future internet

MDPI

Article

On Edge-Fog-Cloud Collaboration and Reaping Its Benefits: A Heterogeneous Multi-Tier Edge Computing Architecture

Niroshinie Fernando *, Samir Shrestha, Seng W. Loke and Kevin Lee

School of Information Technology, Deakin University, Geelong, VIC 3220, Australia; shresthasam@deakin.edu.au (S.S.); seng.loke@deakin.edu.au (S.W.L.); kevin.lee@deakin.edu.au (K.L.)
* Correspondence: niroshinie.fernando@deakin.edu.au

Abstract: Edge, fog, and cloud computing provide complementary capabilities to enable distributed processing of IoT data. This requires offloading mechanisms, decision-making mechanisms, support for the dynamic availability of resources, and the cooperation of available nodes. This paper proposes a novel 3-tier architecture that integrates edge, fog, and cloud computing to harness their collective strengths, facilitating optimised data processing across these tiers. Our approach optimises performance, reducing energy consumption, and lowers costs. We evaluate our architecture through a series of experiments conducted on a purpose-built testbed. The results demonstrate significant improvements, with speedups of up to 7.5 times and energy savings reaching 80%, underlining the effectiveness and practical benefits of our cooperative edge-fog-cloud model in supporting the dynamic computational needs of IoT ecosystems. We argue that a multi-tier (e.g., edge-fog-cloud) dynamic task offloading and management of heterogeneous devices will be key to flexible edge computing, and that the advantage of task relocation and offloading is not straightforward but depends on the configuration of devices and relative device capabilities.

Keywords: edge computing; fog computing; cloud computing; device-enhanced edge

Academic Editors: Yuezhi Zhou and Xu Chen

Received: 12 November 2024
Revised: 20 December 2024
Accepted: 1 January 2025
Published: 7 January 2025

Citation: Fernando, N.; Shrestha, S.; Loke, S.W.; Lee, K. On Edge-Fog-Cloud Collaboration and Reaping Its Benefits: A Heterogeneous Multi-Tier Edge Computing Architecture. *Future Internet* **2025**, *17*, 22. https://doi.org/10.3390/fi17010022

1. Introduction

Cloud computing, despite its widespread adoption, struggles to satisfy the latency and bandwidth demands of IoT applications, necessitating the integration of edge and fog computing to complement cloud capabilities [1,2]. These paradigms, by situating computation closer to data at the network's edge, address latency and bandwidth issues and foster a cooperative dynamic among edge, fog, and cloud layers [3,4]. In this synergy, the device-enhanced edge model enables utilising idle computational resources of IoT devices themselves [5]. This approach not only alleviates the pressure on traditional edge servers but also promotes flexibility and resource efficiency. But the deployment of a cooperative device-enhanced edge-fog-cloud architecture presents significant challenges, including task allocation across heterogeneous resources, dynamic node availability, and maintaining Quality of Service (QoS).

Building upon the Honeybee model [6], we propose an integrated architecture for coordinating resources across edge, fog, and cloud tiers, and empirically evaluate its efficacy. This paper aims to investigate the following research questions within this architecture:

- RQ1: How does node collaboration across edge, fog, and cloud layers impact overall task performance?

- RQ2: How does adjusting task-sharing parameters affect the system's behaviour?
- RQ3: How can dynamic node availability, where heterogeneous nodes may leave or join without prior warning, be handled in a cooperative edge, fog, cloud setting?

Our contributions include a conceptual architecture for dynamic edge-fog-cloud collaboration, the development of a real-world prototype, and an empirical evaluation demonstrating the approach's effectiveness in a physical testbed.

The paper is organised as follows: Section 2 outlines the background and motivation for our architectural approach. Section 3 reviews related work in computation offloading. Section 4 details the Honeybee architecture and its enhancements. Section 5 presents our experimental evaluation, followed by conclusions and future work directions in Section 7.

2. Background and Motivation

The edge-fog-cloud architecture is commonly envisioned as a 3-layered model [7–9]. In this model the edge layer is at the bottom and directly connected to end-user IoT devices, the middle tier is composed of fog nodes, and the cloud layer is at the top, as shown in Figure 1. There are other interpretations of how the interactions between edge, fog, and cloud may occur [10], and this paper follows the model in Figure 1. In this paper, we envision a computing architecture where the IoT devices themselves are utilised as edge resource providers. The idea of a collection of end-user IoT devices collaboratively working as a collective computing resource has been explored in the literature under various terms, such as 'mobile device clouds' [11], 'mobile edge-clouds' [12], 'human-driven edge computing [13], 'collaborative multi-device computing' [14], 'mobile crowd computing' [6] and 'device-enhanced multi-access edge computing' [5] (device-enhanced MEC). In this paper, we use the term 'device-enhanced MEC' to refer to the bottom tier of Figure 1, which provides edge computing services to end-user IoT devices.

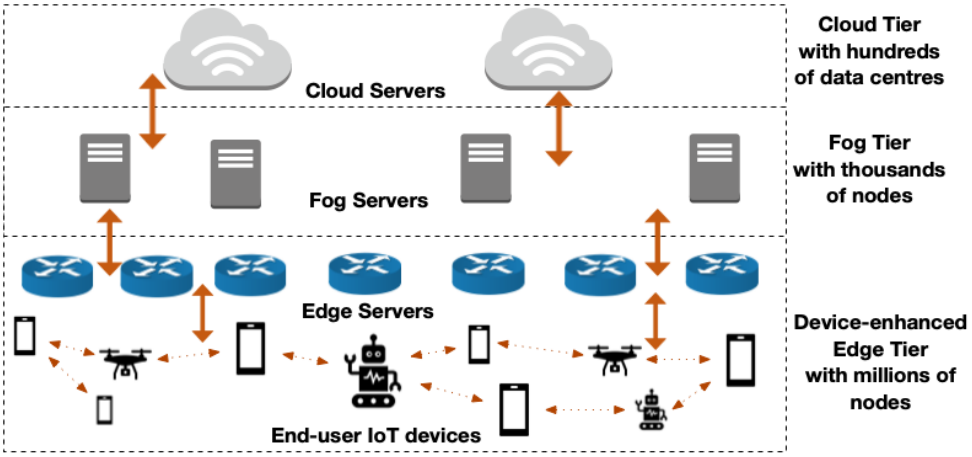


Figure 1. A 3-tier architecture for device-enhanced edge, fog, and cloud computing. End-user IoT devices such as smartphones, drones, and robots are integrated as edge resources, forming a local collective resource, and work collaboratively with conventional edge, fog, and cloud servers.

As shown in the aforementioned research, integrating end-user IoT devices to the edge tier as resource providers can provide performance gain, energy savings, and increased resource utilisation and availability. However, when considering real-life constraints, the co-cooperation amongst device-enhanced edge, fog, and cloud layers becomes essential. We illustrate this in the motivating scenario illustrated in Figure 2.

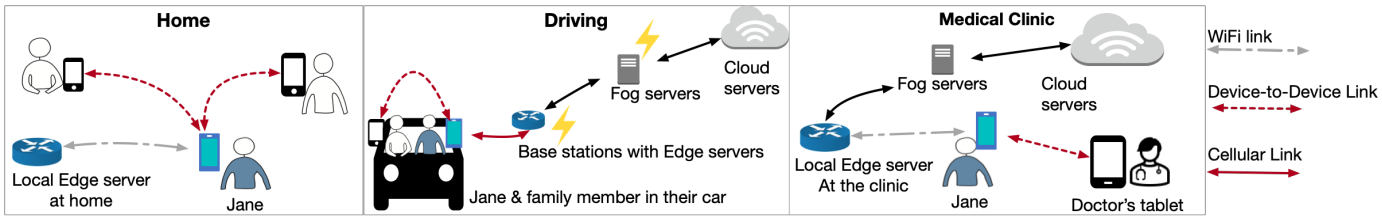


Figure 2. Scenario showing how different contexts require collaboration amongst resource nodes at edge, fog, and cloud tiers.

Here, Jane, has a medical condition which requires her to use a wearable device for constant monitoring of health data. The application QoS requirements in this case include near-real-time response time, a particular level of accuracy, high availability, and secure handling of Jane's sensitive medical data. She is also concerned about the battery drain of her mobile device and data access fees.

The wearable device sends the sensor data to her smartphone for analysis which is not powerful enough to run the AI algorithms required to process this high-velocity stream of data continuously. Hence, this application requires the support of external resources to provide accurate and timely results. Jane's smartphone has a collaborative resource-sharing middleware installed, which runs in the background of the smartphone and the resource nodes, and acts as an intermediary between applications and external resources. Applications that need external resources connect to available resources through the middleware.

In Figure 2, when Jane is at home, Jane's smartphone connects her to a local collective resource, made up of the collective resources of her family members' smartphones and her home edge server to carry out the sensor data processing tasks. In this situation, all of the participating resource nodes can be considered to be at the edge. Since all of the nodes are trusted, and connectivity is robust, Jane does not have the need to request the support of remote servers.

When Jane is traveling in the car with another family member, the local collective resource is made of just two smartphones-Jane's and her family member. Since this particular resource collection does not provide sufficient computing resources to satisfy the QoS requirements of Jane's app, she further shares the app workload with a conventional edge server located at a base station, as well as with fog and cloud servers. In this situation, the entire workload is not offloaded to the servers, but shared amongst the two smartphones (which form the local collective resource in this case) as well.

The app workload involves a large amount of data; hence, reducing data transfers via cellular links is beneficial, due to latency, data access fees, and energy usage. During the drive, Jane's smartphone suffers intermittent data connectivity, and so a considerable amount of work is being supported by the local collective resource, instead of the other servers. How much work is performed by the two smartphones (the local collective resource) and the conventional edge, fog, and cloud servers can depend on their availability and latency, as well as the task scheduling algorithm in Jane's collaborative resource sharing middleware. When the cellular data connectivity drops, this can increase the latency of conventional edge, fog, and cloud servers, impacting their performance. In this case, the two smartphones should be able to pick up most of the workload.

