Resource Optimized Task Offloading in Delay Sensitive Edge Networks

Nasif Fahmid Prangon, Abdalaziz Sawwan, and Jie Wu
Center for Networked Computing, Temple University, USA
Emails: {nasifprangon, sawwan, jiewu}@temple.edu

***Abstract*—Task offloading is a popular approach in distributed systems to optimize resource usage by splitting the workload among devices in the network. This technique is effective in reducing the workload on individual devices, and helping achieve more efficient task execution by reducing execution time and conserving energy among devices. In this paper, we propose a novel method to evaluate task partitioning using a unique metric, effective processing rate (EPR) for edge networks with heterogenous devices. This unique approach focuses on simplifying task offloading based on the communication delay, task processing rate, and the energy available in each device. This approach also covers the impact of extending offloading beyond one-hop neighbors to two-hop neighbors and single-source versus multi-source offloading. We also discuss the willingness to help factor among one-hop and two-hop neighbors and its significance in task offloading performance. An effective strategy to meet these challenges is introduced and validated through theoretical analysis and extensive simulations.**

***Index Terms*—Communication delay, edge networks, energy efficiency, multi-access edge computing, resource optimization, task offloading.**

I. Introduction

Mobile phones, sensors, and communication technologies have become an integral part of modern life in the last decades, thus marking the beginning of the digital era. These devices operate in diverse environments and inherently produce massive amounts of data that are delay-sensitive and need to be processed rapidly [1]. Conventionally, these computations are performed on the cloud servers. However, transmission delay in data and consequently high energy consumption due to the distance between the source of data and the processing unit pose problems [2]. Edge computing addresses this by bringing computational resources closer to users [3].

Multi-access edge computing (MEC) is an extension of edge computing that utilizes operator resources for efficient data processing through wireless transmission [4]. MEC lowers latency and increases energy efficiency by decentralizing the processing. However, decentralized task offloading may still lead to inefficiency, especially when the applications require real-time processing.

This research was supported in part by NSF grant SaTC 2310298 for the first author and NSF grant CNS 2107014 for the second author.

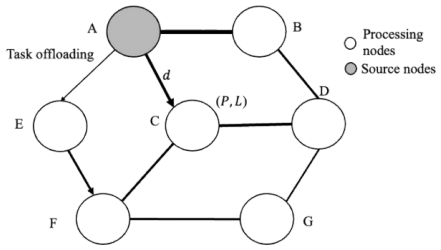


Fig. 1: Task offloading from single source A to neighbor processing nodes C, E, and F with delay d, processing rate P, and lifetime L.

Edge devices suffer from a host of constraints in the form of computation power, communication delay, and battery life, while neighboring resources may remain underutilized. Besides that, edge resources alone might not be enough to meet strict latency requirements. Based on this, we develop a task offloading mechanism between neighboring devices to allow for load sharing and enhance both energy and execution efficiency. Figure 1 shows such an offloading scenario from the source to other neighbors. Node A offloads tasks to neighbors pointed by the arrows with delays indicated by the thickness of the line and a lifetime defined as the time the nodes remain capable of processing tasks. The contribution of this work is a new offloading strategy, which tries to achieve optimal task processing time in an edge network. The novelty is essentially the dynamic distribution algorithm across multi-hop neighbors in order to reduce the processing time while considering energy constraints and communication delay.

Task offloading involves distributing work among any k-hop neighbors (all nodes in the network) based on the effective processing rate (EPR), including the node's delay, processing capability, and energy. It is quite simple in the case of one-hop neighbors, but in the case of two-hop neighbors, there are a number of paths with possibilities of energy constraints. Distribution will need to be effective in case any node is incapable of handling its own fraction of the task to ensure all nodes finish at the earliest possible time.

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

II. Background and Related Works

Task offloading in edge computing has recently attracted much attention since many applications call for rapid data processing [5]. Traditional brute-force methods are impracticable to implement in large-scale networks since the complexity is exponential [6]. Therefore, recent research has focused on algorithms that reduce the search space to improve the efficiency, such as branch and bound, which, though improving computation efficiency, still scale poorly when the network size increases [7]. This motivated the investigation of certain heuristic approaches, such as a genetic algorithm, which balances solution optimality and computational burden via an iterative refinement of offloading strategies following the principles of natural evolution [8]. Despite their benefits, these methods generally require parameter fine-tuning and may not always reach global optimality. This calls for faster and simpler approaches.

