Opportunistic Division and Allocation of Machine Learning Task for WSN

Eri Hosonuma*, Nobuyuki Tanaka†, Takuma Yamazaki‡, Akihito Taya*,
Yuuki Nishiyama§, Kaoru Sezaki§*, Takumi Miyoshi†‡*, and Taku Yamazaki†‡
* Institute of Industrial Science, The University of Tokyo, Tokyo, Japan
† Collage of Systems Engineering and Science, Shibaura Institute of Technology, Saitama, Japan
‡ Graduate School of Engineering and Science, Shibaura Institute of Technology, Saitama, Japan
§ Center for Spatial Information Science, The University of Tokyo, Chiba, Japan
Email: hosonuma@mcl.iis.u-tokyo.ac.jp, {bp19124, mf22132, taku, miyoshi}@shibaura-it.ac.jp,
{taya-a, yuukin, sezaki}@iis.u-tokyo.ac.jp

***Abstract*—Machine learning (ML)-applied sensing systems are widely deployed in real environments in the research and development of wireless sensor networks (WSNs). However, in these systems, the central server must deal with the large amounts of sensing data and high processing costs to execute ML tasks. To deal with these issues, in-network processing methods of ML tasks have been proposed for WSNs. However, their main focus is to decide the division points and allocation strategy of ML tasks, and therefore they do not consider the routing algorithm in distributed environments. This paper proposes an opportunistic division and allocation method of ML tasks in distributed WSN environments that does not rely on a specific path. In the proposed method, each node autonomously makes a forwarding decision based on the remaining computational resource and hop count to distribute the computational load while considering the number of relays. A simulation was performed, and it revealed that the proposed method can appropriately allocate the computational processes of ML tasks and distribute them to WSN nodes.**

***Index Terms*—wireless sensor network, opportunistic routing, machine learning**

I. Introduction

Machine learning (ML)-applied sensing systems for monitoring conditions [1], [2] and detecting targets [3], [4] are widely deployed in the research and development of wireless sensor networks (WSNs) [5], [6]. In these systems, sensing data are aggregated by a sink node, which then sends the data to a central server for executing ML tasks. However, the central server must deal with large amounts of sensing data and a high processing cost for ML tasks.

To alleviate these burdens, a distributed processing method that divides and allocates the computational processes contained in an ML task to wireless sensor networks (WSN) nodes was proposed [7]. Additionally, an allocation strategy of dividing computational processes of multiple ML tasks was also proposed [8]. However, these approaches mainly focused on deciding division points of a computational process in an ML task and allocating them to WSN nodes in known topologies without considering the routing algorithm in distributed environments.

This work was supported by JSPS KAKENHI Grant Number JP21H04886 and NICT, Japan (05601).

This paper proposes an opportunistic division and allocation method of ML tasks using opportunistic routing (OR) [9] that autonomously makes forwarding decisions at the packet level without relying on a specific path.

II. Related Work

A distributed processing method of an ML task realizes the division and allocation of a computational process by using WSN nodes [7]. When a node generates an ML processing request, its computational process is divided, and the divided processes are then allocated to relay nodes. However, this study focused on analyzing the behavior on a simple straight-line topology. Therefore, an allocation algorithm in distributed environments was not considered. Another distributed processing method realizes a division and load-balanced allocation strategy of multiple ML tasks [8]. This method divides the computational processes of multiple ML tasks and allocates them to relay nodes based on the hop count and number of routes stored in each node to distribute the computational load among the WSN nodes. However, this method does not consider the routing and allocation procedure in distributed environments because it assumes that routes and metrics are obtained in advance.

Backoff-based opportunistic routing (OR) [9] can flexibly establish a flexible forwarding path based on cost information, such as a hop count, in distributed environments. When a node receives a data packet, it calculates the backoff time using cost information to the destination and waits for a timer to expire. Upon receiving the same packet from other nodes, it cancels waiting and discards the kept packet. Otherwise, it forwards the kept packet as a forwarder after the timer expires. By repeating this procedure, backoff-based OR can autonomously make forwarding decisions without depending on a specific path. Therefore, by exploiting the backoff-based OR, a process division and allocation mechanism of ML tasks can be realized in distributed environments.