When Jane is at the medical clinic (rightmost situation in Figure 2), her smartphone forms a local collective resource with her doctors tablet via D2D and with the clinic edge server via WiFi. She further shares her work with fog and cloud servers because the doctor's diagnostic process requires the app to provide faster performance. As Jane uses the clinic's high-speed WiFi to connect to the remote servers, there is no concern of intermittent data

connectivity or data access fees. In this situation, the fog and cloud servers should be able to pick up most of the workload due to the availability of high-speed connectivity.

In the three aforementioned contexts described in the scenarios in Figure 2, Jane shared her work with resource nodes at the device level, edge, fog, and cloud levels in various degrees. The three situations had different constraints in terms of connectivity, trust and security, and amount of available resources, yet the application QoS requirements remained the same. Hence, to continue to meet the QoS requirements, the collaboration across edge, fog, and cloud needs to adapt dynamically depending on the context.

3. Related Work

Existing surveys on edge, fog, and cloud computing have comprehensively discussed the synergy between IoT and edge, fog, and cloud computing paradigms, explaining how edge and fog computing can bridge the gap between IoT and cloud computing by moving the computation closer to the end-user IoT devices, thereby addressing issues with energy, latency, and context awareness [4,5,15–18]. Numerous scholarly papers have extensively addressed diverse facets of edge, fog, and cloud tiers, taken as separate tiers. Specifically, for device-enhanced MEC, additional complexities need to be considered due to the dynamic nature of device-based resource providers and their intrinsic characteristics, such as distributed ownership, mobility, finite energy, resource constraints, increased proximity to other device-based resource providers, and Device-to-Device (D2D) communication capacity [5,15,19–22]. Various aspects of the technical feasibility of the lowest tier of device-enhanced MEC, where devices such as smartphones function as edge resource providers, have been demonstrated in frameworks such as MMPI [23], Hyrax [24], MClouds [25], Aura [26], and Honeybee [6]. However, these existing works have not investigated how the device-based resource providers can collaborate and share work with nodes at fog and cloud layers, considering various overheads, impacts on performance, and battery and various offloading parameters.

As highlighted in a recent work on the convergence of edge, fog, and cloud, only a few papers have yet investigated the interactions between these three paradigms in a 3-tier edge-fog-cloud architecture [27]. Below, we discuss related work that has explored this under-researched area. Here, we only focus on work that has investigated the interactions and collaborations between at least two of the edge, fog, and cloud tiers, and do not consider work that only focuses on one tier.

One of the first papers to explore **edge-fog-cloud** collaboration is Flores et al. [20], who proposed the HyMobi framework, which allows a mobile application to interoperate between device-enhanced MEC, fog, and cloud tiers. HyMobi has an incentive mechanism based on credit and reputation, and allows users to lease the resources of their devices as an open commodity in the edge tier, exploiting the social characteristics of the devices to form offloading communities. The effectiveness of HyMobi is demonstrated via a proof-of-concept implementation and a testbed, evaluating the performance and energy consumption of mobile applications and infrastructure awareness in a social-aware environment. However, the performance results are not explored thoroughly with different edge-fog-cloud combinations. Other papers that discuss all three tiers of edge-fog-cloud include [21,28,29]. In [21], a task can be offloaded to either edge, fog or cloud, or executed on the device itself; however, no collaboration amongst the tiers is mentioned. Although the algorithm in [28] can be applied to all three layers of edge-fog-cloud to reduce latency and power consumption, the paper only considers the fog layer. In [29], a Min-Min algorithm, considering cost, makespan, energy, and load balancing, is used to schedule tasks amongst the edge-fog-cloud tiers.

The coordination of work offloading amongst the **edge and cloud** tiers is discussed in [22,30,31]. In [22], the authors propose the HyFog framework, which considers interactions between device-enhanced MEC and cloud, and allows devices to choose the execution mode among local mode, D2D mode, and cloud mode. Simulations show that HyFog's three-layer graph-matching algorithm-based solution is able to minimise the power consumption while ensuring latency requirements. However, there is no discussion about adapting to dynamic conditions, such as device mobility. In comparison, refs. [19,32] only consider collaborated work offloading inside the device-enhanced MEC layer itself, between the devices as resource providers and the conventional edge server/s.

Offloading in hybrid **fog/cloud** systems is discussed in [33–35]. In [33], the authors present a scheme for the joint optimisation of transmit power control, computation and radio bandwidth allocation when offloading in hybrid fog/cloud systems, while guaranteeing user fairness and maximum tolerable delay. In [34], Zahoor et al. present a cloud-fog-based architecture in the context of a smart grid. The authors discuss simulation results of using round robin, throttled, and particle swarm optimisation algorithms to schedule requests from devices such as smart meters on the VMs of a fog-cloud architecture. However, it is unclear if there is any collaboration amongst the fog-cloud tiers. Kumar and Karri [35] present the EEOA (electric earthworm optimisation algorithm) for efficient resource assignment and work scheduling amongst fog-cloud tiers, considering the makespan, cost, and energy usage. Simulation results show that in many cases, the proposed EEOA performs better than alternative methods.

Overall, few papers have considered collaboration amongst all three tiers of edge-fog-cloud [20,21,28,29]. However, amongst all the papers considered in this section, only [20] has used an actual test bed without solely depending on simulations. Although important, simulations lack real-world variability, and do not always capture emergent behaviours arising from the interactions of various components in the edge, fog, and cloud tiers. Only work reported in [19,20,22,29–32] has provided clear evidence of supporting inter-tier interactions and collaborations, and a majority do not discuss support for dynamic conditions. In this paper, we address the gaps highlighted above.

4. An Architecture for Edge-Fog-Cloud Offloading

This section introduces our architecture for supporting inter-tier collaboration amongst edge, fog, and cloud tiers in a device-enhanced MEC context. We have chosen to extend and build on the Honeybee framework since it is open-source, supports proactive worker-centric offloading, and has automatic load-balancing and fault-tolerant mechanisms. Honeybee has also been extended in other work to work with drones [36], robots [37] and dependency-based task scheduling [38], showing its extensibility.

The Honeybee framework (https://github.com/niroshini/honeybee, accessed on 18 December 2023) is an Android implementation of the Honeybee mode [6,39], and enables the formation of mobile edge clouds, via peer-to-peer (P2P) connections using Wi-Fi Direct. In the Honeybee model, the device with the task to be completed is called the 'delegator'. The original task is first decomposed to a pool of jobs, and then is offloaded to 'worker' devices in the vicinity, while also carrying out a portion of the jobs by itself. Honeybee's task-scheduling algorithm is based on work stealing [40] to automatically load-balance the jobs amongst the delegator and the workers. The workers must proactively request to 'steal' jobs, and each time the delegator receives a steal request, it will respond to the requesting worker with a chunk of jobs. In this way, faster workers are able to steal more and more jobs, thus avoiding performance bottlenecks with faster nodes having to wait for slower nodes. The Honeybee model is also able to handle random disconnections as well as exploit random resource node encounters. To address the critical aspect of

incentivising participation of workers, we have previously explored the socio-technical requirements and engineering challenges of this paradigm, identifying effective incentive mechanisms [41]. Our findings provided insights into aligning user motivations with application goals through both intrinsic-social and extrinsic-personal incentives, ensuring sustained engagement while mitigating resource depletion concerns. However, in this work, we do not focus on the design of incentive structures, instead assuming that appropriate mechanisms are already in place across the edge, fog, and cloud layers to support inter-tier collaboration. In this paper, we build on the Honeybee model to extend the cooperative work-stealing mechanism beyond the original Honeybee's local device cloud. We have extended the Honeybee delegator to be able to simultaneously maintain connections with multiple P2P edge servers via Wi-Fi Direct, as well as fog and cloud servers via the LAN and the internet, respectively. The Honeybee worker component has been extended to include support for Java implementations on fog and cloud servers. While our previous work [38] extended the Honeybee framework to support sequential dependency tasks, this study does not utilise the dependency-enabled extension, as it lies beyond the scope of the present investigation. Figure 3 shows a high-level view of the architecture of the extended framework for Edge-Fog-Cloud collaboration.

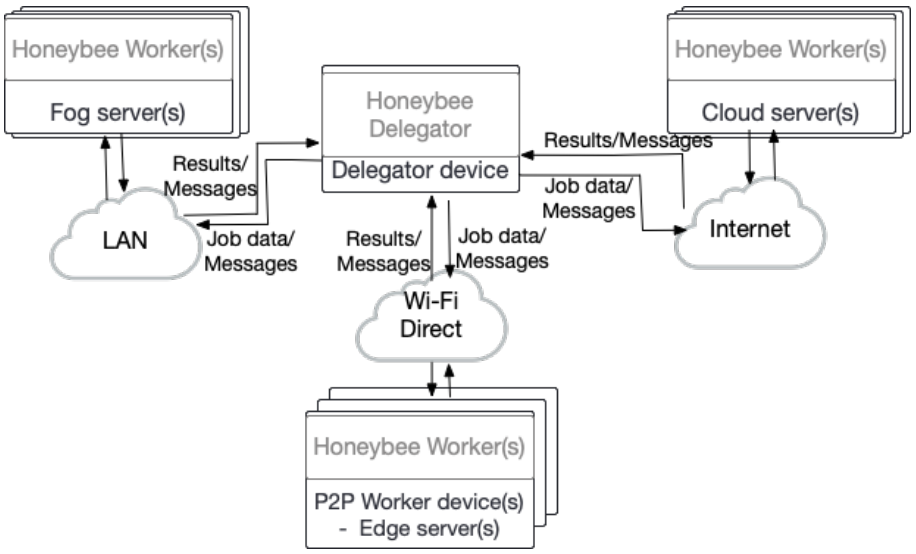


Figure 3. The edge-fog-cloud collaborative architecture.