Task processing has commonly been modeled deterministically, assuming that tasks start processing instantly when they arrive at the edge server [8], [9]. Works such as [9], [10] have developed adaptive offloading strategies that can achieve maximum revenue with ensured service quality by dynamically adjusting decisions according to network conditions and requirements of services. Other works, like [8], have developed a joint optimization of offloading tasks and resource allocation by framing it as a mixed-integer nonlinear programming problem to minimize energy consumption [11]. These methods, although effective, remain NP-hard and hence very computationally expensive

Recent works have targeted efficiency improvement in this direction. For example, [12] has presented an online approach toward jointly optimum network selection and job offloading in multi-dimensional resource-constrained MEC networks, which is near-optimal. Another work [13] modeled the problem of offloading as a generalized allocation model with constraints about hop counts and wireless communication ranges.

Our proposed approach is simpler and faster, enabling quick decisions based on current network states and available resources, avoiding the processing overhead typical of more complex optimization methods. While adaptive approaches like [14] adjust to network conditions and service types, our method puts higher emphasis on speed and simplicity, making it ideal for cases where decision-making needs to be quick, at the possible cost of ignoring fine optimizations brought by more advanced methods in particular instances.

III. System Model and Problem Formulation

A. System Model

We model the edge network as a connected graph $G = (N, E)$ where every node may act either as a source node (SN) or a processing node (PN). Each node presents a communication delay d_{ij} with its neighbors, modeled by the edges connecting the nodes. The network is composed of nodes in set N, facilitating task processing through direct and multi-hop connections (see Table I for symbols and notations).

TABLE I: Symbols and Notations

Symbol	Description
N	Set of nodes in the network
G	Graph representing the network
T_{total}	Total tasks from the source node
T_i	Number of tasks allocated to node i
P_i	Processing rate of node i (tasks per unit time)
d_{ij}	Communication delay for connection (i, j)
dis_i	Shortest path delay from the source to node i
L_i	Lifetime of node i
$t_{\text{rise},i}$	Time (checkpoint) when node i starts processing
$t_{\text{drop},i}$	Time (checkpoint) when node i stops processing
$\{t_1, t_2, \ldots, t_{2\lvert N\rvert}\}$	Sorted precomputed checkpoints
$\mathcal{R}_{\text{eff},j}$	EPR of active nodes within interval $[t_j, t_{j+1})$

Nodes experience two key events:

- *Rise Event*: A node i starts processing tasks at its shortest path delay dis_i from source.
- *Drop Event*: A node i stops processing tasks after being active through its lifetime L_i, i.e., upon energy depletion, assuming a fixed energy consumption rate.

Checkpoints are precomputed as the union of rise and drop events for all nodes. These events define intervals during which tasks are distributed iteratively.

The EPR of an interval represents the total processing capacity available through that interval. It dynamically adjusts as nodes join or leave the system, ensuring efficient task distribution. It is defined for an interval $[t_j, t_{j+1})$ as:

$$\mathcal{R}_{\text{eff},j} = \sum_{i \in N_j} P_i$$

where N_j is the set of active nodes in the interval $[t_j, t_{j+1})$, and P_i is the processing rate of each active node i within that interval. Note that we refer to an interval by the index of its first checkpoint.

B. Problem Formulation

The objective is to develop a task allocation strategy that minimizes the total completion time for the total required task. The optimization problem is:

$$\text{minimize} \quad \max_{i \in N} \left(\frac{T_i}{P_i} + dis_i \right)$$

where T_i is the number of tasks allocated to node i, P_i is the processing rate of node i, and dis_i is the shortest path delay from the source node to node i. The objective is to minimize the task completion time, or makespan.

