decision time and messages
Lu, Wu et al. [38]	Two scenarios are considered: Unlimited processor fog nodes (UPFN) and limited processor fog nodes (LPFN)	Cloud and Fog Servers	minimize the total monetary cost by considering the deadline and capacity constraints	No	Yes	fog nodes' distributions are concentrated	Average cost and Makespan	The manuscript does not find a solution for large-scale networks and other phenomena in dynamic offloading
Sarma, Kumar et al. [49]	A smart gateway as a load balancer in a fog environment	Cloud and Fog Servers	The authors proposed a smart gateway that controls the arrival tasks with minimum cost	No	No	Centralized-—The control unit is located in the Fog layer –Smart gateway	Network delay and Computing time	Despite the outcomes of the centralized system, the proposed solution did not consider other inherent issues of offloading costs
Chakraborty and Mazumdar [9]	Hybrid metaheuristic Greedy Randomized Adaptive Search Procedure and Genetic Algorithm (GRASP-GA)	Cloud and Fog Servers	Hybrid metaheuristic and capacity-based Dynamic-AI for dynamic edge server selection in task offloading	Yes	Yes	Centralized. The control unit is located in the cloud layer	Total execution time and energy consumption	The authors did not study the impact of distance offloading on critical applications. Assigning sensitive applications to fog servers is a better approach despite the impressive results of the proposed algorithms
HybOff (The proposed algorithm)	Real low-cost hybrid offloading approach	Fog Layer	Each heavy device forward the exceeded tasks to its complement device in the cell	Yes	Yes	Decentralized. The central control unit is in fog layer	Utilization of central processing unit	Advantages: It seeks to gain performance from static and dynamic offloading Disadvantage: it cannot work effectively within loaded cells

Moreover, since the offloading process has time and network traffic costs, the proposed algorithm compels fog servers to serve locally as much as possible, meeting the requirements of time-sensitive applications. Thus, following the static offloading approach and local execution (at the leaf server) are the keys to enhancing the system's decision time.

- Distance offloading: While increasing the number of fog servers enhances system availability, it can negatively affect dynamic offloading. The proposed algorithm mitigates distant offloading issues by grouping distributed servers into sets of cells. The clustering concept ensures that all adjacent servers interact with each other. Additionally, forcing leaf servers to execute sensitive applications to be performed locally decreases the number of offloaded tasks across the network.
- Network congestion: As we have explained the reason for and consequences of network congestion, HybOff's strategy will be depicted here. To do that, the HybOff model will utilize static behaviour again with the clustering technique. Both techniques will reduce the number of exchanged messages across the network.
- Flexibility: Given the decentralized behaviour of fog servers, where servers can randomly connect to or leave the fog environment, it is crucial to design a flexible mechanism that instantly reflects the status of connected and reconnected servers. Flexibility is enhanced by identifying a central server in each cluster that tracks clustered servers as they join or leave. Much research uses this feature. However, no research utilizes all of these features in one work.

This work introduces the Hybrid Offloading (HybOff) algorithm, which aims to enhance LB efficiency and resource utilization in fog networks. The development of this hybrid offloading approach was motivated by the persistent challenges and difficulties outlined in the problem statement. Dynamic offloading mitigates these issues but has drawbacks: network congestion, high decision latency, and inefficiency with increased servers and distant offloading. These challenges are critical for time-sensitive applications like healthcare. HybOff addresses these issues to provide adequate load balancing. Even though the techniques used existed before, up to our knowledge, studies have yet to propose this resolution as we used. Figure 3 illustrates the estimated costs associated with prevalent offloading and the essential features it provides.

Problem formulation and terminology

At the outset, Table 2 presents the essential notations used in this work to facilitate the reading.

LB in fog networks demands innovative task allocation for end-user service requests, which are transformed into applications, underscoring the need for efficient task management [39]. In this context, this work adopts the

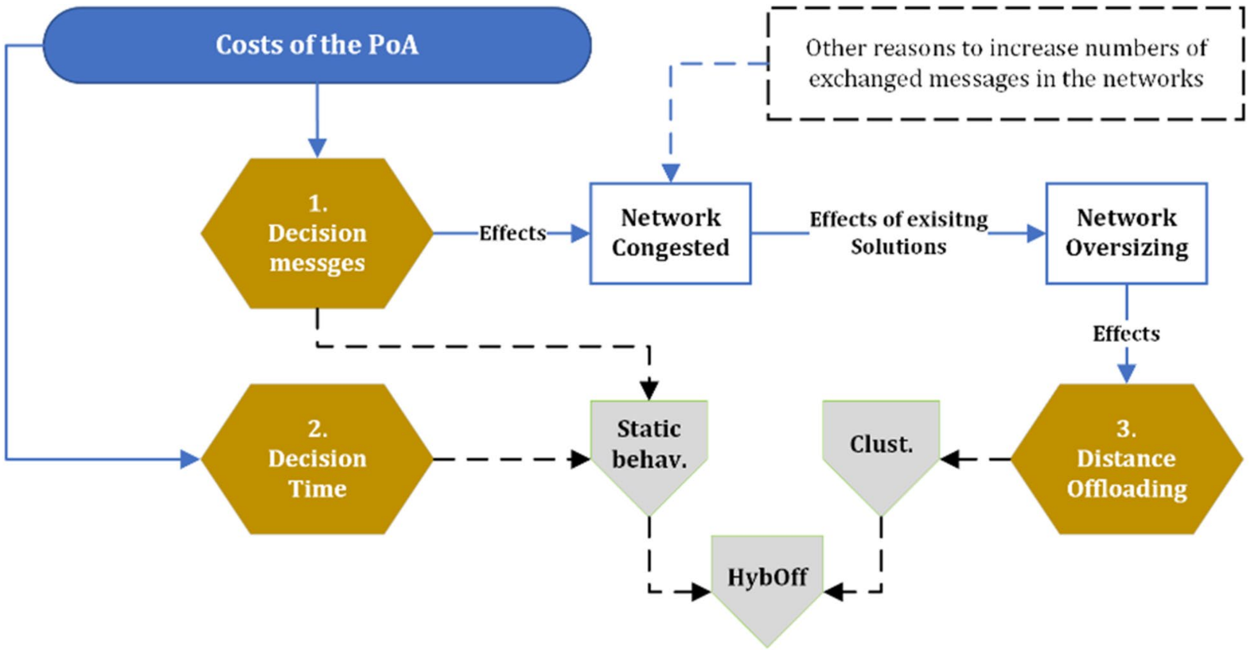


Fig. 3 Prevalent offloading, costs, and solutions

Table 2 Essential notations

Symbol	Definition
TSA	**Time-sensitive application:** Refers to applications with strict time constraints, where processing and response times are critical
HA	**Heavy application:** Denotes applications that require a significant number of computational resources and are resource intensive
CPD	**Cooperating paired servers:** Represents servers that work in tandem or cooperation, often used for LB or redundancy
SOT	**The Static Offloading Table** is a data structure or table that contains information about how tasks are offloaded from one server to another in a static manner
W	**A complete set of system fog servers:** Refers to the entire collection of fog servers in the system, which collectively provide computing resources
n	**Number of system applications:** Represents the count of applications within the system
N	**Number of cells constituted after clustering:** Indicates the total number of cells formed after applying a clustering algorithm or process
Q_k	**The queue of thek^{th} fog server:** Denotes the queue or waiting line for tasks that need to be processed by the k^{th} fog server
F_{ri}	**Thei^{th} fog server:** Refers to the specific or i^{th} fog server in the system
$Fn_i^{CoD_x}$	**The complementary server of x^{th} cell for Fn_i:** Denotes the server in cell x that complements or cooperates with the i^{th} fog server Fn_i
$Fn_i^{RU\%}$	**The utilization percentage of thei^{th} fog server:** Represents the percentage of computational resources used by the i^{th} fog server Fn_i
$Fn_i^{M_x}$	**i^{th} fog server, which acts as a master of cell x:** Refers to the i^{th} fog server that serves as the primary or controlling server for cell x
Cel_i^{Sz}	**The number of servers ini^{th} cell:** Indicates the count of servers present within the i^{th} cell
$Cel_i^{RU\%}$	**The average utilization of thei^{th} cell:** Denotes the percentage of resources used, on average, within the i^{th} cell
App_i	**Thei^{th} application:** Refers to a specific application, often in the context of multiple applications running within the system
$AppTsk_{xy}^{z}$	**The y^{th} task of application x computed in Fn_x:** Describes the task y within application x that is processed by the server Fn_x
μ	**The theoretical difference between each consecutive server in SOT**: Represents the calculated or theoretical variance or difference between consecutive servers listed in the Static Offloading Table (SOT)
$\overline{Sys^{RU\%}}$	**The average system resource utilization:** Denotes the mean or average utilization of resources within the system
$\overline{Sys^{LB}}$	**The average load balance of the cells of the system:** Represents the average distribution of computational load among the cells in the system

assumption that the fog layer consists of a single level of fog servers with no vertical dimension, utilizing only the horizontal dimension for offloading. Additionally, we consider the fog layer to comprise *W* fog servers, denoted as Fn_1 *to* $Fn_{W,}$ alongside *n* applications represented as cloud services.

$$App = \{App_1, App_{2,\ldots,} App_n\} \tag{1}$$

In this work, we categorize applications based on their task partitioning. Light applications that are not partitioned into multiple tasks classified as lightweight or time-sensitive applications (TSA) with a restrict deadline requirements. On the other hand, applications with multiple tasks are categorized as Heavy applications (HA). Each HA, upon reception by the fog server, can be subdivided into a group of subtasks, as shown below:

$$App_i = \left\{AppTsk_{i1}^{x}, AppTsk_{i2}^{y}, \ldots, AppTsk_{iq}^{z}\right\} \tag{2}$$

Each HA is divided into subtasks, such as $AppTsk_{i1}^{x}$ (the 1st subtask in the application i assigned to Fn_x), $AppTsk_{i2}^{y}$ (the 2nd subtask in the application i assigned to Fn_y), and so on, with AppTsk_{iq}^{z} representing the q^{th} subtask assigned to Fn_z. However, although there is a possibility to process the $(n+1)^{th}$ subtask before n^{th} subtask, the system cannot accomplish the whole task, application, unless receive the complete subtasks in the origin server.