Algorithms 1–4 provide an overview on how resource nodes at the edge, fog, and cloud collaboratively work on a distributed set of jobs. As shown in Algorithm 1, the delegator first initialises the job pool and starts consuming the jobs (for processing), while concurrently, it also starts the resource discovery threads for edge, fog, and cloud workers. Algorithm 3 describes the generalised periodic resource discovery thread. The delegator spawns three instances of this thread for workers at the three tiers of edge ($\mathcal{W}_e$), fog ($\mathcal{W}_f$), and cloud ($\mathcal{W}_c$). Each tier uses communication protocols applicable for that tier. Each instance of the three resource discovery threads operates independently, discovering resources specific to the worker type. Whenever a worker node is discovered, the system attempts to establish connections with them, via WiFi-Direct (for P2P edge), LAN (for fog), and internet (for cloud). Upon connecting, each worker will attempt to 'steal' from the delegator's job pool (see Algorithms 4 and 5). Whenever a worker's share of jobs is completed, it will send the results to the delegator, and without waiting to be assigned jobs, will **proactively** attempt to 'steal' jobs from the delegator.

Algorithm 1 Delegator's main thread

Input: Job pool $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$ where $n > 0$, delegator d, edge workers $\mathcal{W}_e$, fog workers $\mathcal{W}_f$, cloud workers $\mathcal{W}_c$
Output: Completed jobs $\mathcal{J}c$, initially $\mathcal{J}c = \emptyset$
Initialise $\mathcal{J}$ to ensure $\mathcal{J} \neq \emptyset$
Start delegator's job consumer thread T_{d_con}
Start $T_{d\_comEdge}$ for edge workers
Start T_{d_comFog} for fog workers
Start $T_{d\_comCloud}$ for cloud workers
while $\mathcal{J} \setminus \mathcal{J}c \neq \emptyset$ **do**
    T_{d_con} consume $\mathcal{J}$
    Update $\mathcal{J}c$ with results from T_{d_con}
end while

Algorithm 2 Delegator's job consumer thread

Input: $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$, $n > 0$, d, $\mathcal{W}_e$, $\mathcal{W}_f$, $\mathcal{W}_c$
Output: $\mathcal{J}c$
Let $\mathcal{W} = \mathcal{W}_e \cup \mathcal{W}_f \cup \mathcal{W}_c$
while $\mathcal{J} \setminus \mathcal{J}c \neq \emptyset$ **do**
    if $\mathcal{J} \neq \emptyset$ **then**
        Consume $\mathcal{J}$
        Update $\mathcal{J}c$
    else Steal from a worker in $\mathcal{W}$ ▷ Traverse workers in connection order
    end if
end while

Algorithm 3 Delegator's periodic resource discovery thread for worker type W

Input: Worker type $W \in \{\mathcal{W}_e, \mathcal{W}_f, \mathcal{W}_c\}$; Communication protocol $Comm$
Output: Established connections stored in a ConcurrentHashMap $\mathcal{M}$
Initialise $\mathcal{M}$ as an empty ConcurrentHashMap
while resources are periodically discovered for W **do**
    for each worker $w \in W$ **do**
        Open socket connection using protocol $Comm$
        Create a *ClientSocketThread* instance for w
        $\mathcal{M}[w] \leftarrow$ ClientSocketThread instance
        Start a communication thread for w ▷ Algorithm 5
    end for
end while

Algorithm 4 Worker's main thread

Input: Worker job pool $\mathcal{J}_w$, initially $\mathcal{J}_w = \emptyset$, Stolen jobs $\mathcal{J}s$, d
Output: Worker's completed jobs $\mathcal{J}c_w$, initially $\mathcal{J}c_w = \emptyset$
Initialise: work_ongoing $\leftarrow$ true
while work_ongoing **do**
    if $\mathcal{J}s \neq \emptyset$ **then**
        $\mathcal{J}_w \leftarrow \mathcal{J}_w \cup \mathcal{J}s$
        while $\mathcal{J}_w \neq \emptyset$ **do**
            Consume $\mathcal{J}_w$
            Update $\mathcal{J}c_w$ and send to d
        end while
    else Steal from d
    end if
end while

Algorithm 5 Delegator's communication thread for each worker

Input: Worker w, job pool $\mathcal{J}$, connection map $\mathcal{M}$
Output: Updated $\mathcal{J}$, completed results $\mathcal{J}c$
Retrieve w's connection details from $\mathcal{M}$
Send *init_signal* to w
while *read* $\neq \emptyset$ **do**
    $m \leftarrow$ *read* ▷ m: received message
    if $m =$ steal_request **then**
        Send *no_jobs_left* if $\mathcal{J} = \emptyset$, else start victim thread T_{d_vic}
    else if $m =$ results **then**
        $\mathcal{J}c \leftarrow \mathcal{J}c \cup$ new results ▷ Store results
        if $\mathcal{J} \subseteq \mathcal{J}c$ **then**
            Execute *on_task_completed*
        end if
    else if $m =$ stolen_jobs **then**
        $\mathcal{J} \leftarrow \mathcal{J} \cup$ stolen jobs; start job consumer thread T_{d_con}
    else if $m =$ no_jobs **then**
        Mark w as idle
        if expired jobs $\neq \emptyset$ **then**
            $\mathcal{J} \leftarrow \mathcal{J} \cup$ expired jobs
        end if
    else if $m =$ worker_heartbeat **then**
        Update w's status in $\mathcal{M}$
    end if
end while

Algorithm 5 manages all communication between the worker node and the delegator node. This includes handling job-stealing requests, processing completed results, delegator functioning as a victim when a worker steals jobs from the delegator (described in Algorithm 6), expiring the oldest jobs (detailed in Algorithm 7), and sending termination signals upon task completion (detailed in Algorithm 8).

Algorithm 6 Delegator's victim thread

Input: Job pool $\mathcal{J}$, steal limit L_s, chunk size C_s
Output: Stolen job list $\mathcal{J}s$
Initialise $\mathcal{J}s \leftarrow \emptyset$
if $\mathcal{J} \neq \emptyset$ **then**
    if $|\mathcal{J}| > L_s$ **then**
        job $\leftarrow$ get first job in $\mathcal{J}$
    end if
    while *job* $\neq$ null **do**
        $\mathcal{J}s \leftarrow \mathcal{J}s \cup \{job\}$ ▷ Add job to stolen list
        if $|\mathcal{J}| \leq L_s$ **then**
            break
        end if
        if $|\mathcal{J}s| \geq C_s$ **then**
            break ▷ Steal chunk size is constant across all workers
        end if
        job $\leftarrow$ get next job in $\mathcal{J}$
    end while
end if
Transmit $\mathcal{J}s$ to worker

Algorithm 7 Delegator's job expiration method

Input: Pending results list $\mathcal{R}$
Output: Expired jobs $\mathcal{J}_{\text{expired}}$
Initialise $\mathcal{J}_{\text{expired}} \leftarrow \emptyset$
if $\mathcal{R} \neq \emptyset$ **then**
    Select worker w_{ex} with the oldest pending jobs from $\mathcal{R}$
    $\mathcal{J}_{\text{expired}} \leftarrow$ pending jobs of w_{ex}
end if
return $\mathcal{J}_{\text{expired}}$

Algorithm 8 Delegator's on_task_completed method

Input: Connection map of all connected workers $\mathcal{M}$
Output: Task completion signal sent, all connections closed
for all connected workers $w \in \mathcal{M}$ **do**
    Send *termination_signal* to w
    Close connection with w
end for

Over the course of program execution, any participating device can assume both roles of *thief* and *victim*. Hence, the worker can 'steal' from the delegator and the delegator can also 'steal' from workers if its own job pool is empty, but the task is not yet completed (see Algorithm 2). When a device receives a steal request, depending on its available job queue, it can decide to become a victim (see Algorithm 6). The victim then removes a C_s number of jobs (i.e., a chunk) from its own job list, and transmits them to the thief. The chunk size, C_s, defines the number of jobs a node can steal at a time. It is set as a small percentage of the total job count to minimise loss and enable easy reassignment if the worker disconnects.

- Stealing decisions: When the delegator steals from workers, workers are traversed sequentially in the order they connected to the delegator. The delegator attempts to steal jobs from each worker until successful or until all workers have been tried. Workers can only steal from the delegator due to an implementation constraint in peer-to-peer (P2P) connection protocols. In technologies such as Bluetooth and Wi-Fi Direct, only a star topology is supported, not a mesh. As a result, P2P connections between workers are not feasible.
- Conflict avoidance: The delegator maintains a synchronised lock on the job pool $\mathcal{J}$ during job allocation to prevent race conditions. This ensures that jobs are not simultaneously assigned to multiple workers.
- Complexity and overhead: The sequential worker selection ensures that worker traversals are $O(n)$, where n is the number of connected workers. Locking and job allocation operations remain $O(1)$, ensuring low overhead for each steal request.
- Fault tolerance: Worker heartbeats and job expiry are two main fault tolerance mechanisms in the framework. Each worker periodically sends a 'heartbeat' signal to the delegator to indicate its availability (see Algorithm 5). Other communications, such as job results, are also considered as heartbeats. If no heartbeat is received from a worker for a pre-determined consecutive interval, the delegator deems the worker as disconnected, and their assigned jobs are returned to the job pool for reassignment. If a job remains uncompleted beyond a predefined expiry time, it is marked as 'expired' and returned to the job pool, allowing other nodes, including the delegator, to process it (see Algorithms 5 and 7). The delegator invokes the job expiry mechanism only after its own job queue is exhausted and a steal attempt fails, ensuring efficient resource utilisation while preventing indefinite delays.

This proactive design lends itself for more opportunism as the availability and resourcefulness of each worker is unknown a priori, and subject to change any time [6]. For example, a worker's availability can be impacted if/when a worker node receives a call while it is participating in task execution, or if its location changes due to the owner moving away. This process continues until the pool is exhausted, or until a worker disconnects. The resource discovery threads are periodic; hence, potential workers can join at any point in time.

4.1. Upper and Lower Bounds for Speedup

In this section, extending previous work in Honeybee [6], we formulate theoretical upper and lower bounds to analyze the best- and worst-case speedup scenarios within a device-enhanced edge-fog-cloud computing architecture.