III. Proposed Method

This paper proposes a process division and load-balanced allocation method of ML tasks with an adaptive node selection

considering the computational resources of WSN nodes. Fig. 1 illustrates an overview of the proposed method. The proposed method adopts a backoff-based OR that can make a forwarding decision based on a backoff time calculated using both hop counts to a sink node and the remaining computational resources of potential forwarders. When a potential forwarder is selected as a forwarder, it calculates its computation effort based on its remaining computational resource and starts to execute the allocated process of the ML task. After finishing the process, the forwarder sends the processing result as data to the next forwarder. The proposed method repeats this procedure until the processing result reaches the sink node.

Discovery phase: When a source node S generates an ML task, it starts flooding a request packet to discover the sink node D if S does not have a cost information h_{SD} that denotes the S's hop count to D. Each receiver r records cost information $h_{r\mathrm{S}}$ in its cost table and forwards the received packet. When D receives the request packet, it sends a reply packet to S. Note that the reply packet is forwarded based on a backoff time calculated by each receiver r as in [9]. Each receiver of the reply packet r records and updates $h_{r\mathrm{D}}$ by using the traversed hop count in the packet header. When S receives the reply packet, it calculates target computational cost $\hat{p}_{\mathrm{S}} = P_{\mathrm{S}}/(h_{\mathrm{SD}}+1)$, where P_{S} is the total computational cost generated by S, and then it moves to the forwarding phase.

Forwarding phase: Fig. 2 illustrates the procedure of the forwarding phase. A forwarder i including the source node calculates its computational effort p_i as described in the following equation, Eq. (1), and starts to execute a part of the ML task.

$$p_i = \begin{cases} \hat{p}_{\mathrm{S}} & (\hat{p}_{\mathrm{S}} \leq Q_i - q_i) \\ Q_i - q_i & (\hat{p}_S > Q_i - q_i) \end{cases}, \qquad (1)$$

where Q_i $(0 < Q_i)$ denotes i's entire computational resource and q_i $(0 < q_i \leq Q_i)$ denotes i's total on-going computational load. When i finishes the computation, it writes $\hat{p}_{\mathrm{S}}$, hop count $h_{i\mathrm{D}}$, the remaining process, the addresses of S, D, and i, and a sequence number in the header. Then, it broadcasts the processing result as data.

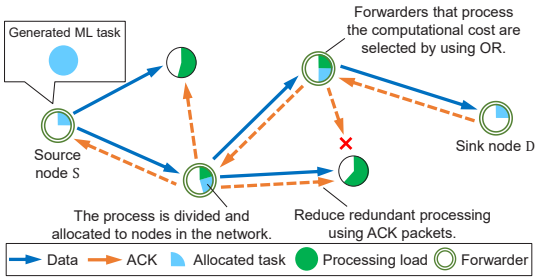


Fig. 1. Overview of the proposed method.

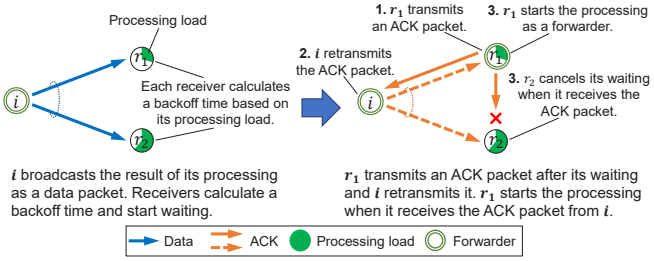


Fig. 2. Procedures of the forwarding phase in the proposed method.

Each receiver r calculates a backoff time b_r based on $h_{r\mathrm{D}}$ in its cost table and $h_{i\mathrm{D}}$ in the header of the received packet, and starts waiting as a potential forwarder. Note that details on the backoff time calculation are described below. When r finishes waiting, it broadcasts an ACK packet with i's address as a previous-hop forwarder. When i receives the ACK packet, it rebroadcasts the received ACK packet with r's address for selecting r as the next-hop forwarder. If r receives the ACK packet from i, it becomes a forwarder and starts its computation as described above. When a potential forwarder in waiting receives an ACK packet in which it is not selected itself as the previous- or next-hop forwarder, it stops waiting to suppress redundant computations. To repeat the above procedure, the proposed method can autonomously make forwarding and allocating decisions of the divided processes in an ML task.

Backoff time calculation: The proposed method adopts two backoff time calculation algorithms: the hop count-based and computation load-based algorithms, to realize an appropriate forwarder selection.