HAs are distributed to different servers for parallel processing once the partitioning process is completed. In contrast, TSAs are executed locally and receive the highest priority in the server's private queue (Q), which is used to sort and re-sort received tasks.

$$Q = Q_1, Q_{2,\ldots,} Q_W \tag{3}$$

In PoA, subtasks are generally queued on the system's servers when the server's computing power is insufficient to handle them immediately. For example, subtasks from App_i are organized as follows:

$$App_i = \left\{AppTsk_{i1}^{5}, AppTsk_{i2}^{4}, AppTsk_{i3}^{7}\right\} \tag{4}$$

It is important to note that application i is concurrently served by Fn_4, Fn_5, and Fn_7. In contrast, HybOff is designed to accept application subsets from a single server, reducing the load on network bandwidth. For instance, Fn_6 maintains tasks in its private queue, and it cooperates with Fn_9, as shown in Eq. (5):

$$Q_6 = \left\{AppTsk^6_{ax}, AppTsk^9_{bx}, AppTsk^9_{bz}, AppTsk^6_{ay}\right\} \tag{5}$$

This proposed algorithm describes the workload as a task or amount of work performed by a system, application, or service during a specific period. Equation (6) shows the total time consumed (TET) for the workloads in the queue.

$$TET_{Q_i} = \sum\nolimits_{x \in Q_i} ET_{AppTsk^i_x} \tag{6}$$

where, $ET_{AppTsk^i_x}$ represents the execution time in milliseconds (ms) per subtask, with *'x'* denoting its index. To improve the *TET*, a set of tasks ($AppTsk^i_x$) must be managed in each server queue, where the *ET* cannot be enhanced in this study (it is assumed to be fixed). Therefore, the LB issue can be addressed by efficiently redirecting the workload within each server's queue, as described in Eq. (5).

Except for TSAs, this study employs a fixed-price algorithm for evaluating the available servers [31, 32, 53, 54]. Consequently, all servers have identical offloading costs. HybOff prefers to select a target server with sufficient resources, necessitating an evaluation process. Due to the homogeneity in server specifications, a suitable metric is utilized to identify the most appropriate servers for offloading. In the case of HybOff, each server's resource utilization percentage (*RU%*) acts as an indicator to assess its available capacity, as computed in [21]. This metric depends on the computing power required to execute offloaded and local tasks. It is worth noting that HybOff does not factor in offloading costs in its calculations, as time-sensitive applications are executed locally [34, 35].

HybOff design

In essence, control systems in multi-processing environments come in two forms: central and distributed. Central control, a traditional algorithm, suffers from reliability issues, as system failure can occur if the primary controller malfunctions [51]. Consequently, recent research favours distributed systems, where each computing unit functions independently. However, distributed systems lack certain central system advantages, like centralized server selection based on a comprehensive system analysis [48].

This work adopts a central-distributed control system as the optimal solution to combine central and distributed control aspects. It achieves this by segmenting the extensive system into autonomous mini-controlled systems, forming the HybOff algorithm. This algorithm comprises interconnected computing cells, each housing a cluster of adjacent fog servers governed by an elected fog server known as the master fog server (Fn^M). Conversely, the other cell servers are referred to as followers. This design empowers Fn^M to monitor and supervise the performance of the followers, enhancing system flexibility. Even if a cell loses connection with others, each maintains an autonomous control system [61]. The interconnection of these cells forms the central-distributed control system, a framework that facilitates the implementation of HybOff, which requires multi-cells with distributed control.

In implementing the autonomous control system, each fog server is equipped with three modules: HybOffMonitor, HybOffComm, and HybOffSched. These modules handle monitoring, communication, and offloading, creating an independent control system for the fog servers, as depicted in Fig. 4. Table 3 details that fog servers operate in two modes: basic and advanced. The advanced mode is activated in the master server, while the followers remain in the primary mode.

In the basic mode, followers continuously use their monitor module to assess their workload and report it to the master server. The master server's scheduler module processes the data collected by the monitor module, determining the offloading policy needed for task allocation. The offloading process commences once the necessary information is disseminated within the cell via the communicator module. The communicator module is responsible for facilitating communication and message exchange among servers within the cell. The communicator module's thread is periodically generated to ensure all servers receive the necessary information. Additionally, it uses heartbeat information to address churn issues that may arise due to server crashes or new servers joining the network [10].

In summary, each master server collects workload data from the followers, processes it centrally, and then broadcasts the required offloading information to the cell servers to initiate static offloading.

Hybrid framework

The HybOff algorithm's structure comprises a network of interconnected, distributed, and autonomously managed fog servers referred to as cells. To initiate and operate the proposed algorithm, several steps must be performed:

1. ***Clustering***: The concept of HybOff draws inspiration from the self-similarity load balancing w(SSLB) structure, which forms segments (cells) of distributed fog servers with an equal number of fog servers [31, 32]. Unlike SSLB, HybOff does not impose any restrictions on the similarity of cells; instead, the

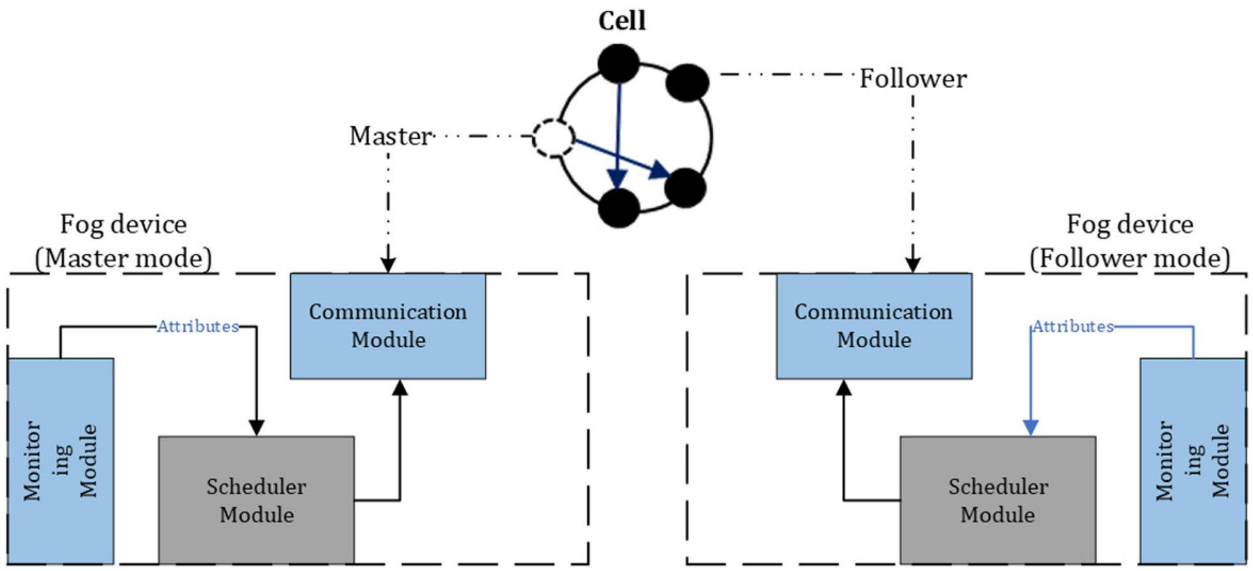


Fig. 4 Architecture of HybOff algorithms. It comprises three essential components: HybOffMonitor, HybOffComm, and HybOffSched, consistently maintained across all fog servers [31, 32]

Table 3 Features of HybOff modules

Module	Basic (Followers)	Advanced (Master)
Monitoring Module	Reporting the utilization percentage periodically	Reading the utilization percentage of the followers periodically
Comm. Module	Acknowledging and updating the target server for offloading process	Maintaining static offloading table policy updating to create the list of targeted servers
Sched. Module	Exchanging the server messages across the cell	It works as a gateway to block internal messages within the cell and handle the outboard messages

clustering algorithm selects cell members regardless of their size, which mean there is no restrictions on cell size. This behavior gives a space to HybOff to build a cell just with adjacent servers. As depicted in Fig. 5, this approach ensures that cells are constructed using adjacent servers. However, significant benefits, such as reduced energy consumption and increased bandwidth, can be achieved if we confine adjacent servers in the same cluster to form cells, thereby minimizing communication with remote servers [15, 50]. Additionally, Li, Zhuang et al. [31] suggest that servers within a close geographical area tend to exhibit similar behaviour, such as server joins or crashes [24]. Therefore, initiating a federation of computing systems is crucial [8]. We employ the simple K-means algorithm described in [33] to build distributed cells, an algorithm known for its exceptional performance in large-scale environments.