We assume that a given edge-fog-cloud computing environment consists of x P2P edge nodes, including the delegator, y fog workers, and z cloud workers, where x, y, z are non-negative integers.

Let us denote each node in the device-enhanced edge layer as n_{e_i}, where $1 \leq i \leq x$. The delegator is denoted as n_{e_1}. Each fog worker is denoted as n_{f_j}, where $1 \leq j \leq y$, and each cloud worker is denoted as n_{c_l}, where $1 \leq l \leq z$.

The time taken to complete m jobs on the delegator n_{e_1} is denoted as t_{e_1}. The time taken for an edge worker n_{e_i} to receive, complete, and return results for m jobs is denoted as t_{e_i}, where $i > 1$. Hence, the relative computational power of an edge worker n_{e_i} compared to the delegator n_{e_1} can be given as:

$$\frac{t_{e_i}}{t_{e_1}} = k_{e_i} \tag{1}$$

Similarly, the time taken for a fog server n_{f_j} to complete and return results for m jobs is denoted as t_{f_j}, and for a cloud server n_{c_l} as t_{c_l}. The relative computational powers of fog and cloud workers compared to the delegator n_{e_1} can be represented by constants k_{f_j} and k_{c_l}, respectively:

$$\frac{t_{f_j}}{t_{e_1}} = k_{f_j} \tag{2}$$

$$\frac{t_{c_l}}{t_{e_1}} = k_{c_l} \tag{3}$$

The parameters k_{e_i}, k_{f_j}, and k_{c_l}, introduced in Equations (1), (2), and (3), respectively, serve as indicators of the relative efficiency of task execution by worker nodes in comparison to the delegator. Specifically, a value of $k < 1$ signifies that the worker node, be it within the edge, fog, or cloud tier, can process and return the results of the assigned tasks more expediently than the delegator performing the same tasks in a monolithic manner. This computational advantage highlights the potential benefit of offloading tasks to worker nodes with $k < 1$. Conversely, a value of $k \geq 1$ suggests that task offloading may not yield substantial speedups, as the worker node does not demonstrate a computational speed advantage over the delegator.

Even when $k > 1$, indicating a worker's lower computational power compared to the delegator, offloading can still enhance overall performance. This improvement arises because the delegator and workers operate concurrently, increasing total system throughput. Essentially, the collective output of all computing nodes, despite individual limitations, can contribute to achieving speedup. Thus, the decision to offload tasks should consider the system-wide contribution, highlighting the importance of a holistic approach in optimising the edge-fog-cloud architecture's efficiency.

It is important to note that the relative computational power indicators k_{e_i}, k_{f_j}, and k_{c_l} are assumed to be constant for the purposes of this theoretical analysis. This simplification is intended to provide a tractable baseline for deriving upper and lower bounds on speedup. In practice, these indicators may vary dynamically due to fluctuations in network traffic (e.g., WLAN, LAN, and internet) and the computational load on worker devices, especially edge devices with constrained resources. These variations can influence task completion times and, consequently, the speedup achieved in real-world scenarios.

4.1.1. Best-Case Scenario

Deriving from [6], only considering the P2P edge workers, assuming jobs of equal computational complexity and ignoring overheads, the upper bound for speedup can be given as:

$$S_{\text{upper}} = 1 + \sum_{i=2}^{x} \left(\frac{1}{k_{e_i}} \right) \tag{4}$$

This formula illustrates the ideal scenario where the combined effort of all devices maximises the task processing speed. Realistically, it should be noted that given the presence of overheads and the variability in job sizes, the realised speedups are likely to fall below this theoretical maximum.

Incorporating the fog and cloud workers, the updated formula for the upper bound for speedup becomes:

$$S_{\text{upper}} = 1 + \sum_{i=2}^{x} \left(\frac{1}{k_{e_i}} \right) + \sum_{j=1}^{y} \left(\frac{1}{k_{f_j}} \right) + \sum_{l=1}^{z} \left(\frac{1}{k_{c_l}} \right) \tag{5}$$

4.1.2. Worst-Case Scenario

The lower bound for speedup is formulated, extending from [6], for edge, fog, and cloud workers as follows.

We consider the worst-case scenario where the collective capability of the worker devices across the edge, fog, and cloud tiers is significantly less than that of the delegator. This scenario might occur due to high network latency, worker unavailability, or extremely weak workers.

In such situations, the delegator undertakes the parallelised task execution without the assistance of any worker nodes, incurring overheads, including costs related to task parallelisation and ongoing worker search efforts, as well as communication costs across the edge, fog, and cloud layers.

The utmost job completion time, t_{worst}, is represented as:

$$t_{\text{worst}} = t_M + c_{\text{edge}} + c_{\text{fog}} + c_{\text{cloud}} + e_{\text{edge}} + e_{\text{fog}} + e_{\text{cloud}}$$

Under these assumptions, the lower bound for speedup can be given as:

$$S_{\text{lower}} > \frac{t_M}{t_M + c_{\text{edge}} + c_{\text{fog}} + c_{\text{cloud}} + e_{\text{edge}} + e_{\text{fog}} + e_{\text{cloud}}} \tag{6}$$

This formulation reflects the minimal speedup achievable under adverse conditions.

5. Experimental Evaluation

An image processing task (face detection) was chosen as the computational task to be shared amongst the nodes. The job pool on the delegator consists of 1000 unique PNG images at 1024 × 1024 resolution, obtained from the FFHQ dataset (https://github.com/NVlabs/ffhq-dataset, accessed on 4 March 2024). We selected images starting from

filenames 66000.png to 66999.png. The mean image size was 1.3 MB, with a median of 1.31 MB and a standard deviation of 0.161. The images were stored in the delegator device at the start of each experiment. The nodes used for the experiment are given in Table 1. These devices represent a range of computational capacities across the edge, fog, and cloud layers. The Moto G5S Plus (D1) represents a constrained edge device typical in IoT setups, while the OnePlus 6 (D2) demonstrates a more powerful edge environment. The Dell Inspiron 5502 (F1) serves as a fog server, representing resource-rich devices for intermediate computation and the AWS EC2 t3.xlarge instance (C1) exemplifies a high-performance cloud resource for centralised processing. These choices demonstrate the applicability of our approach to devices across a wide spectrum of computational capacities, from constrained edge devices to powerful cloud resources.

All results were obtained from experiments conducted on the specified physical devices, ensuring a realistic evaluation of the system's performance. No simulations were used. The aims of each experiment and related RQs are given in Table 2. Each experiment was conducted five times, and the results were averaged to ensure statistical reliability. The network capacity and latency details for the three different edge-fog-cloud tier networks in the experiments are given in Table 3.

Table 1. Specifications of the delegator and worker nodes.

Node	CPU	RAM	OS
Moto G5S Plus (D1)	Qualcomm MSM 8953 Snapdragon 625	4 GB	Android 8.1
Oneplus6 (D2)	Qualcomm SDM845 Snapdragon 845	8 GB	Android 11
Dell Inspiron 5502 (F1)	11th Gen Intel(R) Core(TM) i7-1165G7 @2.80GHz 1.69GHz	16 GB	Microsoft Windows 10 Pro (x64)
AWS EC2 instance t3.xlarge (Cx where $x \in [1..12]$)	4vCPU upto 3.1 GHz Intel Xeon Platinum Processor	16 GB	Ubuntu Server 20.04 LTS

Table 2. Experiment overview.

Experiment	Aim	RQ
1,2,3,4,5,6,9	Compare the performance of the system when there are multiple configurations of edge, fog, cloud nodes, with heterogeneous platforms, capacities, connection protocols, working together	RQ1
7	Investigate the impact of task sharing configuration parameters in an edge, fog, cloud collaboration setup	RQ2
8, 10	Investigate the impact of dynamically adding/removing or slowing down worker nodes	RQ3

Table 3. Network capacity and latency details for different network types.

Network Type	Latency Range (ms)	Bandwidth (Mbps)
P2P Edge (Wi-Fi Direct-based)	5–10	50–100
Fog Server (LAN-based)	5–30	100–150
Cloud Server (Internet-based)	200–400	30–50

5.1. Experiment 1: Baselines

This experiment examines the computational capacities of each worker node in comparison to the delegator. This sets the baseline values to understand the performance gains and battery usage for the later experiments. The delegator first executes the entire task monolithically (the 'monolithic version' refers to the task without any of the parallelising components). Next, the delegator offloads the entire task to each worker, and each node processes the entire task sequentially. Two sets of baseline experiments are carried out:

(1) D1 as delegator, with D2, F1, and C1 as workers, and (2) D2 as delegator, with D1, F1, and C1 as workers. The speedup metric, denoted as S, provides the relative performance gain achieved through offloading compared to monolithic execution. Table 4 shows the results of this study. Note that Avg. Tr. time denotes the average transmission time per job in the table. This includes all associated communication delays, such as the propagation time, any network routing or switching delays, and job preparation and acknowledgment processing at both ends. Battery % indicates the battery usage of the delegator during the execution of the experiment. This was measured by recording the battery percentage of each device just before the experiment started and immediately after it ended.

Table 4. Experiment 1: Baseline experimental results for each worker with D1 and D2 as delegator.

Node	D1 as Delegator				D2 as Delegator			
	S	**Battery (%)**	**Avg Tr. Time (ms)**	**Avg Total Time (ms)**	**S**	**Battery (%)**	**Avg Tr. Time (ms)**	**Avg Total Time (ms)**
Delegator	N/A	17.60	N/A	3,900,012	N/A	8.99	N/A	1,042,548
Fog	2.70	4.00	797	1,446,771	0.85	5.60	627	1,043,236
Cloud	1.43	8.39	1948	2,725,512	0.56	7.39	1286	1,674,836
Edge	2.57	5.20	184	1,515,956	0.17	23.60	2002	6,089,051

With D1 as delegator, the results indicate that D2, F1, and C1 exhibit speedup factors of approximately 2.57, 2.70, and 1.43 over D1, respectively, Note that this comparative performance takes transmission delays into account. This is why even though the fog (F1) and cloud (C1) nodes may be computationally more resource rich than the P2P edge node (D2), the overall performance in D1's perspective is less than D2. As can be seen from the battery usage on D1, offloading significantly reduces D1's battery consumption, even with the additional communication costs.