Algorithm 1 Precomputing Checkpoints

Require: Directed graph representing the network $G = (N, E)$, delays d_{ij}, lifetimes L_i
Ensure: Sorted list of checkpoints $\{t_1, t_2, \ldots, t_{2|N|}\}$
1: Run Dijkstra's algorithm to compute shortest path delays dis_i for all nodes
2: **for** each node $i \in N$ **do**
3: $t_{\text{rise},i} \leftarrow dis_i$
4: $t_{\text{drop},i} \leftarrow t_{\text{rise},i} + L_i$
5: Sort the list of events $\{t_{\text{rise},i}, t_{\text{drop},i} | \forall i \in N\}$ in ascending order to form $\{t_1, t_2, \ldots, t_{2|N|}\}$
6: **return** Sorted list of checkpoints $\{t_1, t_2, \ldots, t_{2|N|}\}$

The task allocation is subject to the following constraints:

- The total tasks distributed to all nodes must equal the total tasks from the source:

$$\sum_{i \in N} T_i = T_{\text{total}}$$

- Tasks allocated to any node must be non-negative:

$$T_i \geq 0, \quad \forall i \in N$$

Task distribution is performed iteratively over intervals defined by precomputed checkpoints, which represent the rise and drop events of nodes. At each interval, the EPR for the whole system $\mathcal{R}_{\text{eff},j}$ is updated dynamically based on the active nodes. This ensures efficient task allocation while adapting to dynamic network conditions.

IV. Algorithm Overview

A. Single-Source Resource Optimized Task Allocation (SSROTA)

The SSROTA algorithm assigns work between processing nodes in an ideal manner such that all processing nodes will have completed processing its part at the same time, minimizing overall run-time. Unlike conventional static scheduling frameworks, in the proposed model, reassigning work distribution at precaculated checkpoint times is supported, and efficient use of processing capacities can be achieved.

Task distribution Algorithm 2 is guided by a set of precomputed checkpoints, where each node has a rise event and a drop event calculated through Algorithm 1. A rise event occurs when a node becomes active and starts processing at time $t_{\text{rise},i}$, which is determined by computing the shortest path delay from source node. A drop event occurs when a node stops processing at time $t_{\text{drop},i} = t_{\text{rise},i} + L_i$. These events are collected and sorted into a global list of checkpoints, denoted as $\{t_1, t_2, \ldots, t_{2|N|}\}$, which partitions the task execution timeline into multiple intervals.

Algorithm 2 Task Distribution with Checkpoints

Require: Precomputed checkpoints $\{t_1, t_2, \ldots, t_{2|N|}\}$, total tasks T_{total}, $N_j \forall$ intervals $[t_j, t_{j+1})$
Ensure: Tasks distributed so all nodes finish at the same time
1: Initialize $j \leftarrow 1$, $T_{\text{remaining}} \leftarrow T_{\text{total}}$
2: **while** $T_{\text{remaining}} > 0$ **do**
3: Evaluate $\mathcal{R}_{\text{eff},j} = \sum_{i \in N_j} P_i$ // Effective processing rate
4: $\Delta t \leftarrow \min\left(t_{j+1} - t_j, \frac{T_{\text{remaining}}}{\mathcal{R}_{\text{eff},j}}\right)$ // Compute required processing time
5: $T_{[t_j, t_j + \Delta t]} = \mathcal{R}_{\text{eff},j} \times \Delta t$ // Tasks completed in interval
6: $T_{\text{remaining}} \leftarrow T_{\text{remaining}} - T_{[t_j, t_j + \Delta t]}$ // Update remaining tasks
7: $j \leftarrow j + 1$ // Move to the next interval
8: **return** $t_{j-1} + \Delta t$ // Total completion time

The task execution process iterates through the sorted checkpoints, distributing tasks dynamically in each interval. An interval $[t_j, t_{j+1})$ is the time span between two consecutive checkpoints, where active nodes N_j process tasks at an effective rate $\mathcal{R}_{\text{eff},j} = \sum_{i \in N_j} P_i$. The duration of each interval is computed as $\Delta t = t_{j+1} - t_j$, during which tasks are processed at an effective rate $\mathcal{R}_{\text{eff},j}$. The number of tasks completed within an interval is given by $\Delta t \cdot \mathcal{R}_{\text{eff},j}$, and these tasks are proportionally distributed among active nodes based on their processing rates. The remaining task count is updated as: $T_{\text{remaining}} - (\Delta t \cdot \mathcal{R}_{\text{eff},j})$.