The design defines computing servers as the set W, comprising m points in Euclidean space. The objective is to partition the W servers into N sets referred to as cells (Cel_1, Cel_2, Cel_3, ..., Cel_N), each having a master. The variable cell size enables the K-means algorithm to discover the optimal server clustering. The size of any cell is defined as:

$$2 \leq Cel_i^{Sz} \leq W \,|\, \forall i \in N \tag{7}$$

where, Cel_i^{Sz} represents the number of fog servers in the i^{th} cell, which can be either odd or even. For instance, in Fig. 5, the system consists of 19 fog servers *(W=19)* as per the clustering algorithm, they are organized into four cells *(N=4)*. Each cell accommodates a different number of fog servers according to the position of servers, as determined by Eq. (7). By the end of this step, considered a core of the HybOff principle, the model can walk into the next steps.

2. ***Master server:*** In each initialized cell, a controller server is randomly elected to oversee cell activities [48]. It is performed each time the master server is missing. The Fn^M assumes various responsibilities, including:

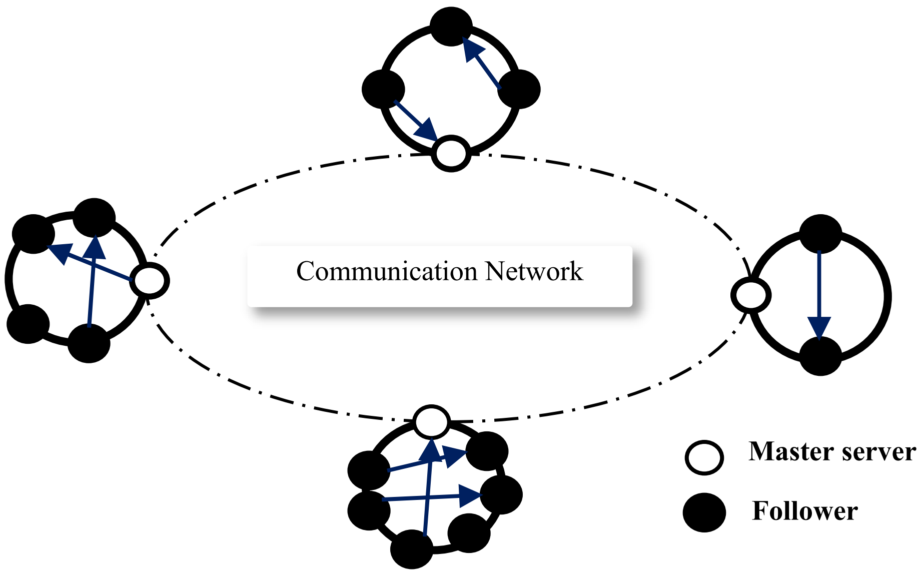


Fig. 5 HybOff structure. In this centrally distributed architecture, contiguous servers are grouped as a cell. Each cell's servers interact with one another while choosing a master for external communication and establishing a SOT

A. Collecting RU information from cell servers, including its data.
B. Updating the offloading table in the scheduler module.
C. Periodically broadcasting the required offloading information within the cell.
D. Monitoring cell servers to exclude any deactivated servers from the offloading process.
E. Serving as a gateway, connecting followers with external systems, and keeping exchanged cell messages confined within the cell, thereby preserving system bandwidth.

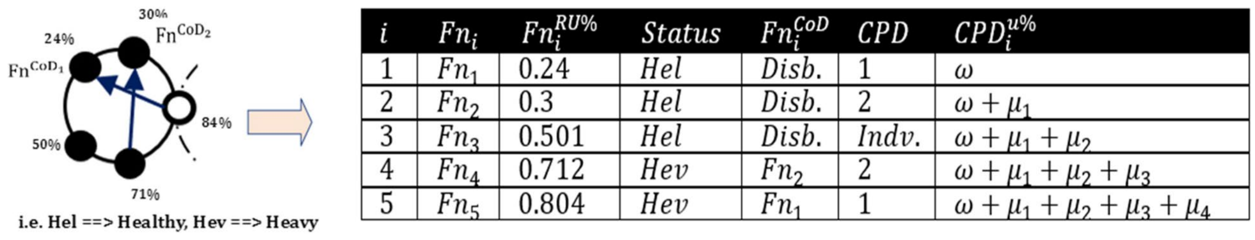
For example, if the master server in a specific cell fails, followers will lose external connections, such as those with the cloud and offloading functions. Nevertheless, followers can continue to perform essential computing functions until another master is selected.

3. ***SOT Policy and CPDs:*** In addition to the HybOff architecture, the SOT policy plays a pivotal role in its design. A static table is inadequate for a system that requires flexibility. Therefore, SOT is a dynamic template within the master's scheduler module. It is unnecessary to offload all fog servers in the cell; instead, SOT contains crucial cell data, including $Fn^{RU\%}$, the target offloading server, and fog identification. The latter is a unique number connecting to each server's Internet Protocol (IP) address as a reference number. Each fog server corresponds to an individual row in the SOT.

Algorithm 1. Building and maintaining SOT in the master server

```
Output: $Fn_i^{CoD_x}$
Input: $\forall\ Fn \in Cel_a, Cel^{RU\%}, Cel_a^{Sz}$
Start;
mid = $approx\left(\frac{Cel_a^{Sz}}{2}\right)$;
Do
    If (OddSize($Cel_a^{Sz}$)
        disabOff($Fn_{mid}$)                          //Disable the offloading function
    for (i = 1: mid; i++)
        $CPD_i = \left(Fn_i, Fn_{Cel_a^{Sz}-i+1}\right)$          //Creating a CPDs
        BroadCast($Fn_{Cel_a^{Sz}-i+1}^{CoD_i}$)              // Update CPDs
      End
  End
```

Once the necessary data is available, SOT ranks active fog servers in ascending order based on their resource utilization. This approach follows an ascending pattern, placing heavy servers at the end of the table and lighter servers at the top. After sorting the cell servers, SOT creates cooperating paired servers (CPDs). A CPD consists of two fog servers within the same cell with opposite resource utilization readings. The first server has the highest reading, while the second, known as a complementary server (CoD), has the lowest reading. This pairing is illustrated in Fig. 6, where $Sys_{avg}^{RU\%} = 54\%$, CPDs are formed by pairing opposite servers using Algorithm 1. Equation (8) specifies the servers participating in each pair, with $Cel^{sz} - i + 1$ representing the index for the fog node paired with fog node *i:*



i	Fn_i	$Fn_i^{RU\%}$	$Status$	Fn_i^{CoD}	CPD	$CPD_i^{u\%}$
1	Fn_1	0.24	Hel	$Disb.$	1	ω
2	Fn_2	0.3	Hel	$Disb.$	2	$\omega+\mu_1$
3	Fn_3	0.501	Hel	$Disb.$	$Indv.$	$\omega+\mu_1+\mu_2$
4	Fn_4	0.712	Hev	Fn_2	2	$\omega+\mu_1+\mu_2+\mu_3$
5	Fn_5	0.804	Hev	Fn_1	1	$\omega+\mu_1+\mu_2+\mu_3+\mu_4$

Fig. 6 Static offloading table in the master server

$$CPD_i = [Fn_i, Fn_j], where\ j = Cel^{Sz} - i + 1 \tag{8}$$

4. ***Broadcasting:*** After the creation of CPDs, the master server broadcasts complementary server information throughout the cell. The "*broadcast( )*" function informs the heavy cell servers about their complementary servers. In contrast, the middle server (in the SOT when N is odd) disables the offloading function to operate independently without participating in the offloading process, achieved through "*disabOff( )*". Additionally, light followers must also disable the offloading function to prevent the system from entering a thrashing state. In a thrashing state, all servers spend time forwarding tasks among themselves without executing their primary functions [27]. The HybOff algorithm avoids this state by employing the "*disabOff ( )*" function, which restricts specific and unnecessary servers from forwarding tasks. However, the function only prevents servers from offloading within the cell, allowing them to continue offloading outside the cell or to the cloud when necessary.
5. ***Static offloading:*** Heavy servers initiate offloading as soon as they receive information about their complementary servers (Algorithm 2). They forward heavy tasks using the Last In, Last Out (LIFO) procedure, with priority given to all TSAs in their queue. Servers continue to utilize their complementary servers until they receive updated information from SOT.