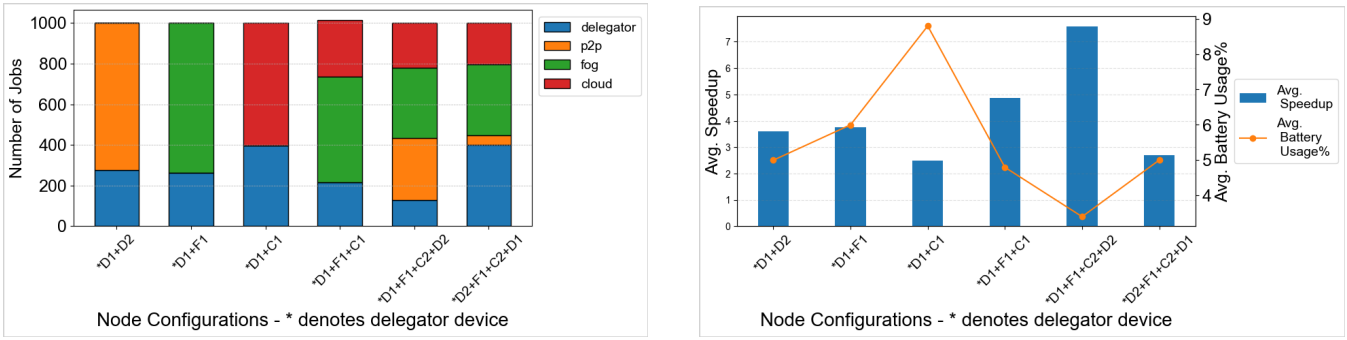
With D2 as delegator, F1's speedup (0.85) is only slightly lower than D2's monolithic execution, but C1 demonstrates a marked decrease in speedup with S = 0.56. Despite C1 having significant computational power, due to latency issues that are well known with cloud computing, jobs offloaded to C1 take more time to return results. Finally, the P2P edge node (D1) is the slowest, with S = 0.17. This is due to D1 being significantly less powerful than D2 (see Table 1).

5.2. Experiment 2: Delegator and P2P (Edge)

In this experiment, the delegator D1 collaborates with a peer-to-peer edge device D2. Comparative analysis indicates that D2 operates at a rate 2.57 times faster than D1, as detailed in Section 5.1. D2's superior computational capacity compared to D1 is further demonstrated by D2 executing approximately 2.6 times as many jobs as D1. This performance discrepancy is clearly depicted in Figure 4a. Hence, this configuration achieves a speedup of 3.59, as documented in Figure 4b. The average task completion time for this experiment was 1,084,998 ms.

5.3. Experiment 3: Delegator and Fog

In this experiment, D1 cooperates with F1 to complete the jobs together. This results in a speedup of 3.76, as seen in the Figure 4b. This is also due to the fact that F1 is more powerful than D1, as evidenced by the number of jobs completed by each node, as seen in Figure 4a. The average task completion time for this experiment was 1,038,146 ms.



(**a**) (**b**)

Figure 4. Results for Experiments 2–6 and 9. (**a**) Jobs completed by each node for Experiments 2–6,9. (**b**) Speedup gains and battery usage for varying node configurations.

5.4. Experiment 4: Delegator and Cloud

Here, D1 shares the workload with C1, resulting in a speedup of 2.48, as seen in Figure 4b. Node C1 is more powerful than the delegator D1, as evidenced by the number of jobs completed by each node in Figure 4a. The average task completion time for this experiment was 1,571,844 ms.

5.5. Experiment 5: Delegator, Fog, and Cloud

In this experiment, the delegator (D1) is collaborating with both F1 and C1. As can be seen from from Figure 4b, this results in a speedup of 4.85. The average task completion time for this experiment was 803,566 ms.

5.6. Experiment 6: Delegator, P2P, Fog, and Cloud

In this experiment, a comprehensive 3-tier configuration, comprising a P2P edge device (D2), a fog server (F1), a cloud server (C1), and the delegator (D1), collaboratively processes the tasks. This integrated architecture achieves a significant speedup of approximately 7.5, as documented in Figure 4b. The average task completion time for this experiment was 514,485 ms. D2 and F1, being the fastest workers, are able to complete almost two-thirds of the total jobs (Figure 4a). Figure 5 illustrates the cumulative number of jobs completed (y-axis) over time in milliseconds (x-axis) for each node (D1, D2, F1, C1). The lines represent the total number of jobs completed by each node at any given point in time. The gap between the lines provides an indication of how many jobs each device has completed relative to the others, with a larger gap signifying a higher contribution.

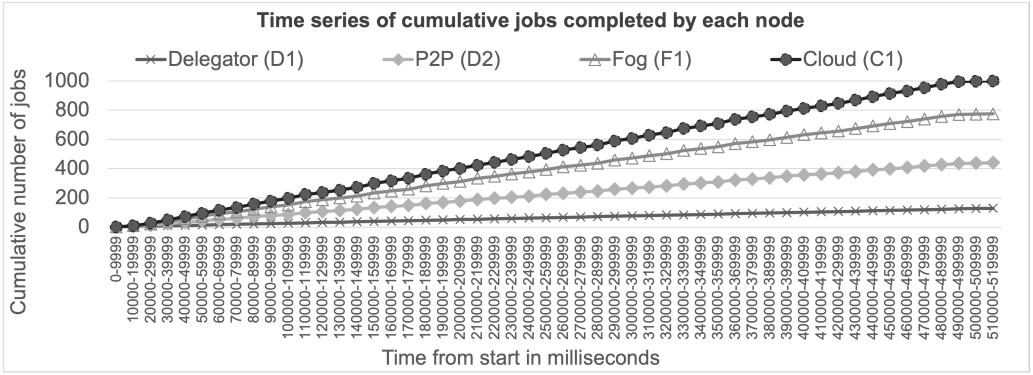
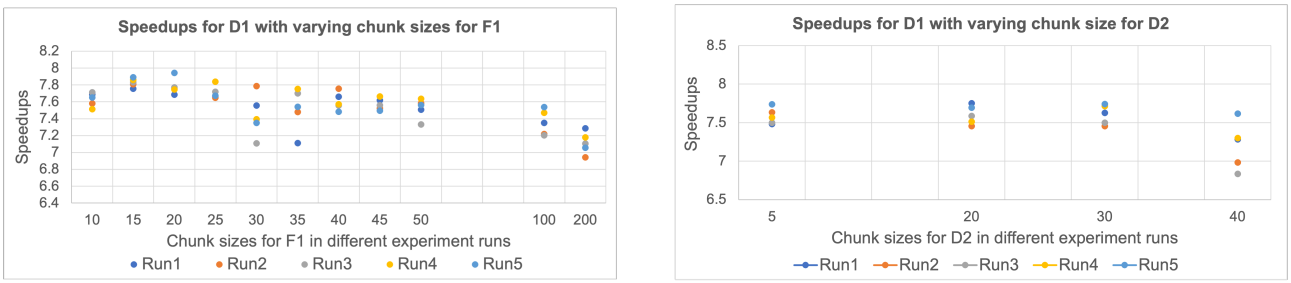


Figure 5. Time series of number of jobs completed by each node. Experiment 6: Results for D1 working with D2, F1, and C1: Time series of number of jobs completed by each node.

5.7. Experiment 7: Varying the Chunk Size

Chunk size, defined as the number of jobs dispatched per steal request to a worker, is a critical parameter within the Honeybee framework for optimising performance. An excessively large chunk size results in workers incurring substantial wait times for job reception rather than execution, while an overly small chunk size leads to too many steal requests, thereby increasing overheads.

The effect of chunk size is notably pronounced when job data are sizeable. In this paper, the job data are images, with an average size of 1.3 MB. The aim of this experiment is to calibrate chunk size to enhance performance. Initially, the experiment maintains a fixed chunk size for D2 and C1 at 5, iterating over various chunk sizes for F1 (see Figure 6b), which was identified as the most efficient worker in preceding experiments. Subsequently, the experiment is replicated with D2 (see Figure 6a) given its comparable efficacy to F1 in job completion.



(**a**) (**b**)

Figure 6. Experiment 7: Varying the chunk size. (**a**) Speedup gains for the D1 with varying chunk size for F1 and constant chunk size for nodes D2 and C1. (**b**) Speedup gain for the D1 with varying chunk size for D2 and constant chunk size for nodes F1 and C1.

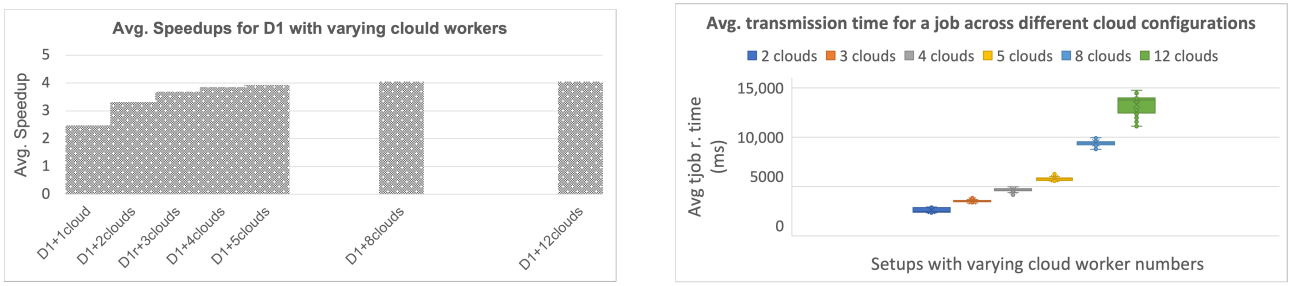
Increasing the chunk size beyond Honeybee's default of 5 yields performance gains for F1, with chunk sizes of 15 and 20 emerging as optimal for maximising speedups (Figure 6a). Nonetheless, augmenting the chunk size beyond these values appears to diminish returns. Conversely, for worker D2, the default chunk size gives the best speedup (Figure 6b). This phenomenon can be explained by examining the trade-off between communication time and job processing. An increase in chunk size results in proportionately longer job transmission times. Thus, an equilibrium must be struck between the latency of job arrival—a function of both the data volume per job and the chunk size—and the frequency of steal requests initiated by the worker. A diminutive chunk size prompts a higher rate of steal requests, while an excessive chunk size can lead to disproportionate waiting periods for job arrival relative to the job execution time.