At each checkpoint, $\mathcal{R}_{\text{eff},j}$ is updated. When a rise event occurs $t_{\text{rise},i}$, the processing rate of the newly activated node i is added, such that $\mathcal{R}_{\text{eff},j} \leftarrow \mathcal{R}_{\text{eff},j} + P_i$. Conversely, at a drop event $t_{\text{drop},i}$, when a node i exhausts its lifetime and stops processing, its processing rate is removed, leading to $\mathcal{R}_{\text{eff},j} \leftarrow \mathcal{R}_{\text{eff},j} - P_i$. This dynamic adjustment ensures that task allocation is continuously optimized until all tasks are completed, at which point the algorithm terminates.

By dynamically reassigning jobs at each checkpoint, SSROTA augments preallocated job distribution and enables workload adaptability, as shown in Figure 2. The checkpoints include rise and drop events where, first E is added followed by C, and F. Before the task could be finished, E dies. Also, the allocated task finishes before reaching B with a higher delay. Active nodes alone execute tasks, with workload distribution continuously adjusted based on the processing capacities of the available nodes. At each interval, tasks are allocated in proportion to the processing rates of active nodes, ensuring that computational resources are utilized efficiently. This approach ensures that all active nodes

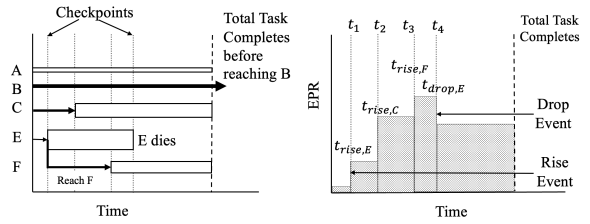


Fig. 2: Task execution process with checkpoints.

complete their workloads around the same time, minimizing overall execution time and maximizing resource utilization using the EPR metric.

B. Task Allocation for Multi-Source

The algorithm extends to multi-source distribution of work over shared networked machines. We assume exclusive node sharing, meaning that a node can only process tasks from one source at a time. Unlike single-source, multi-source work distribution introduces new locking constraints at a node, where an occupied node cannot receive work from a new source while its already processing task received from prior sources.

The algorithm accomplishes this through dynamically altering $\mathcal{R}_{\text{eff},j}$ and locking nodes once they are assigned jobs until they finish it depicted by waiting in Figure 3. The waiting duration of a locked node is added in delay calculations for future intervals, and thus, each source efficiently utilizes its accessible resources.

For instance, in Figure 3, source D offloads tasks to nodes B, C, G, and F, among which C and F are also accessible to source A. Since source A have already allocated tasks to these nodes, F remains locked for source D until the tasks from A is completed while the task for C arrives after it is already released. As a result, source D is forced to allocate more tasks to other available nodes, such as B and G, based on processing capacity. This node-locking mechanism prevents an optimal global allocation but ensures that each source still utilizes its available resources in the most efficient way.

C. Algorithm Extension for Willingness to Help

As an extension to the algorithms, we discuss the impact of willingness to help among neighbors. As we traverse deeper into the network, despite choosing the shortest path, the delay between the SN and PN increases. Additionally, in multi-hop networks, nodes are more likely to share multiple parents or sources, which further impacts their processing efficiency [15]. These factors influence a node's *willingness to help*, which is modeled as a dynamic factor λ_i. The EPR of the network, incorporating willingness, is defined as:

$$\mathcal{R}_{\text{eff},j} = \sum_{i \in N_j} \lambda_i \cdot P_i$$

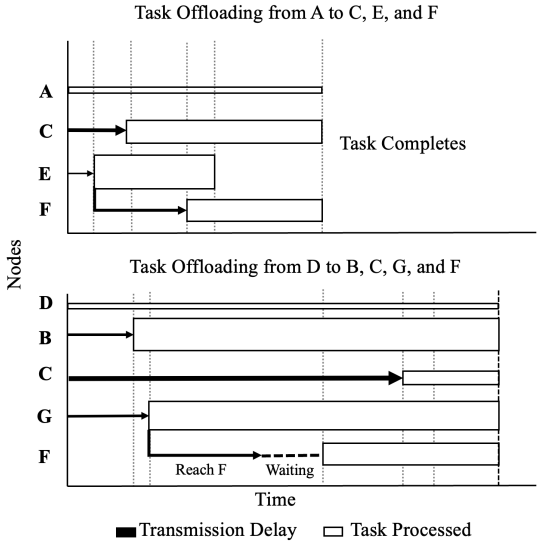


Fig. 3: Multi-source task offloading with shared processing nodes.

where $\lambda_i \in [0, 1]$ represents the willingness of node i to assist in task processing.