Algorithm 2. Static offloading (all servers)

```
Output: Provide destination server
Input: $Cel^{RU\%}$, $Fn^{RU\%}$
Start;
Do ($Fn_a^{RU\%} \geq Cel^{RU\%}$)
    $Fn_b^{CoD} \leftarrow Offload(Fn_a, HA(AppTsk_{xy}^a)$        //Static Offloading
End
```

HybOff requires verification that heavy servers surpass the average load of the cell. In this algorithm, offloading occurs independently within each cell once a server is categorized as heavy. To establish the appropriate categorization criteria for servers, the average utilization ratio of each cell must first be calculated. Equation (9) provides the formula for categorizing each cell:

$$Cel_i^{RU\%} = \frac{\sum_{x=1}^{Cel_i^{Sz}} (Fn_x^{RU\%} \times Fn_x^{\varphi})}{(Cel_i^{Sz} - \sum Fn^{\varphi})} \tag{9}$$

where, $Cel_i^{RU\%}$ represents the average utilization ratio for cell i, and Fn_φ is 1 if the fog server is active and 0 otherwise. The cell servers will not initiate offloading until the categorization criteria are met. In this algorithm, if $Fn_i^{RU\%} \geq Cel^{RU\%}$, Fn_i is considered a heavy server; otherwise, it is categorized as a healthy server. This condition deactivates the algorithm when all servers are not overloaded. For example, if all cell servers have a low load, no offloading process will commence, and each server will manage its workload locally. Thus, we can define this cell as a balanced cell, a feature that significantly benefits network bandwidth.

Let us consider an illustrative example to comprehend the relationships among cell servers. In previous Fig. 5, if the clustering algorithm forms a cell with five fog servers, the first server, after ranking in the SOT, has a utilization percentage of $Fn_1^{RU\%} = \omega$. It is important to note that there are variations in the utilization percentages among the sequentially ranked servers, denoted as μ_1, μ_2, μ_3, and μ_4 in our calculations. In this example, we have two $CPDs$, CPD_1 and CPD_2, each with a unique utilization reading. However, to calculate the RU for the i^{th} pair, we need to apply the following relationship:

$$CPD_i^{RU\%}(i, Cel_i^{Sz}) = 2\omega + \sum_{a=1}^{i-1} \mu_a + \sum_{b=1}^{Cel_i^{Sz}-i} \mu_b \tag{10}$$

where $\mathrm{CPD}_i^{\mathrm{RU\%}}$ represents the utilization percentage of CPD_i in the cell. Using Eq. (10), CPD_1 contains Fn_1 and Fn_5, while CPD_2 contains Fn_2 and Fn_4. When the load reaches the average cell load, Fn_4 and Fn_5 will offload their workloads to Fn_1 and Fn_2, respectively. Fn_3 operates independently as it has an adequate load. In cases where the number of cell servers is even, all servers are included in computing pairs. The utilization percentage in each pair is as follows:

$$CPD_1^{RU\%} = 2\omega + \mu_1 + \mu_2 + \mu_3 + \mu_{4.} \tag{11}$$

$$CPD_2^{RU\%} = 2\omega + \mu_1 + \mu_1 + \mu_2 + \mu_3 \tag{12}$$

Unfortunately, there is no mathematical relation that can predict μ. For simplicity, we assume that the utilization value between each sequential server is constant $\left(\mu_1 = \mu_2 = \cdots = \mu_{(sz-1)} = \mu\right)$. If so, we can conclude that: $CPD_1^{RU\%} = CPD_2^{RU\%} = 2\omega + 4\mu$, which represents the utilization percentage for any *CPD* in the previous example. In other words, HybOff equalizes the loaded pairs cell-wise. This work predicts the RU% for the cooperative pair servers using the following formula:

$$CPD^{RU\%}\left(Cel_i^{Sz}\right) = 2\omega + \left(Cel_i^{Sz} - 1\right)\mu \tag{13}$$

Mathematically, all CPDs in the cell have the same load. However, the load of CPDs depends on the number of fog servers in the cell. For example, after the clustering algorithm builds the cells, Cel_aandCel_b contain 6 and 13 servers, respectively. According to Eq. (13), $CPD_a^{RU\%} = 2\omega + 5\mu$, and $CPD_b^{RU\%} = 2\omega + 12\mu$. This means that the amount of shared computational load for each *CPD* increases with the cell size.

The proposed algorithm

The identified drawbacks will be effectively addressed through the integration of the cell concept within our hybrid offloading framework. In this design, Fog servers are structured into cells, where each server pairs up for resource sharing. Our proposed algorithm is aimed at maintaining consistent average load levels across Fog servers within each cell, and you can visualize the algorithm's flowchart in Fig. 7.

As depicted in the figure, this hybrid LB algorithm capitalizes on the strengths of static and dynamic offloading strategies. Our proposed algorithm brings five crucial enhancements to fog load balancing:

- Reintroduction of static offloading: We are reintegrating the efficiency of static offloading into our approach.
- Minimal message exchanges: Our algorithm minimizes message exchange between servers, streamlining the LB process for greater efficiency.

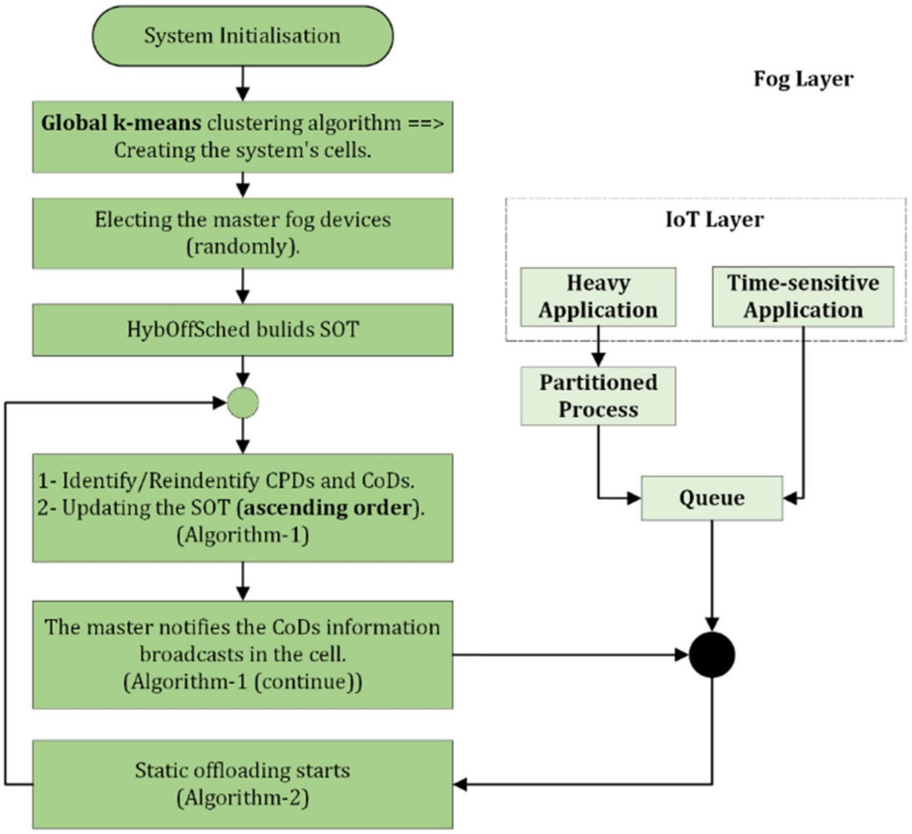

System Initialisation
Global k-means clustering algorithm ==> Creating the system's cells.
Electing the master fog devices (randomly).
HybOffSched bulids SOT
1- Identify/Reindentify CPDs and CoDs.
2- Updating the SOT (ascending order).
(Algorithm-1)
The master notifies the CoDs information broadcasts in the cell.
(Algorithm-1 (continue))
Static offloading starts
(Algorithm-2)
Fog Layer
IoT Layer
Heavy Application
Time-sensitive Application
Partitioned Process
Queue

Fig. 7 HybOff process flowchart

- Reduced decision-making time: We have significantly reduced the time required to make offloading decisions.
- Local management of urgent applications: Our approach encourages servers to handle urgent applications locally, eliminating the necessity for global allocation.
- Efficient network management: We employ a cell-based approach for network management, reducing latency, alleviating network congestion, and enhancing overall load balancing.

The subsequent Section illustrates these improvements through a series of comprehensive experiments.

Experiments and results

Preliminary explanations

This Section assesses the proposed algorithm and demonstrates how the hybrid offloading structure outperforms other classical LB schemes. Generally, the essential requirement of an effective LB is to keep all the computing units equally loaded by avoiding overloaded or underloaded cases [51]. The *RU%* of servers are used to evaluate the effectiveness of LB.

To demonstrate the efficiency of HybOff, we consider the example depicted in Fig. 8 for a mathematical analysis. Figure 8 (a) shows five fog servers with fixed differences (μ) after clustering in a specific cell. The scheduler module collects the resource utilization for cell servers to rank them in ascending order, as shown in Fig. 8 (b), to facilitate creating the CPDs shown in Fig. 8 (C). These figures illustrate how opposite servers share their load while the middle server operates independently. After a period of offloading, all cell servers have the same load, as shown in Fig. 8 (d). This figure proves that HybOff has the ability (mathematically and in the ideal case when μ is fixed) to balance the usage of the resources of fog servers in the cell by using Eq. (13).