This experiment highlights the need to configure chunk sizes for the different requirements for different tiers, as in this case, for fog and P2P edge. Fog servers, with greater processing power and bandwidth, can handle larger chunks effectively, while edge devices perform better with smaller chunks, compensating for their limited speed and capacity.

5.8. Experiment 8: Scalability

The results from previous experiments in this section show that adding more workers leads to increased speedups. However, it can be expected that speedups may eventually stabilise even if more and more nodes are added, due to the overheads of parallelisation and transmission costs. In this experiment, we investigate the scalability of adding more and more workers using instances of similar cloud workers. Starting with one cloud worker, the number of cloud workers is gradually increased till 12 to test the op-

timal number of workers for the highest speedups. Figure 7 illustrates the results of this experiment.



(**a**) (**b**)

Figure 7. Experiment 8: Results of scaling up cloud workers. (**a**) Speedups for delegator D1 with varying number of cloud workers (1 to 12). (**b**) Avg. job transmission time (ms) from delegator D1 to different setups of varying number of cloud workers.

From Figure 7a, it can be seen that the system performance increases with the addition of cloud workers. However, the returns plateau after five cloud servers. From that point onwards, the speedups remain stable even though more cloud workers are added. This can be explained in that having more workers increases the job distribution/transmission overheads, as illustrated in Figure 7b. With an increasing number of cloud workers, the average transmission time for a job increases. As long as this transmission overhead is offset by the benefit of increased computing resources, the system could continue to have increasing speedups. However, it is evident from Figure 7a that the point of diminishing returns for this particular setup occurs at five cloud servers. This result with cloud workers is also consistent with the experimental results for P2P mobile workers investigated in [6], which highlights that regardless of the type of worker (whether P2P edge/fog/cloud), there will be an optimal number of workers for a given job setup.

5.9. Experiment 9: Relative Node Capability

In this experiment, the role of the devices D1 and D2 have been switched, so that D2 is the delegator and D1 is a P2P edge worker. When D2 acts as the delegator, and works with other workers F1, C1, and D1, the jobs can be completed with an average speedup of 2.72, as shown in the node configuration *D2*+F1+C1+D1* in Figure 4b. The average task completion time for this experiment was 382,891 ms. On average, out of 1000 jobs, D2 completed around 400 jobs and is the dominant node, as shown in Figure 4a. It is followed by F1, which completed around 349 jobs. C1 completed 203 jobs and D1 is the weakest of all, completing only 48 jobs.

5.10. Experiment 10: Dynamic Changing of the Node Availability

In real-world scenarios, it can be expected that the node availability would change dynamically. In this set of experiments, the node availability was manually changed to investigate how well this would be handled by the system by examining the rate of job completion (how many jobs were completed by each node at a given time). Figure 8 shows the results of this set of experiments. Note that the jobs are not completely identical (although generally similar), and therefore, the rate of job completion is not a 100% indicator of how much 'work' a node would have done, as the processing of some jobs may be more or less complex than others.

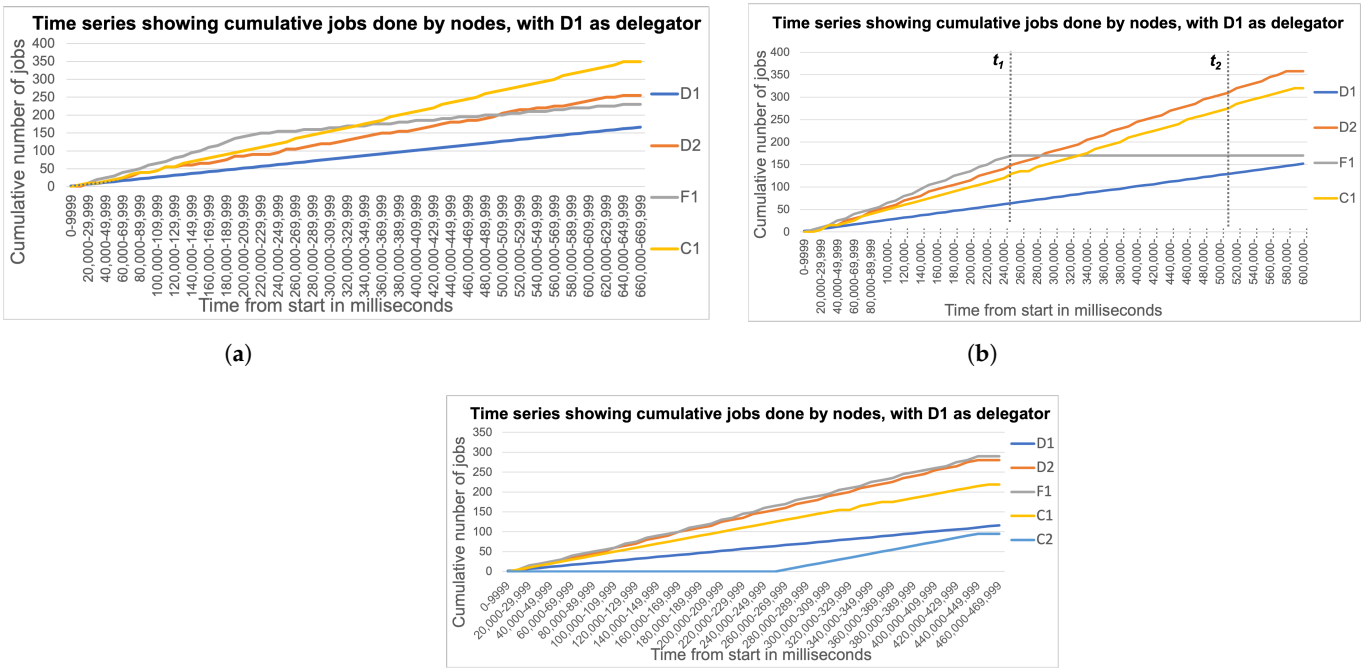


Figure 8. Experiment 10: Results for D1 working with D2, F1, and C1 under dynamic conditions. (**a**) Time series of cumulative jobs completed by the nodes: Slowing down F1. (**b**) Time series of cumulative jobs completed by the nodes: Disconnect F1. (**c**) Time series of cumulative jobs completed by the nodes: Add new cloud worker.

5.10.1. Degradation of Network (Pausing Node to Emulate Reduced Availability)

We first investigated the impact of a node's availability being decreased with no prior warning, by programmatically slowing down F1. This was achieved by pausing F1 for 20 s before working on the stolen jobs once it has completed 150 jobs. As can be seen from Figure 8a, the rate of job completion of F1 starts to decrease around 220,000 ms. This is evident in the change of the slope of F1 in Figure 8a. The other nodes' (D1, D2, C1) rate of job completion does not appear to have been significantly increased to compensate for F1's slow down in this instance, although closer inspection reveals a slight uptick of C1's rate of job completion. This may have been because although F1 was slowed down, it was still stealing jobs from the pool. Hence, it may not have given sufficient opportunity for the other nodes to steal more jobs at a higher rate. This experiment was performed five times, and the average speedup obtained was 5.873. This can be compared with the average speedup gained when all the nodes D1, D2, F2, C1 were working with no degradation in Experiment 6, as given in Section 5.6, where the average speedup was 7.582. Although the impact of the F1's decreased performance is evident from the reduced speedup, Honeybee's job scheduling and load balancing methods were still able to handle the unexpected reduction in F1's availability without needing to reconfigure the system parameters.

5.10.2. Degradation of Network (Random Loss of Node)

As the second experiment in this set, we tested Honeybee's fault tolerance mechanisms by programmatically disconnecting F1 halfway through the task. As can be seen from Figure 8b, F1's slope is flat from 250,000 ms onwards. The slopes of D1, D2, F1, and C1's curves at the point of disconnection (at t_1 in Figure 8b) are 2.498, 6.142, 7.03, and 5.195, respectively. After another 250,000 ms has elapsed, at 510,000 ms (at t_2 in Figure 8b), their slopes are 2.496, 6.333, 0, and 5.795, where both D2 and C1 have noticeable upticks in their rates of job completion. This can be explained by D2 and C1 stepping up to steal more jobs in the absence of F1. Unlike in the previous experiment shown in Figure 8a, here, the total

absence of F1 provided an opportunity for the other nodes to engage in increased stealing. Although the disconnection here was random, Honeybee's load-balancing mechanisms were still able to (1) ensure any jobs assigned to F1 were added back to the pool so that job execution can complete, and (2) seamlessly load-balance the jobs amongst the remaining nodes automatically without any interventions. This experiment was performed five times, and the average speedup obtained was 6.421, compared to the average speedup of 7.582, when all nodes were performing without any disconnections, as given in Section 5.6. Interestingly, the average speedup that was obtained here is actually higher than that of the 5.873 recorded in the previous experiment in Figure 8a. This suggests that it is more optimal to have a lower number of nodes with high availability than to have more nodes with decreased availability.

5.10.3. Adding a New Node Opportunistically

In the third and last experiment in this set, we investigated Honeybee's ability to incorporate more resource nodes opportunistically halfway through task execution, and whether this can provide benefits to overall performance. As can be seen in Figure 8c, the new cloud worker C2 was added at 260,000 ms. This experiment was performed five times, and the average speedup obtained was 8.32, compared to the average speedup of 7.582, when nodes D1, D2, F1, and C1 were performing without any disconnections, as given in Section 5.6. As can be seen from the job completion curves in Figure 8c, the addition of the new node C2 has not caused any disruption to the performance of the other nodes. From the increased speedup obtained here, it is evident that Honeybee can handle the ad hoc addition of new nodes, and thereby exploit opportunistic node encounters without requiring any reconfiguration or interventions.