In the hop count-based calculation proposed in [9], a receiver of a data packet r calculates δ_r, which denotes a difference between the estimated hop count to D and $h_{r\mathrm{D}}$ in r's cost table. The estimated hop count is calculated by subtracting one from the hop count to D, which is stored in the received packet. Therefore, δ_r denotes an index how become close the data packet to the D if r forwards it. Next, r calculates the backoff time b_r using δ_r obtained by $\mu_r(\delta_r) = u\big(\varsigma_r(\delta_r+\gamma) - \varsigma_r(\delta_r)\big)$ and $b_r = T_{\max}\big(\varsigma_r(\delta_r) + \mu_r(\delta_r)\big)$, where ς_r is a sigmoid function, $T_{\max}$ is a max backoff time, u is a uniform random decimal of $(0, 1)$, and γ is a parameter. This calculation method realizes that a potential forwarder can obtain a shorter backoff time than the others when it is the closest to the sink node. However, as this method does not consider the on-going computational cost of nodes, it can cause an uneven computational load among nodes.

To balance the computational loads of the nodes, this paper proposes a computational load-based calculation that considers computational resource utilization q_r/Q_r. The receiver r's backoff time b_r is calculated by $b_r = T_{\max}\big(\varsigma_r(\delta_r + \frac{q_r}{Q_r}) + \mu_r(\delta_r + \frac{q_r}{Q_r})\big)$. When there are potential forwarders with the same δ_r, this method gives more priority to a potential forwarder with the smallest computational resource utilization q_r/Q_r to calculate the shortest backoff time than the other methods. Therefore, it can distribute the computational load among nodes with the same costs.

IV. Performance Evaluation

A. Simulation Setup

The performance of the proposed method was evaluated using QualNet [10]. In the evaluation, 25 nodes were placed in a grid with 100 m spacing. The node placed at the upper left corner was set to a source node S, and the node placed at the lower right corner was set to a sink node D. The

nodes employed IEEE 802.11b as a wireless medium, and their communication range and data rate were set to 110 m and 11 Mbps. In the proposed method, Q_i was set to one as the normalised value, and $Q_i = 1$ was 60 s. S generated 100 requests of the ML tasks including P_{S} in 2 s intervals.

As evaluation indices, this paper adopted each node i's peak computational load q_i^{peak}, its standard deviation σ_{peak} excluding S and D, i's total amount of processed computational cost p_i^{total}, its standard deviation σ_{total} excluding S and D, the server's total amount of processed computational cost $p_{\mathrm{server}}^{\mathrm{total}}$, and total number of nodes reaching Q_i. Note that packet transmission success rates of hop count-based and computational load-based calculation were 94.7 % and 94.6 %, respectively. Therefore, evaluation results only include almost no effects owing to packet losses.

B. Simulation Results

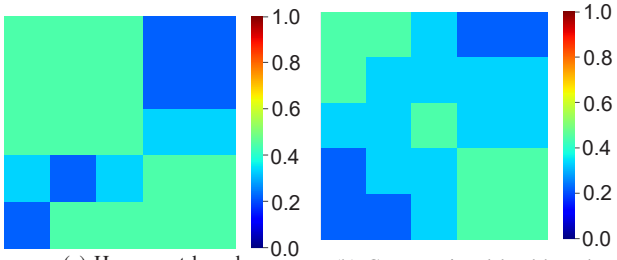
Fig. 3 shows the heat maps of q_i^{peak} for both calculation methods for $P_{\mathrm{S}} = 1$. Although both methods increased the q_i^{peak} of nodes around the S and D, the computational load-based calculation balanced the q_i^{peak} of nodes around the center of the topology.

Fig. 4 shows both methods increased σ_{peak} and σ_{total} with increasing P_{S}. In particular, the computational load-based calculation suppressed the increases more than the hop count-based calculation.

Fig. 5 shows $p_{\mathrm{server}}^{\mathrm{total}}$ and the total number of nodes reaching Q_i with varying P_{S}. The hop count-based calculation increased the number of nodes reaching Q_i as P_{S} increased since it does not consider the computational load; consequently, $p_{\mathrm{server}}^{\mathrm{total}}$ significantly increased. In contrast, the computational load-based calculation can distribute the computational load, thereby reducing the server's load since it can avoid selecting a forwarder that reaches its computational capacity if available.

V. Conclusion

This paper proposed a decentralised division and allocation method of ML tasks using backoff-based OR. The proposed method realized the division and allocation mechanism of an ML task based on the computational load of relay nodes and the load-balanced adaptive relay node selection based on backoff-based OR. In the performance evaluation, the simulation results indicated that the proposed method can properly distribute the processing cost of ML to each relay node. As a future direction of this study, we will evaluate the proposed method using more realistic division models, such as the discrete processing cost, and random topologies.