Using Eq. (13), we find that $CPD^{RU\%}$ equals $2\omega+4\mu$ for each pair, where the x for the cell servers is $\omega+2\mu$. Fortunately, the middle server also has the same load of $\omega+2\mu$. HybOff balances load by dividing the cell servers into multiple pairs and ensuring an equal distribution through sharing. HybOff successfully balances the load and creates balanced cells by ensuring that opposite servers share the load.

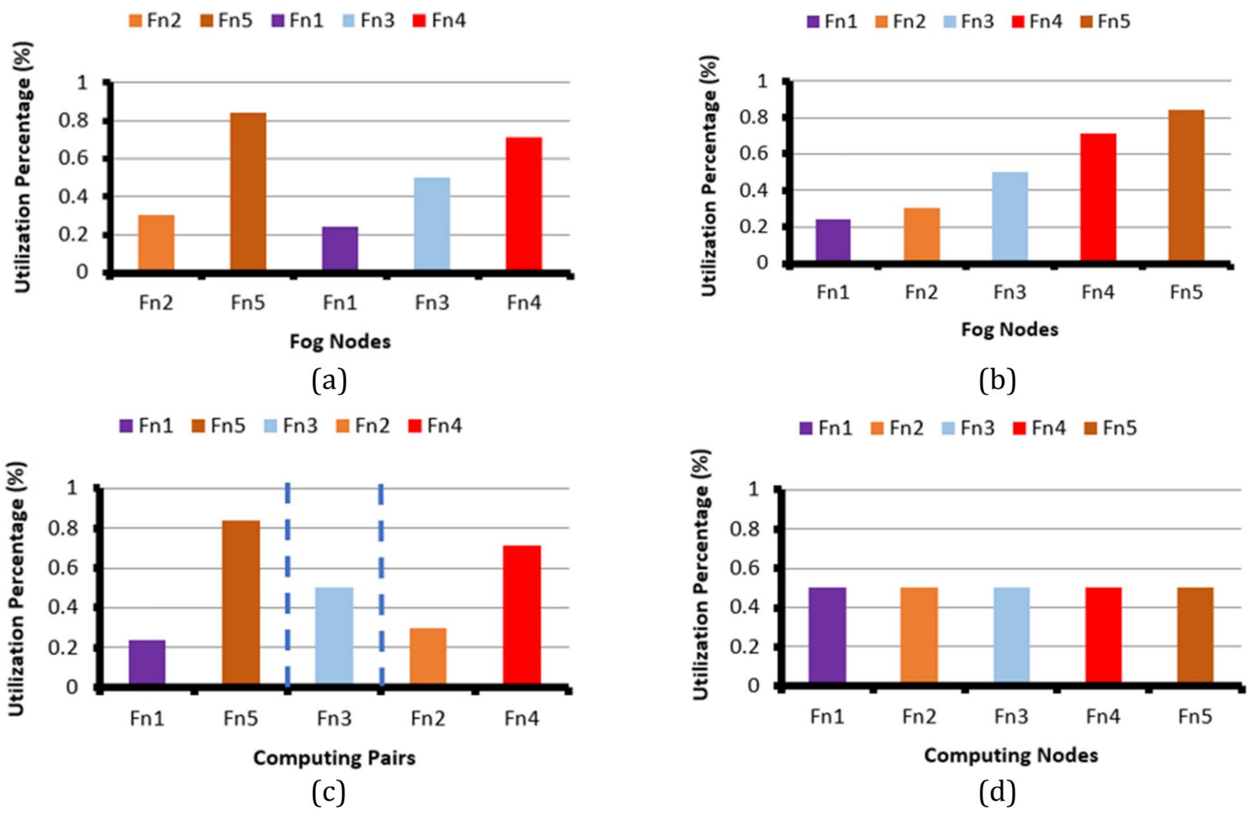


Fig. 8 HybOff model, Balance of resource utilization-mathematical, (**a**) before offloading, (**b**) sorted servers, (**c**) paired servers, and (**d**) after offloading

Environment description

To evaluate the proposed algorithm, three metrics are employed 1) resource utilization ratio of the fog system, 2) loading balancing resource usage among fog servers, and 3) system performance. Resource utilization measures the usage of all the distributed fog servers' computing resources. LB determines the distributed tasks among computing servers in the fog layer. The system performance checks the efficiency of the entire algorithm.

Simulation setup

The experiment follows the algorithm described in Fig. 1, as outlined in the work by Lu, Zheng et al. [38]. It consists of W distributed fog servers and N created cells, which are selected according to the clustering algorithm used in this article. Cloud services, denoted as *'n'*, are available on all fog servers, and offloading is initiated only in cases of computing power shortage. Tasks of varying sizes are processed on the fog servers. The initial experiment settings are summarized in Table 4:

It's important to note that this experiment focuses exclusively on the fog layer and does not consider the cloud. The simulation tool used is iFogSim, which is responsible for creating the necessary environment (Gupta, Vahid Dastjerdi et al. 2017). The experiment assesses various parameters across different server scales and data sizes and examines resource utilization over time, considering random combinations of data sizes and scales, as detailed in Tables 5 and 6.

Fog server specifications

The specifications of the fog servers used in the experiment are provided in Table 5:

Evaluation metrics

To evaluate the algorithm's performance, we measured RU in the described case studies using three different schemes: the static SoA, the PoA, and our proposed HybOff scheme. PoA is adapted from [56, 57], while SoA is configured using classical static offloading. The resource utilization ratio of the system in the experiment is calculated using Eq. (14), where certain function components from the HybOff algorithm were modified and reused to implement SoA and PoA:

Table 4 Initial parameters of experiment

Parameter	*W*	*n*	ω	μ	ET/task
Value	Up to 300	15 apps	18%	7%	3 ms

Table 5 Specifications of fog servers

F_{ri}	Capacity	RAM	CPU
Fn_1	100 MB	7 MB	120 MHz
Fn_2	150 MB	15 MB	80 MHz
⋮	⋮	⋮	⋮
Fn_i	200 MB	10 MB	100 MHz

$$\overline{Sys^{RU\%}} = \sum_{b=1}^{N} \frac{\sum_{a=1}^{Cel_b^{Sz}} Fn_a^{RU\%}}{Cel_b^{Sz}}, if\, Fn_a^{RU\%} \cong \mp 5\%\, Sys_{Avg}^{RU\%} \tag{14}$$

Task specifications

The specifications of the tasks used in the experiment are detailed in Table 6:

Resource utilization

In this work, the resource utilization ratio of the system ($Sys^{RU\%}$) refers to the ratio between the number of resources utilized and the total amount of system resources. The utilized resource is any processor of fog server which consumed more than or equaled the average cell utilization of its processing power. To do this, the RU needs to be calculated at every detection time using Sys^{avgRU}, Eq. (15).

$$Sys^{avgRU} = \sum_{\mathrm{i}=1}^{\mathrm{t}} Sys_i^{RU\%}/t, \tag{15}$$

where t represents the number of detection times during the experimental period. Figure 9 (a) illustrates that the system utilization ratio fluctuation is lower for SoA and PoA. This is primarily because HybOff enforces cooperation among opposite servers, enabling the system to tap into previously unexplored resources and communicate directly with the most affected servers to offload their load. In contrast, SoA and PoA experience inefficiencies in redistributing workload, resulting in a leakage of fully utilized servers. Equation (16) presents the formula used to calculate the system's utilization during the experimental period.

Table 6 Task specifications

Process	Process size	Partitions	Sensitivity	Priority
P1	5 MB	1/1	TSA	High
P21	6 MB	1/3	HA	Low
P22	6 MB	2/3	HA	Low
P23	9 MB	3/3	HA	Low
⋮	⋮	⋮	⋮	⋮