6. Summary and Discussion

The speedups and battery savings (for delegator) are summarised in Table 5. As shown in Table 5, the proposed collaborative and proactive method achieves the highest speedups and battery savings when the nodes across edge, fog, and cloud are all working together.

RQ1: The impact of node collaboration across edge, fog, and cloud layers on overall task performance

Experiments 1 to 6, and 9 investigated the impact on performance and battery life of delegating computations to different combinations of P2P edge, fog, and cloud nodes. Experiment 1 demonstrates that transmission (and latency) times must be considered when offloading computations to other hosts. We also noted that with today's mobile devices, fog nodes or edge nodes can sometimes perform better even than cloud nodes (of course, depending on the configuration of the cloud nodes). Experiment 2 showed that two peer devices (D1 and D2) can complete jobs as much as 3.6 times faster, in particular, when a device offloads some computations to a more powerful device. Experiment 3 showed that a device (i.e., D1) offloading to a powerful fog node (F1) can complete jobs as much as 3.76 times faster, in particular, when a device offloads some computations to a more powerful device. Also, though D1 using F1 is faster than D1 using D2, it is noted that a higher battery usage is required when D1 uses F1 than if it uses F1 (due to the transmission power required to a fog node as opposed to a peer node). Experiment 4 showed that a device (i.e., D1) offloading to a cloud node (C1) can complete jobs as much as 2.48 times faster, in particular, when a device offloads some computations to a more powerful device. It is noted that a higher battery usage is required when D1 uses C1 than if it uses F1, as we would expect. Experiment 5 showed that a device (i.e., D1) offloading to a fog node (F1) and cloud node (C1) can achieve a speed up of 4.85. However, because less work is performed by D1, even though it is transmitting jobs to the other devices, its battery usage

decreased. Experiment 6 showed that a device (i.e., D1) offloading to a a peer node (D2), a fog node (F1), and a cloud node (C1) can achieve a speed up of 7.5. However, because even less work is performed by D1, even though it is transmitting jobs to the other devices, its battery usage decreased. Experiment 9 shows that the more powerful (relative to workers) a device is, the less advantage there is in delegating, e.g., it shows that since D2 is more powerful, the speedup from delegation is less than would be gained if D1 was the delegator.

Table 5. Experiment summary.

Exp.	Configuration: D1 as Delegator	Avg Battery Saving %	Avg Speedup
1	D1 monolithic	-	1.00
1	offload to D2	77.27	2.57
1	offload to F1	52.33	2.70
1	offload to C1	70.45	1.43
2	D1 + D2	71.59	3.59
3	D1 + F1	65.91	3.76
4	D1 + C1	50.00	2.48
5	D1 + F1 + C1	72.78	4.85
6	D1 + D2 + F1 + C1	80.68	7.58
7	Varying chunks for D2		
7	Chunk size = (D2 = 5, F1 = 5, C1 = 5)	80.68	7.58
7	Chunk size = (D2 = 20, F1 = 5, C1 = 5)	80.68	7.57
7	Chunk size = (D2 = 30, F1 = 5, C1 = 5)	79.55	7.57
7	Chunk size = (D2 = 40, F1 = 5, C1 = 5)	79.55	7.10
7	Varying chunks for F1		
7	Chunk size = (D2 = 5, F1 = 10, C1 = 5)	77.27	7.62
7	Chunk size = (D2 = 5, F1 = 15, C1 = 5)	75.06	7.81
7	Chunk size = (D2 = 5, F1 = 20, C1 = 5)	75.00	7.74
7	Chunk size = (D2 = 5, F1 = 25, C1 = 5)	79.55	7.50
7	Chunk size = (D2 = 5, F1 = 30, C1 = 5)	76.19	7.46
7	Chunk size = (D2 = 5, F1 = 35, C1 = 5)	77.27	7.51
7	Chunk size = (D2 = 5, F1 = 40, C1 = 5)	80.68	7.64
7	Chunk size = (D2 = 5, F1 = 45, C1 = 5)	79.55	7.59
7	Chunk size = (D2 = 5, F1 = 50, C1 = 5)	78.47	7.51
8	Adding Cloud Workers		
8	D1 + 2 Clouds	58.01	3.33
8	D1 + 3 Clouds	57.44	3.69
8	D1 + 4 Clouds	59.15	3.87
8	D1 + 5 Clouds	67.05	3.93
8	D1 + 8 Clouds	60.23	4.01
8	D1 + 12 Clouds	60.23	4.06
Exp.	Configuration: D2 as delegator	Avg. Battery Saving%	Avg Speedup
9	D2 monolithic	-	1.00
9	offload to D1	−162.51	0.17
9	offload to F1	37.71	0.85
9	offload to C1	17.80	0.56
9	D2 + D1 + F1 + C1	44.38	2.72

RQ2: Impact of task sharing configuration parameters in an edge, fog, cloud collaboration setup

Experiment 7 shows that delegation parameters such as chunk size have a significant impact on performance, increasing speedup by as much as 3–5% (e.g., in a 10 h job, that would be 30 min of savings), and that parameter optimisation should be tailored to the specific requirements of each tier.

RQ3: Dynamically adding/removing or slowing down worker nodes

Experiment 8 shows that there are limits to delegation—employing more cloud workers does not always mean improvements to speedup due to increasing transmission times

and the delegator's limitations in managing multiple workers. The speedup saturates at around 4.1 even with eight cloud nodes or more. There are also diminishing returns so that the gain in speedup with each additional node is less. Experiment 10 demonstrates the ability of the Honeybee framework to compensate and adapt to nodes disconnecting or slowing down across diverse tiers of edge, fog, and cloud—other nodes automatically do more jobs and do so in a way where the additional jobs are distributed fairly uniformly among the remaining nodes so that no particular node suddenly increases in job uptake (the increases in cumulative jobs across different devices over time remain fairly linear). Conversely, the addition of a new worker reduces load or completed jobs on all nodes fairly uniformly.

Overall, for Honeybee-based work stealing style offloading computations, we note that, with the current networking technology, offloading (especially to nearby nodes, i.e., fog or peer nodes) becomes a way to reduce battery consumption on the delegator, i.e., with $D1$ as delegator, we have: $batt(D1, D2, F1, C1) < batt(D1, F1, C1) < batt(D1, D2) < batt(D1, F1) < batt(D1, C1)$, where $batt$ is the battery usage on the delegator. Also, speedups are to be gained with offloading, despite additional transmissions required, and in particular, offloading to more devices helps more, unless the delegator device is relatively much more powerful than the device(s) being offloaded to, and despite transmissions to fog or cloud nodes, the more powerful the device being offloaded to, the greater the speedups attainable: with D1 as delegator, we have: $su(D1)[1] < su(F1)[2.37] < su(D1, C1)[2.48] < su(D2)[2.53] < su(D1, D2)[3.6] < su(D1, F1)[3.76] < su(D1, F1, C1)[4.85] < su(D1, D2, F1, C1)[7.5]$, where su denotes the speed up with the given configuration (also given in square brackets), and with the more powerful D2 as delegator, the speedup attained is no more than 2.8. While speedups are to be gained via offloading, there are limits to how much gain can be achieved and the advantage of using more workers, and there are diminishing returns with more nodes being used, with clear implications. For estimations of costs (with paying for more cloud nodes) versus benefits (from improved speedups), a mechanism to determine the optimal number of nodes (minimising the cost/benefit ratio) is required. It might not be necessary to scale a system beyond some threshold number of devices (in our experiments, less than half a dozen other worker nodes are adequate). Since delegation or offloading requires resources of the delegator, a multilevel delegation/offloading architecture might be required, e.g., a hierarchical or graph-based delegation where workers themselves delegate to others, recursively, would be needed to obtain further speedups when a large number of devices are actually available.

There is a need to tune parameters to obtain good performance relative to the resources/costs employed, e.g., chunk size, number of workers, and there is a need to compensate for disconnections, slow downs or node failures, as well as addition of new nodes, which our Honeybee algorithm does successfully.

7. Conclusions and Future Work

In this work, we evaluated the proposed system of collaborative edge-fog-cloud architecture exclusively on a real testbed with actual devices, ensuring practical applicability and eliminating reliance on simulations. Our experiments suggest that dynamic cooperative computations involving edge, fog, and cloud nodes are advantageous and should be facilitated as a "normal" device function. Honeybee has provided a means to enable such cooperative computations in a seamless way that is dynamically responsive to the availability and changing capacities of devices at run-time. As computers in our pockets and surroundings become more powerful and increase in number, the problem is, and increasingly so, not the lack of computational capabilities, but an increase in untapped

idle resources (e.g., the case of idle resources on desktops [42] and cloud nodes [43] easily extends to smartphones and other devices).

This paper has experimented with a range of configurations involving edge, fog, and cloud nodes, but there are many more to explore. The optimal configurations for computations are not easily arrived at analytically, but best effort, heuristic-based, functional validation approaches are probably more practical, e.g., a rule of thumb might be "offload only when surrounded by more capable devices which are not too far away", where the capabilities of devices can only be detected by performing some jobs for a short monitored period of time (as opposed to requiring explicit sharing of device information due to privacy reasons). Moreover, our theoretical analysis of speedup bounds currently assumes constant relative computational power indicators k_{e_i}, k_{f_j}, and k_{c_l}. However, in real-world scenarios, these values vary due to fluctuating network conditions and computational loads. Extending our model with probabilistic or time series-based approaches could capture these dynamics, enabling adaptive task offloading and configuration optimisation, and enhancing the framework's robustness in variable environments. While our experiments used D1 as a representative of modestly resourced edge devices, future work will involve testing the system on smaller and more constrained IoT platforms. This will further validate the scalability and adaptability of our approach across diverse edge environments. Hence, much further work remains.