(a) Hop count-based calculation. (b) Computational load-based calculation.

Fig. 3. Variance of q_i^{peak} when $P_{\mathrm{S}} = 1$. Although both methods increased q_i^{peak} around nodes S and D, the computational load-based calculation can distribute q_i^{peak}.

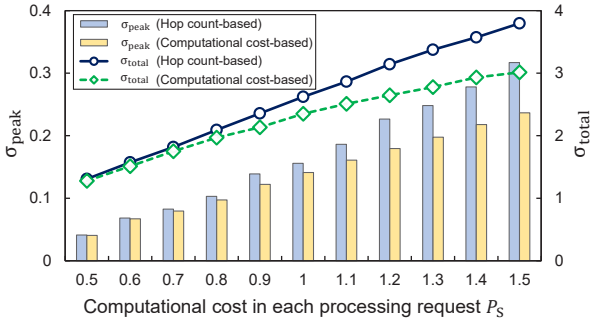


Fig. 4. σ_{peak} and σ_{total} varying P_{S}. Unlike the hop count-based calculation, the computational load-based calculation suppresses the increase in σ_{peak} and σ_{total} even with an increase in P_{S}.

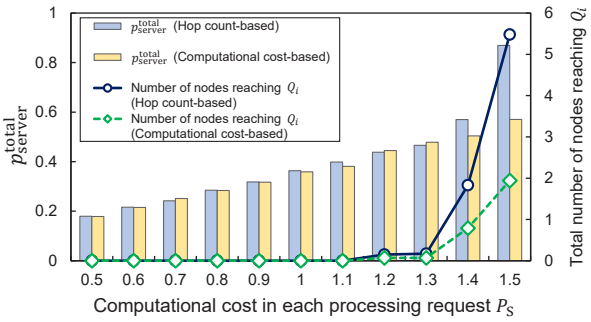


Fig. 5. $p_{\mathrm{server}}^{\mathrm{total}}$ and total number of nodes reaching Q_i with varying P_{S}. The hop count-based calculation significantly increases the computational burden on the server and the number of nodes reaching Q_i as P_{S} increases, whereas the computational load-based calculation suppresses them.

References

[1] H.Alemdar and C.Ersoy, "Wireless sensor networks for healthcare: a survey," Comput. Netw., vol. 54, no. 15, pp. 2688–2710, Oct. 2010.

[2] S.L.Ullo and G.R.Sinha, "Advances in smart environment monitoring systems using IoT and sensors," Sensors, vol. 20, no. 11, pp. 1–18, May 2020.

[3] S.Kumar and S.K.Das, "Target detection and localization methods using compartmental model for internet of things," IEEE Trans. Mobile Comput., vol. 19, no. 9, pp. 2234–2249, Sept. 2020.

[4] X.Wang, L.Gao, S.Mao, and S.Pandey, "CSI-based fingerprinting for indoor localization: a deep learning approach," IEEE Trans. Veh. Tech., vol. 66, no. 1, pp. 763–776, Jan. 2017.

[5] M.A.Alsheikh, S.Lin, D.Niyato, and H.-P.Tan, "Machine learning in wireless sensor networks: algorithms, strategies, and applications," IEEE Commun. Surv. and Tutor., vol. 16, no. 4, pp. 1996–2018, April 2014.

[6] H.Sharma, A.Haque, and F.Blaabjerg, "Machine learning in wireless sensor networks for smart cities: a survey," Electronics, vol. 10, no. 9, pp. 1–22, April 2021.

[7] K.Umeda, T.Nishitsuji, T.Asaka, and T.Miyoshi, "Distributed processing method for deep learning in wireless sensor networks," IEICE Commun. Exp., vol. 10, no. 8, pp. 505–510, Aug. 2021.

[8] J.Motoyama, R.Ooka, T.Miyoshi, T.Yamazaki, and T.Asaka, "Distributed processing allocation of machine learning in wireless sensor networks," IEEE Int. Conf. on Consumer Electronics - Taiwan (ICCE-TW 2020), Taoyuan, Taiwan, pp. 1–2, Nov. 2020.

[9] T.Yamazaki, R.Yamamoto, T.Miyoshi, T.Asaka, and Y.Tanaka, "PRIOR: prioritized forwarding for opportunistic routing," IEICE Trans. Commun., vol. E100-B, no. 1, pp. 28–41, Jan. 2017.

[10] QualNet, https://www.keysight.com/jp/ja/assets/3122-1395/technical-overviews/QualNet-Network-Simulator.pdf.