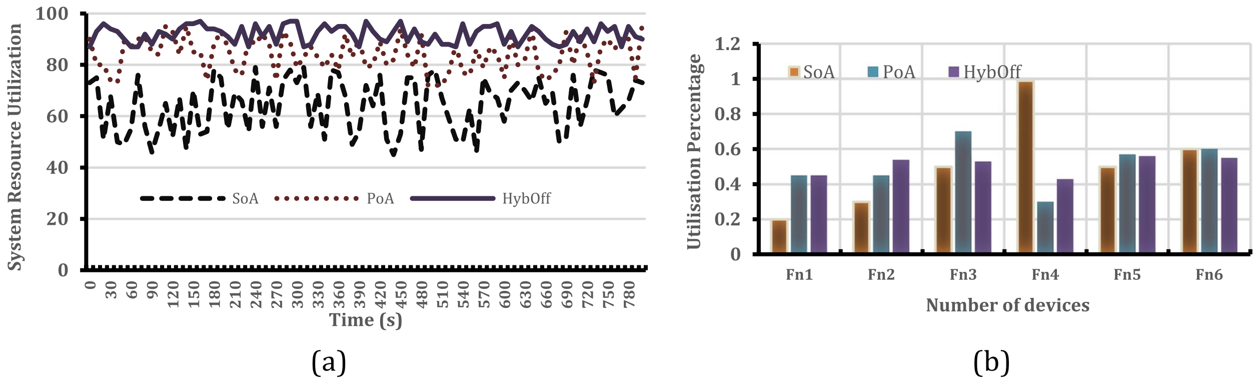


Fig. 9 Resource Utilization over time and changing number of servers. **a** Resource Utilization of the System with Changing of Time, and (**b**) Resource Utilization Percentage

$$Sys^{RU\%} = \frac{\sum_{i=1}^{N} Cel_i^{RU\%}}{N} \quad (16)$$

To evaluate HybOff's efficiency in utilizing available resources at the server level, we need to determine the average RU under fixed fog scales and varying data growth rates. To achieve this, different amounts of data is generated for the same cell scales. The experiment involved continuously increasing data generation and monitoring the capacity of fog servers in the cell. The average RU provides insights into the algorithm's effectiveness in leveraging the available resources. In Fig. 9 (b), the RU of fog servers in a single cell containing six fog servers is depicted. The figure illustrates the captured RU of cell servers at specific times. While all prevalent algorithms achieve approximately 76% utilization of edge resources, HybOff maintains an average of 50% in the cell. This indicates that HybOff evenly distributes the workload among the fog servers, unlike SoA and PoA, which fall short in this aspect. HybOff's advantage stems from the clustering technique, which divides the fog servers into mini fogs. This approach allows HybOff to treat each cell as a mini-fog system, making it easier to manage and control. Additionally, hybrid offloading enhances RU further.

Load balancing

To assess the effectiveness of the proposed algorithm, this Section evaluates the level of balanced RU among servers in the fog layer and compares it to that of SoA and PoA, with the target level defined in [51] where all fog servers were equally loaded. LB is defined as the percentage of healthy fog servers in the cell, with a ± 5% threshold value of ($Sys^{RU\%}$). In this experiment, however, we considered any server close to the average system utilization as a healthy server. To do this, we need to count the healthy server's cell-wise during the experiment, which satisfies the criteria previously mentioned. Eq. (14) is used to calculate the RU for the HybOff algorithm, while Eq. (17) is used to calculate the average RU for the SoA and PoA.

$$\overline{Sys^{RU\%}} = \frac{1}{N}\sum_{a=1}^{N} Fn_a^{RU\%},\ \text{if}\, Fn_a^{RU\%} \cong \mp 5\% Sys_{Avg}^{RU\%} \quad (17)$$

Figure 10 (a) depicts the percentage of fog servers classified as balanced across various system scales, with experiments ranging from 1 to 300 servers, all using a fixed data size. The graph underscores HybOff's ability to consistently maintain a high percentage of healthy servers, closely aligning with the ideal curve. At 150 fog servers, SoA, PoA, and HybOff achieved percentages of 64%, 88%, and 97% for balanced servers, respectively. Impressively, HybOff continued to perform exceptionally well even with 230 fog servers. However, the dynamic scheme's performance deteriorated when the number of fog servers reached 300, revealing communication overhead as a bottleneck.

The performance of the static approach exhibits a decreasing slope, consistent with its strategy. Nevertheless, the results clearly indicate that HybOff excels in large-scale networks, primarily because the network is fragmented, and the central-distributed approach makes it easier to control and maintain. In contrast to the theoretical estimation of HybOff, which suggests effective load equalization among all computing servers, the experimental results do not align with this mathematical estimation. This discrepancy arises from the variable and uncontrolled nature of µ. The uncontrolled differences among consecutive servers diminish the performance of HybOff.

However, standard deviation (σ) serves as a crucial tool for assessing data dispersion. A smaller standard deviation signifies that data points are closely clustered

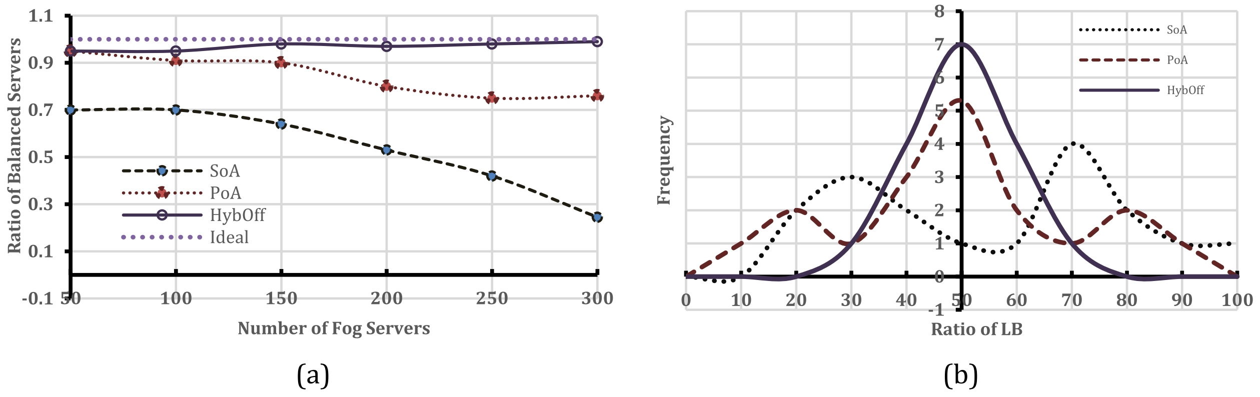


Fig. 10 The percentage of healthy servers for the three algorithms with the same mean value but with different standard deviations, where (**a**) is the percentage of balanced servers, and (**b**) it the standard deviation for algorithms

around the central measure [41]. In this work, all algorithms were tailored to maintain equal load distribution among computing servers. As previously defined, fog servers with computing loads within the 47.5% to 52.5% range are considered healthy. In this experiment, we tallied the number of servers loaded at approximately 50% for each class within each algorithm. Figure 10 (b) demonstrates that HybOff exhibited the lowest standard deviation, while SoA showed the highest. This indicates that HybOff had the most servers meeting the balanced criteria. Although PoA also upheld a substantial number of balanced servers, SoA struggled to keep servers within the target range. The performance results were 39%, 68%, and 95% for SoA, PoA, and HybOff, respectively. All three algorithms had the same mean value, $\overline{x}$=17. SoA, PoA, and HybOff had standard deviations of 20.4, 16.9, and 9.7, respectively. Evidently, HybOff outperformed the other algorithms.

System performance

It is essential to compare the performance of HybOff with SoA and PoA to assess the proposed algorithm's effectiveness. To evaluate each system's performance, we analyzed the execution of time-sensitive applications. As previously mentioned, all servers in the fog layer are tasked with serving time-sensitive applications locally without offloading. For resource-intensive applications, offloading is considered only when the computing servers are deemed fully loaded [58].

Figure 11 (a) depicts system execution time comparisons between HybOff, SoA, and PoA, evaluating their efficiency across various server scales and data sizes. HybOff exhibits notable effectiveness in handling time-sensitive applications and ensuring resource allocation in receiving servers. It excels in resource-intensive tasks by offloading to Complementary Servers (CoDs) without distant offloading, outperforming other algorithms. Figure 11 (b) illustrates the system execution time for the three algorithms with a fixed amount of generated data and an increasing number of fog servers. Initially, with just one fog server handling all the generated tasks, all the algorithms consumed significant time. However, as the number of fog servers increased, each algorithm exhibited a distinct behaviour. While

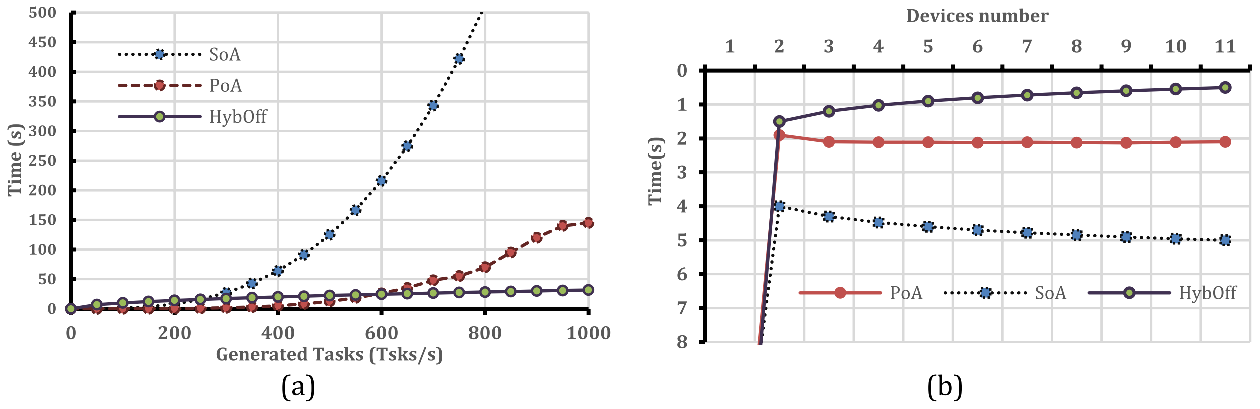


Fig. 11 TSAs' performance evaluation with (**a**) different data sizes and (**b**) different system scales

all solutions showed a declining trend, HybOff consistently outperformed the others. With increasing servers, HybOff's performance led to reduced system execution time. Specifically, HybOff achieved a system performance 1.6 times and 1.4 times better than SoA and PoA, respectively, when operating with 100 servers. This demonstrates HybOff's efficiency in optimizing system performance and resource utilization as fog server numbers increase.