Author Contributions: Conceptualisation, N.F.; methodology, N.F., S.W.L. and K.L.; software, N.F. and S.S.; validation, N.F. and S.S.; investigation, N.F., S.S., S.W.L. and K.L.; data curation, N.F. and S.S.; writing—original draft preparation, N.F., S.S., S.W.L. and K.L.; writing—review and editing, N.F., S.S., S.W.L. and K.L.; visualisation, N.F. and S.S.; supervision, N.F., S.W.L. and K.L.; project administration, N.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wang, S.; Zhang, X.; Zhang, Y.; Wang, L.; Yang, J.; Wang, W. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access* **2017**, *5*, 6757–6779. [CrossRef]
2. Ren, J.; Yu, G.; He, Y.; Li, G.Y. Collaborative cloud and edge computing for latency minimization. *IEEE Trans. Veh. Technol.* **2019**, *68*, 5031–5044. [CrossRef]
3. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
4. Cruz, P.; Achir, N.; Viana, A.C. On the Edge of the Deployment: A Survey on Multi-Access Edge Computing. *ACM Comput. Surv. (CSUR)* **2022**, *55*, 1–34. [CrossRef]
5. Mehrabi, M.; You, D.; Latzko, V.; Salah, H.; Reisslein, M.; Fitzek, F.H. Device-enhanced MEC: Multi-access edge computing (MEC) aided by end device computation and caching: A survey. *IEEE Access* **2019**, *7*, 166079–166108. [CrossRef]
6. Fernando, N.; Loke, S.W.; Rahayu, W. Computing with nearby mobile devices: A work sharing algorithm for mobile edge-clouds. *IEEE Trans. Cloud Comput.* **2016**, *7*, 329–343. [CrossRef]
7. Ortiz, G.; Zouai, M.; Kazar, O.; Garcia-de Prado, A.; Boubeta-Puig, J. Atmosphere: Context and situational-aware collaborative IoT architecture for edge-fog-cloud computing. *Comput. Stand. Interfaces* **2022**, *79*, 103550. [CrossRef]
8. Stojmenovic, I. Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In Proceedings of the 2014 Australasian Telecommunication Networks and Applications Conference (ATNAC), Melbourne, Australia, 26–28 November 2014; pp. 117–122.
9. Alharbi, H.A.; Aldossary, M. Energy-efficient edge-fog-cloud architecture for IoT-based smart agriculture environment. *IEEE Access* **2021**, *9*, 110480–110492. [CrossRef]
10. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012; pp. 13–16.

11. Fahim, A.; Mtibaa, A.; Harras, K.A. Making the case for computational offloading in mobile device clouds. In Proceedings of the 19th Annual International Conference on Mobile Computing & Networking, Miami, FL, USA, 30 September–4 October 2013; pp. 203–205.
12. Drolia, U.; Martins, R.; Tan, J.; Chheda, A.; Sanghavi, M.; Gandhi, R.; Narasimhan, P. The case for mobile edge-clouds. In Proceedings of the 2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing, Sorrento Peninsula, Italy, 18–21 December 2013; pp. 209–215.
13. Bellavista, P.; Chessa, S.; Foschini, L.; Gioia, L.; Girolami, M. Human-enabled edge computing: Exploiting the crowd as a dynamic extension of mobile edge computing. *IEEE Commun. Mag.* **2018**, *56*, 145–155. [CrossRef]
14. Zhang, W.; Flores, H.; Pan, H. Towards collaborative multi-device computing. In Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Athens, Greece, 19–23 March 2018; pp. 22–27.
15. Wang, B.; Wang, C.; Huang, W.; Song, Y.; Qin, X. A survey and taxonomy on task offloading for edge-cloud computing. *IEEE Access* **2020**, *8*, 186080–186101. [CrossRef]
16. Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J.P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.* **2019**, *98*, 289–330. [CrossRef]
17. Mach, P.; Becvar, Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
18. Taleb, T.; Samdanis, K.; Mada, B.; Flinck, H.; Dutta, S.; Sabella, D. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1657–1681. [CrossRef]
19. He, Y.; Ren, J.; Yu, G.; Cai, Y. D2D communications meet mobile edge computing for enhanced computation capacity in cellular networks. *IEEE Trans. Wirel. Commun.* **2019**, *18*, 1750–1763. [CrossRef]
20. Flores, H.; Sharma, R.; Ferreira, D.; Kostakos, V.; Manner, J.; Tarkoma, S.; Hui, P.; Li, Y. Social-aware hybrid mobile offloading. *Pervasive Mob. Comput.* **2017**, *36*, 25–43. [CrossRef]
21. Ranji, R.; Mansoor, A.M.; Sani, A.A. EEDOS: An energy-efficient and delay-aware offloading scheme based on device to device collaboration in mobile edge computing. *Telecommun. Syst.* **2020**, *73*, 171–182. [CrossRef]
22. Chen, X.; Zhang, J. When D2D meets cloud: Hybrid mobile task offloadings in fog computing. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–6.
23. Doolan, D.C.; Tabirca, S.; Yang, L.T. MMPI a message passing interface for the mobile environment. In Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia, Linz, Austria, 24–26 November 2008; pp. 317–321.
24. Marinelli, E. Hyrax: Cloud Computing on Mobile Devices Using MapReduce. Master's Thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA, 2009.
25. Miluzzo, E.; Cáceres, R.; Chen, Y.F. Vision: MClouds-computing on clouds of mobile devices. In Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services, Low Wood Bay, UK, 25 June 2012; pp. 9–14.
26. Hasan, R.; Hossain, M.; Khan, R. Aura: An incentive-driven ad-hoc IoT cloud framework for proximal mobile computation offloading. *Future Gener. Comput. Syst.* **2018**, *86*, 821–835. [CrossRef]
27. Firouzi, F.; Farahani, B.; Marinšek, A. The convergence and interplay of edge, fog, and cloud in the AI-driven Internet of Things (IoT). *Inf. Syst.* **2022**, *107*, 101840. [CrossRef]
28. Abbasi, M.; Mohammadi-Pasand, E.; Khosravi, M.R. Intelligent workload allocation in IoT–Fog–cloud architecture towards mobile edge computing. *Comput. Commun.* **2021**, *169*, 71–80. [CrossRef]
29. Bisht, J.; Vampugani, V.S. Load and cost-aware min-min workflow scheduling algorithm for heterogeneous resources in fog, cloud, and edge scenarios. *Int. J. Cloud Appl. Comput. (IJCAC)* **2022**, *12*, 1–20. [CrossRef]
30. Gupta, S.; Lozano, A. Computation-bandwidth trading for mobile edge computing. In Proceedings of the 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 11–14 January 2019; pp. 1–6.
31. Huang, M.; Liu, W.; Wang, T.; Liu, A.; Zhang, S. A cloud–MEC collaborative task offloading scheme with service orchestration. *IEEE Internet Things J.* **2019**, *7*, 5792–5805. [CrossRef]
32. Diao, X.; Zheng, J.; Wu, Y.; Cai, Y. Joint computing resource, power, and channel allocations for D2D-assisted and NOMA-based mobile edge computing. *IEEE Access* **2019**, *7*, 9243–9257. [CrossRef]
33. Du, J.; Zhao, L.; Feng, J.; Chu, X. Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. *IEEE Trans. Commun.* **2017**, *66*, 1594–1608. [CrossRef]
34. Zahoor, S.; Javaid, N.; Khan, A.; Ruqia, B.; Muhammad, F.J.; Zahid, M. A cloud-fog-based smart grid model for efficient resource utilization. In Proceedings of the 2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC), Limassol, Cyprus, 25–29 June 2018; pp. 1154–1160.
35. Kumar, M.S.; Karri, G.R. Eeoa: Cost and energy efficient task scheduling in a cloud-fog framework. *Sensors* **2023**, *23*, 2445. [CrossRef] [PubMed]

36. Alwateer, M.; Loke, S.W.; Fernando, N. Enabling drone services: Drone crowdsourcing and drone scripting. *IEEE Access* **2019**, *7*, 110035–110049. [CrossRef]
37. Loke, S.W.; Abkenar, A.B.; Fernando, N. Towards an Edge-Cloud Platform for Multirobot-Multihuman Cooperation in Urban IoT Ecosystems. In Proceedings of the WiP IEEE International Conference on Pervasive Computing and Communications (PerCom), Austin, TX, USA, 23–27 March 2020.
38. Nagesh, S.S.; Fernando, N.; Loke, S.W.; Neiat, A.; Pathirana, P.N. Opportunistic mobile crowd computing: Task-dependency based work-stealing. In Proceedings of the 28th Annual International Conference on Mobile Computing And Networking, Sydney, NSW, Australia, 17–21 October 2022; pp. 775–777.
39. Fernando, N.; Loke, S.W.; Rahayu, W. Mobile crowd computing with work stealing. In Proceedings of the 2012 15th International Conference on Network-Based Information Systems, Melbourne, Australia, 26–28 September 2012; pp. 660–665.
40. Blumofe, R.D.; Leiserson, C.E. Scheduling multithreaded computations by work stealing. *J. ACM (JACM)* **1999**, *46*, 720–748. [CrossRef]
41. Fernando, N.; Arora, C.; Loke, S.W.; Alam, L.; La Macchia, S.; Graesser, H. Towards human-centred crowd computing: Software for better use of computational resources. In Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), Melbourne, Australia, 14–20 May 2023; pp. 90–94.
42. Rosales, E.; Sotelo, G.; de la Vega, A.; Díaz, C.O.; Gómez, C.E.; Castro, H. Harvesting Idle CPU Resources for Desktop Grid Computing While Limiting the Slowdown Generated to End-Users. *Clust. Comput.* **2015**, *18*, 1331–1350. [CrossRef]
43. Zhang, B.; Dhuraibi, Y.A.; Rouvoy, R.; Paraiso, F.; Seinturier, L. CloudGC: Recycling Idle Virtual Machines in the Cloud. In Proceedings of the 2017 IEEE International Conference on Cloud Engineering (IC2E), Vancouver, BC, Canada, 4–7 April 2017; pp. 105–115. [CrossRef]