D. Algorithm Analysis

Theorem 1. *The algorithm converges to an optimal task allocation using precomputed checkpoints.*

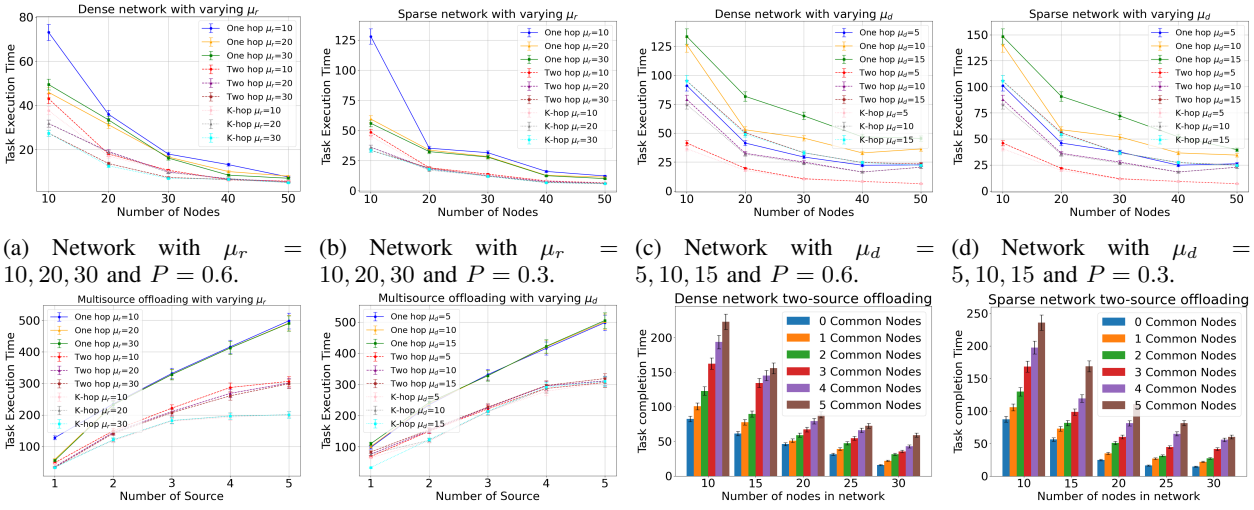
Proof: Suppose, for contradiction, that the algorithm does not yield an optimal task allocation. This would imply the existence of another allocation, denoted as A', that results in a lower completion time or a more efficient utilization of available nodes compared to the allocation A produced by our algorithm.

Our algorithm ensures that all *terminally-active nodes* (nodes that remain active until the last processing interval) finish at the same time while maximizing the utilization of all available nodes before reaching the final processing time where task completes.

From Algorithm 1, we precompute all checkpoints by determining the shortest path delays using Dijkstra's algorithm. These checkpoints partition the timeline into intervals where the effective processing rate $\mathcal{R}_{\text{eff},j}$ dynamically adjusts as nodes join or leave the computation.

In Algorithm 2, tasks are distributed proportionally within each interval. The effective processing rate $\mathcal{R}_{\text{eff},j}$ determines the number of tasks completed in that interval, ensuring that tasks are allocated efficiently and fairly among active nodes within the interval.

Now, consider any alternative allocation A' where at least one terminally-active node, denoted as n, is assigned a higher task load than in our allocation A. The *minimum* amount of time required for n to process its assigned task T'_n is given by $t' = dis_n + \frac{T'_n}{P_n}$, where



(a) Network with $\mu_r = 10, 20, 30$ and $P = 0.6$.
(b) Network with $\mu_r = 10, 20, 30$ and $P = 0.3$.
(c) Network with $\mu_d = 5, 10, 15$ and $P = 0.6$.
(d) Network with $\mu_d = 5, 10, 15$ and $P = 0.3$.
(e) Network with $\mu_r = 10, 20, 30$.
(f) Network with $\mu_d = 5, 10, 15$.
(g) Dense network with $P = 0.6$.
(h) Sparse network with $P = 0.3$.