Discussion

This study aimed to enhance our understanding of LB within FC environments by introducing a novel offloading algorithm called HybOff. HybOff was developed to address the inherent challenges associated with existing LB techniques. Our findings demonstrate that adopting a hybrid approach that combines the strengths of both static and dynamic algorithms significantly improves system performance.

Key findings

Our comprehensive analysis of the experimental results, as presented in Figs. 9, 10, and 11, has revealed several key findings:

- Decision time: In a distributed computing system, offloading is the key to elevating the computing load in the overloaded servers. Although offloading is one of the essential tools in this type of computing environment, it has accumulated costs. Figs. 11 and 12 show that the total time consumed to perform the task is low for HybOff, even with increasing tasks or servers. This performance is for the algorithm features that avoid the process having an extra cost. HybOff's hybrid approach allows for direct offloading, effectively eliminating the delay associated with decision-making in the offloading process. This approach contributes to faster and more efficient resource allocation.
- Distance offloading: Unlike other algorithms that struggle with large-scale networks, HybOff excels by avoiding offloading to unknown fog locations, a characteristic more reflective of real-world FC scenarios. Moreover, HybOff is excellent by processing sensitive applications at the received server by giving advantage by avoiding offloading to this type of tasks. However, clustering reduces latency and improves system efficiency.
- Decision messages: HybOff's static behaviour reduces the need for current system state messages, minimizing the exchange of messages among servers and reducing network bandwidth usage. A hybrid approach is crucial for optimizing network performance.
- Superfluous offloading: While other algorithms may experience performance degradation when handling time-sensitive applications (TSAs), HybOff excels by keeping TSAs local, saving transmission time and network resources. It also efficiently manages heavy applications (HAs) by offloading them only to adjacent servers, thus minimizing network congestion.
- Anti-thrashing state: HybOff effectively prevents the system from entering a thrashing state by employing the "disable offloading" function. This ensures that underloaded servers within each area share their resources with the most affected servers, ultimately optimizing system utilization.

These findings align with existing literature that underscores the effectiveness of dynamic offloading as a strategy for LB in FC. However, our study further demonstrates the viability of incorporating classical static offloading into modern network design. These results mark the first direct demonstration of this hybrid approach, offering valuable insights for future research in FC.

Limitations and future directions

Despite the promising findings, this study has identified two potential limitations:

- High-load scenarios: HybOff may not operate efficiently in scenarios with a substantial load within a single cell. When all computing nodes in a cell reach their utilization limits, the "disabOff()" function activates, leading to offloading processes across cells or to the cloud, which may introduce undesired consequences such as network congestion and distant offloading. Future research should explore sustainable solutions for high-load scenarios within a single cell.
- Metric selection: While HybOff uses CPU load and network state as reference metrics to assess fog server loads, it does not consider other server metrics like memory usage and energy consumption. Future investigations could consider a more comprehensive set of metrics for a nuanced assessment.

Implications

These findings have both theoretical and practical implications. Reviving the use of static offloading techniques, previously deemed impractical in modern network design, emerges as a critical consideration. Additionally, adopting approaches like HybOff in industrial computing platforms may help reduce unnecessary network expansion and enhance system performance. In conclusion, HybOff offers a robust and efficient computing environment for fog systems, outperforming prevalent dynamic algorithms and providing valuable theoretical and practical insights for LB in FC scenarios. Future research can build on these insights to address the identified limitations and further advance the field of FC.

Conclusion

This work aimed to enhance the performance of critical applications in large-scale fog networks by introducing a novel algorithm named "HybOff". HybOff represents an LB offloading technique that adeptly harnesses the benefits of both static and dynamic offloading methods, resulting in substantial performance improvements for time-sensitive applications, regardless of network scale. The offloading strategies generated by each algorithm in this investigation were simulated utilizing the iFogSim platform. Through a comparative analysis of diverse metrics encompassing resource utilization, load distribution, and system performance, we discerned the merits and demerits of each approach. The outcomes of these algorithms affirm that, irrespective of network size, HybOff consistently fulfills the requisites of Application Service Dependencies (ASD).

Furthermore, the experimental results strongly corroborate the efficacy of HybOff. It demonstrates a notable reduction in the volume of offloading messages, distance traversed, and the repercussions of offloading decisions. These outcomes effectively mitigate the inherent deficiencies encountered in traditional offloading techniques. Notably, the proposed algorithm enhances LB by an impressive 97%, a substantial improvement compared to the 64% and 88% achieved by SoA and PoA, respectively. Moreover, it elevates the average system utilization rate by 50% and enhances system performance by 1.6 times and 1.4 times compared to SoA and PoA, respectively.

Authors' contributions

H.S wrote the main manuscript, performed experiments, and prepared figures. R.S. assisted in writing and preparing the figures. F.R assisted in the analytic and theoretical validation of the new approach. M.N reviewed the manuscript. H.H Assistance supervised the study. T.J provide general support. M.P fully supervised the whole study. Hamza Sulimani (Corresponding author).

Funding

This declaration is not applicable.

Availability of data and materials

No datasets were generated or analysed during the current study.

Declarations

Ethics approval and consent to participate.

This declaration is not applicable.

Competing interests

The authors declare no competing interests.

Received: 14 December 2023 Accepted: 1 May 2024
Published online: 28 May 2024

References

1. Aazam M, Zeadally S, Harras KA (2018) Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities. Futur Gener Comput Syst 87:278–289
2. Albalawi M, Alkayal E, Barnawi A, Boulares M (2022) Load Balancing Based on Many-objective Particle Swarm Optimization Algorithm with Support Vector Regression in Fog Computing. J Eng Appl Sci Technol. 4:1–10. SRC/JEAST-170. https://doi.org/10.47363/JEAST/2022
3. Alsharif MH, Jahid A, Kelechi AH, Kannadasan R (2023) Green IoT: A review and future research directions. Symmetry 15(3):757
4. Alzoubi YI, Gill A, Mishra A (2022) A systematic review of the purposes of Blockchain and fog computing integration: classification and open issues. J Cloud Comput 11(1):1–36
5. Apat HK, Nayak R, Sahoo B (2023) A comprehensive review on Internet of Things application placement in Fog computing environment. Internet of Things 23:100866
6. Bala B, Behal S (2024) AI techniques for IoT-based DDoS attack detection: Taxonomies, comprehensive review and research challenges. Comp Sci Rev 52:100631
7. Burhan M, Alam H, Arsalan A, Rehman RA, Anwar M, Faheem M, Ashraf MW (2023) A Comprehensive Survey on the Cooperation of Fog Computing Paradigm-based IoT Applications: Layered Architecture, Real-Time Security Issues, and Solutions. IEEE Access 11:73303–73329
8. Cao B, Li Z, Liu X, Lv Z, He H (2023) Mobility-aware multiobjective task offloading for vehicular edge computing in digital twin environment. IEEE J Selected Areas Commun 41(10):3046–3055
9. Chakraborty S, Mazumdar K (2023) A Hybrid GRASP-GA based collaborative task offloading technique in fog computing. Multimedia Tools and Appl 83:119–148
10. Chen B, Hu J, Zhao Y, Ghosh BK (2022) Finite-time velocity-free rendezvous control of multiple AUV systems with intermittent communication. IEEE Transactions on Systems, Man, and Cybernetics: Systems 52(10):6618–6629
11. Dai X, Xiao Z, Jiang H, Lui JC (2023) UAV-assisted task offloading in vehicular edge computing networks. IEEE Trans Mob Comput 23(4):2520–2534
12. Das R, Inuwa MM (2023) A review on fog computing: issues, characteristics, challenges, and potential applications. Telematics and Informatics Reports 10:100049
13. Datta SK, Bonnet C (eds) (2017) An edge computing architecture integrating virtual IoT devices. 2017 IEEE 6th Global Conference on Consumer Electronics (GCCE), IEEE
14. Dhyani D (2023) E-Health data risks & protection for public cloud: An elderly healthcare usecase for Swedish municipality
15. Ebrahim, M. and A. Hafid (2023). "Privacy-Aware Load Balancing in Fog Networks: A Reinforcement Learning Approach." arXiv preprint arXiv:2301.09497
16. El Kafhali S, Tayebi M, Sulimani H (2024) An Optimized Deep Learning Approach for Detecting Fraudulent Transactions. Information 15(4):227
17. Elbamby MS, Bennis M, Saad W, Latva-Aho M, Hong CS (2018) Proactive edge computing in fog networks with latency and reliability guarantees. EURASIP J Wirel Commun Netw 2018:1–13
18. Goel GAK, Chaturvedi (2023) A Systematic Review of Task Offloading & Load Balancing Methods in a Fog Computing Environment: Major Highlights & Research Areas. 2023 3rd International Conference on Intelligent Communication and Computational Techniques (ICCT), IEEE
19. Gowri V, Baranidharan B (2023) Multi Objective Hybrid Load Balancing Based Optimization Algorithm for Improving Fog Computing Performance
20. Gupta A, Gupta SK (2022) A survey on green unmanned aerial vehicles-based fog computing: Challenges and future perspective. Transactions on Emerging Telecommunications Technologies 33(11):e4603
21. Gupta H, Vahid Dastjerdi A, Ghosh SK, Buyya R (2017) iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. Software: Practice and Experience 47(9):1275–1296