Fig. 4: Task offloading performance with a fixed task size $T = 50$, varying node count, mean delay μ_d, and mean computation power μ_r (a)–(d). Multi-source task offloading with $T = 50$, $P = 0.5$, and varying μ_r, μ_d (e) and (f). Two-source offloading vs. common nodes and increasing network size (g) and (h).

dis_n is the shortest-path delay for n from source, and P_n is its fixed processing rate.

However, in our algorithm, the allocation ensures that all terminally-active nodes complete at the optimal finishing time , where A is strictly less than A', meaning that our approach allows n (and all other terminally-active nodes) to finish *earlier* than it would under A'.

Since any alternative allocation A' results in at least one terminally-active node finishing later than our optimal finishing time, this contradicts the assumption that A' is more optimal. Thus, our algorithm provides the optimal solution. □

E. Complexity Analysis

Precomputing Checkpoints: Dijkstra's algorithm for dense graphs has a time complexity of $O(|N| + |E| \log |N|)$ when using a min-priority queue. For sparse graphs, it reduces to $O(|N| \log |N|)$. Computing the rise and drop times for all nodes takes $O(|N|)$. Combining and sorting the events list requires $O(|N| \log |N|)$. Thus, the overall complexity remains dominated by the shortest path algorithm, which is $O(|N| + |E| \log |N|)$.

Task Distribution with Checkpoints: The algorithm iterates through checkpoints, requiring $O(k)$, where k is the number of events (at most $2|N|$). For each interval, computing the tasks and updating the EPR involves $O(|N|)$ operations. The overall complexity is $O(|N|^2)$.

V. Experimental Simulation and Evaluation

In this section, we evaualte the practical implications of our algorithm, which is applicable for a network of any size. The primary purpose of the evaluation is to see the impact of selecting only one-hop neighbors and two-hop neighbors and extending to k hop neighbors comprising all the nodes. We generated random graphs for both dense and sparse configurations using NetworkX [16]. Each node was sampled from uniform distributions for processing rate *r*, energy *e*, and delay *d*. A 95% confidence interval was calculated for each property to ensure variability. Edges were generated using the Erdős-Rényi model, with edge probability $P \in [0, 1]$.

A. Single-Source Offloading Analysis

Figure 4a shows task execution times in a dense graph ($T = 50$, $P = 0.6$, $\mu_d = 5$, $\sigma_d = 1$, $\sigma_r = 5$). Execution times decrease as node count increases (10–50) across resource rates ($\mu_r = 10, 20, 30$) for one-hop, two-hop, and k-hop types, demonstrating scalability. For $\mu_r = 10$, one-hop time drops from 73.16 to 7.58 as nodes increase from 10 to 50.

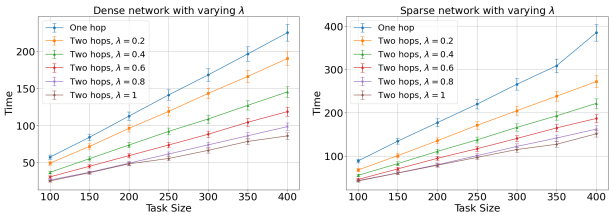
Figure 4b depicts similar trends in a sparse graph (density 0.3), confirming the algorithm's adaptability in reducing execution time across different node counts and resource rates.

Figure 4c compares execution times for varying delays μ_d, showing that increasing node count reduces times across all delay levels, with multi-hop strategies outperforming one-hop, especially in dense graphs.

Figure 4d reflects consistent results in sparse graphs, reinforcing the algorithm's effectiveness in minimizing execution time for both dense and sparse configurations.

B. Multi-Source Offloading Analysis

Figure 4e and 4f show task execution times for multi-source offloading ($T = 50$, $P = 0.5$) with varying mean



(a) Dense network with $P = 0.6$ (b) Sparse network with $P = 0.3$

Fig. 5: Task offloading for one-hop and two-hop with $\lambda = 0.2, 0.4, 0.6, 0.8, 1$.

delays μ_d and processing rates μ_r. Increasing sources and delays $\mu_d = 5, 10, 15$ lead to higher execution times, especially for one-hop tasks, while k-hop tasks perform better. Similar trends can be seen with resource rates $\mu_r = 10, 20, 30$, where k-hop tasks minimize execution times as resources improve. This rise in total time is due to more conflicts between sources where they contend for resources that can process their assigned tasks.