22. Hussein MK, Mousa MH (2020) Efficient task offloading for IoT-based applications in fog computing using ant colony optimization. IEEE Access 8:37191–37201
23. Jebur TK (2023) Greening the internet of things: A comprehensive review of sustainable iot solutions from an educational perspective. Indonesian Journal of Educational Research and Technology 3(3):247–256
24. Jiang H, Xiao Z, Li Z, Xu J, Zeng F, Wang D (2020) An energy-efficient framework for internet of things underlaying heterogeneous small cell networks. IEEE Trans Mob Comput 21(1):31–43
25. Jiang Y-L, Chen Y-S, Yang S-W, Wu C-H (2018) Energy-efficient task offloading for time-sensitive applications in fog computing. IEEE Syst J 13(3):2930–2941
26. Jiang Y, Li C, Zhang Y, Zhao R, Yan K, Wang W (2021) Data-driven method based on deep learning algorithm for detecting fat, oil, and grease (FOG) of sewer networks in urban commercial areas. Water Res 207:117797
27. Kaur K, Sachdeva M (2020) Fog computing in IoT: An overview of new opportunities. Proceedings of ICETIT 2019:59–68
28. Khan AA, Laghari AA, Shaikh ZA, Dacko-Pikiewicz Z, Kot S (2022) Internet of Things (IoT) security with blockchain technology: A state-of-the-art review. IEEE Access 10:122679–122695
29. Kuempel CD, Adams VM, Possingham HP, Bode M (2018) Bigger or better: the relative benefits of protected area network expansion and enforcement for the conservation of an exploited species. Conserv Lett 11(3):e12433
30. Kumar MGV, Karunakaran S, Chandre S, Godi RK, Manirajkumar P, Balaram A (2023) Implementation of Microgrid Digital Twin System for Unmanned Vehicles with Cloud Computing Techniques. SN Computer Science 4(5):566
31. Li C, Zhuang H, Wang Q, Zhou X (2018) SSLB: self-similarity-based load balancing for large-scale fog computing. Arab J Sci Eng 43(12):7487–7498
32. Li Q-K, Lin H, Tan X, Du S (2018) $H\infty$ consensus for multiagent-based supply chain systems under switching topology and uncertain demands. IEEE Transactions on Systems, Man, and Cybernetics: Systems 50(12):4905–4918
33. Likas A, Vlassis N, Verbeek JJ (2003) The global k-means clustering algorithm. Pattern Recogn 36(2):451–461
34. Liu C, Wu T, Li Z, Ma T, Huang J (2022) Robust online tensor completion for IoT streaming data recovery. IEEE transactions on neural networks and learning systems 34(12):10178–10192
35. Liu J, Li G, Huang Q, Bilal M, Xu X, Song H (2022) Cooperative resource allocation for computation-intensive IIoT applications in aerial computing. IEEE Internet Things J 10(11):9295–9307
36. Lu H, Gu C, Luo F, Ding W, Liu X (2020) Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. Futur Gener Comput Syst 102:847–861
37. Lu J, Osorio C (2018) A probabilistic traffic-theoretic network loading model suitable for large-scale network analysis. Transp Sci 52(6):1509–1530
38. Lu S, Wu J, Wang N, Duan Y, Liu H, Zhang J, Fang J (2023) "Resource provisioning in collaborative fog computing for multiple delay-sensitive users." Software: Practice and Experience 53(2):243–262
39. Lyu T, Xu H, Zhang L, Han Z (2023) Source selection and resource allocation in wireless powered relay networks: an adaptive dynamic programming based approach. IEEE Internet Things J 11(5):8973–8988
40. Ma J, Hu J (2022) Safe consensus control of cooperative-competitive multi-agent systems via differential privacy. Kybernetika 58(3):426–439
41. Martinez MN, Bartholomew MJ (2017) What does it "mean"? A review of interpreting and calculating different types of means and standard deviations. Pharmaceutics 9(2):14
42. Meurisch, C., A. Seeliger, B. Schmidt, I. Schweizer, F. Kaup and M. Mühlhäuser (2015). Upgrading wireless home routers for enabling large-scale deployment of cloudlets. Mobile Computing, Applications, and Services: 7th International Conference, MobiCASE 2015, Berlin, Germany, November 12–13, 2015, Revised Selected Papers 7, Springer.
43. Mouradian C, Naboulsi D, Yangui S, Glitho RH, Morrow MJ, Polakos PA (2017) A comprehensive survey on fog computing: State-of-the-art and research challenges. IEEE communications surveys & tutorials 20(1):416–464
44. Mukherjee M, Shu L, Wang D (2018) Survey of fog computing: Fundamental, network applications, and research challenges. IEEE Communications Surveys & Tutorials 20(3):1826–1857
45. Mutlag AA, Abd Ghani MK, Mohd O, Abdulkareem KH, Mohammed MA, Alharbi M, Al-Araji ZJ (2023) A new fog computing resource management (FRM) model based on hybrid load balancing and scheduling for critical healthcare applications. Physical Communication 59:102109
46. Pavlovic D (2008) Network as a computer: ranking paths to find flows. Springer, International Computer Science Symposium in Russia
47. Qu Z, Liu X, Zheng M (2022) Temporal-Spatial Quantum Graph Convolutional Neural Network Based on Schrödinger Approach for Traffic Congestion Prediction. IEEE Trans Intell Transp Syst 24(8):8677–8686
48. Saeik F, Avgeris M, Spatharakis D, Santi N, Dechouniotis D, Violos J, Leivadeas A, Athanasopoulos N, Mitton N, Papavassiliou S (2021) Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions. Comput Netw 195:108177
49. Sarma B, Kumar R, Tuithung T (2019) Fog Computing: An Enhanced Performance Analysis Emulation Framework for IoT with Load Balancing Smart Gateway Architecture. IEEE, 2019 International Conference on Communication and Electronics Systems (ICCES)
50. Sethi V, Pal S (2023) FedDOVe: A Federated Deep Q-learning-based Offloading for Vehicular fog computing. Futur Gener Comput Syst 141:96–105
51. Sulimani H, Alghamdi WY, Jan T, Bharathy G, Prasad M (2021) Sustainability of Load Balancing Techniques in Fog Computing Environment. Procedia Computer Science 191:93–101
52. Sulimani H, Sajjad AM, Alghamdi WY, Kaiwartya O, Jan T, Simoff S, Prasad M (2022) Reinforcement optimization for decentralized service placement policy in IoT-centric fog environment. Transactions on Emerging Telecommunications Technologies 34(11):e4650
53. Sun G, Li Y, Liao D, Chang V (2018) Service function chain orchestration across multiple domains: A full mesh aggregation approach. IEEE Trans Netw Serv Manage 15(3):1175–1191
54. Sun G, Zhu G, Liao D, Yu H, Du X, Guizani M (2018) Cost-efficient service function chain orchestration for low-latency applications in NFV networks. IEEE Syst J 13(4):3877–3888
55. Tang Q, Xie R, Yu FR, Huang T, Liu Y (2020) Decentralized computation offloading in IoT fog computing system with energy harvesting: A dec-POMDP approach. IEEE Internet Things J 7(6):4898–4911
56. Tran-Dang H, Kim D.-S. (2023) Bandit Learning for Distributed Task Offloading in Fog Computing Networks: Literature Review, Challenges, and Open Research Issues. Springer, International Conference on Network-Based Information Systems
57. Tran-Dang H, Kim D-S (2023) Dynamic collaborative task offloading for delay minimization in the heterogeneous fog computing systems. Journal of Communications and Networks 25(2):244–252
58. Xiao Z, Shu J, Jiang H, Lui JC, Min G, Liu J, Dustdar S (2022) Multi-objective parallel task offloading and content caching in D2D-aided MEC networks. IEEE Trans Mob Comput 22(11):6599–6615
59. Xie H, Ding D, Zhao L, Kang K, Liu Q (2024) A two-stage preference driven multi-objective evolutionary algorithm for workflow scheduling in the Cloud. Expert Syst Appl 238:122009
60. Xu D, Liu L, Zhang N, Dong M, Leung VC, Ritcey JA (2023) Nested Hash Access with Post Quantum Encryption for Mission-Critical IoT Communications. IEEE Internet Things J 10(14):12204–12218
61. Yang D, Zhu T, Wang S, Wang S, Xiong Z (2022) LFRSNet: A robust light field semantic segmentation network combining contextual and geometric features. Front Environ Sci 10:996513

Publisher's Note