Figure 4g compares task completion in a dense network with two-source offloading across varying node counts (10-30) and 0-5 common nodes. Fewer common nodes lead to lower completion times, but as network size grows, execution times decrease due to better load distribution. Fig. 4h shows similar trends in sparse networks, where higher common nodes increase completion times, but larger networks improve load handling by adding more processing nodes.

C. Willingness to Help Analysis

In resource-optimized task allocation, λ is crucial, especially as tasks move further from the source. Figure 5a and 5b show that in both dense and sparse networks, decreasing λ from 1.0 to 0.0 increases execution times for two-hop offloading outperforming with $\lambda = 0.2$ performing close to one-hop solution . Sparse networks show a similar trend, though with higher overall times due to fewer nodes. Higher λ improves load balancing and execution times, particularly in dense networks with more neighbors.

VI. Conclusion

In this paper, we presented a resource-aware task offloading technique for edge networks using the EPR metric to simplify task partitioning that considers communication delay, processing rate, and node lifetime. The tasks were optimally partitioned among intervals between checkpoints. We extended the offloading from one-hop to two-hop neighbors and compared these with the k-hop scenario. Simulation shows that our algorithm reduces the execution time of tasks by optimizing the task offloading, making it efficient and scalable for dense and sparse networks. The results indicate that nodes that lie farther away from the source may not be effective due to the delay overhead. Future work could be extended to the investigation of inter-edge collaboration, where multiple edge networks coordinate to optimize resource utilization, and adaptive models for the willingness-to-help factor can further enhance task offloading strategy efficiency.

References

[1] A. Lakhan, D. K. Sajnani, M. Tahir, M. Aamir, and R. Lodhi, "Delay sensitive application partitioning and task scheduling in mobile edge cloud prototyping," in *2nd International Conference on 5G for Ubiquitous Connectivity: 5GU 2018*. Springer, 2020, pp. 59–80.

[2] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2017.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[4] A. Filali, A. Abouaomar, S. Cherkaoui, A. Kobbane, and M. Guizani, "Multi-access edge computing: A survey," *IEEE Access*, vol. 8, pp. 197 017–197 046, 2020.

[5] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.

[6] R. Lin, Z. Zhou, S. Luo, Y. Xiao, X. Wang, S. Wang, and M. Zukerman, "Distributed optimization for computation offloading in edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 12, pp. 8179–8194, 2020.

[7] P. Zhou, K. Shen, N. Kumar, Y. Zhang, M. M. Hassan, and K. Hwang, "Communication-efficient offloading for mobile-edge computing in 5g heterogeneous networks," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 237–10 247, 2020.

[8] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2018.

[9] A. Samanta and Z. Chang, "Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3864–3872, 2019.

[10] Y. Chiang, C.-H. Hsu, G.-H. Chen, and H.-Y. Wei, "Deep q-learning-based dynamic network slicing and task offloading in edge network," *IEEE Transactions on Network and Service Management*, vol. 20, no. 1, pp. 369–384, 2022.

[11] X. Li, L. Zhao, K. Yu, M. Aloqaily, and Y. Jararweh, "A cooperative resource allocation model for iot applications in mobile edge computing," *Computer Communications*, vol. 173, pp. 183–191, 2021.

[12] X. Qi, H. Xu, Z. Ma, and S. Chen, "Joint network selection and task offloading in mobile edge computing," in *Proceedings of the IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 475–482.

[13] C. Chen, Y. Zeng, H. Li, Y. Liu, and S. Wan, "A multihop task offloading decision model in mec-enabled internet of vehicles," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3215–3230, 2022.

[14] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2020.

[15] J.-C. Kuo, W. Liao, and T.-C. Hou, "Impact of node density on throughput and delay scaling in multi-hop wireless networks," *IEEE Transactions on Wireless Communications*, vol. 8, no. 10, pp. 5103–5111, 2009.

[16] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Laboratory, Los Alamos, NM, USA, Tech. Rep., 2008.