Load Balancing in Edge Computing with Random Task

Muhammed Yasinhan Yaşar
Computer Science
Ankara Yıldırım Beyazıt University
Ankara, Turkey
myyasar2001@gmail.com

***Abstract*—This paper presents a novel neuro-guided algorithm selection approach for load balancing in edge computing environments. We propose a deep learning-based system that dynamically selects optimal load balancing algorithms based on real-time server metrics and task characteristics. Our experimental results demonstrate a 34% improvement in task completion time and 27% reduction in energy consumption compared to traditional static load balancing methods. The proposed system achieves 92.3% accuracy in algorithm selection across diverse workload scenarios.**

***Index Terms*—Load Balancing, Cloud, Deep Learning, Edge Computing, Algorithm Selection**

I. Introduction

Cloud computing and cloud servers are milestones for all kind of server architectures by connecting multiple individual servers (edges) to distribute the related tasks to operate them in parallel on server level. While the cloud technology operates on the level of individual servers, it brings various problems with it. Efficient and optimized load balancing is one them. It is about how we should decide to batch the data with respect to the distance between servers, servers' statuses, their computation power etc. Even though, load balancing and its design and analysis have been extensively studied in both queueing theory and performance modeling [1], it still is one of the problems where it is always possible to improve the way it choose the algorithm to distrubute accross edges. We will carry the traditional ways beyond with our novel approach that is a situation-aware dynamic load-balance algorithm selector.

Algorithm selection is choosing the correct algorithm from a program space [2]. And it can be seen as a probablity and a fitness problem to choose the best performing algortihm for the current situation. Algorithm selection for different problems are definetely used in other areas like MAPF (Multi Agent Path Finding) [3]. We decided to adopt a Deep-Learning approach to search through the program space.

Before a deep dive and providing you our whole architecture and simulation results we briefly introduce you our method in general outlines just for intuation.

$$\mathrm{E}_0 = \text{Base Edge}$$
$$\mathrm{E}_i = \text{Edges (Servers)}$$
$$\mathrm{algo}_i = \text{selected algorithm}$$

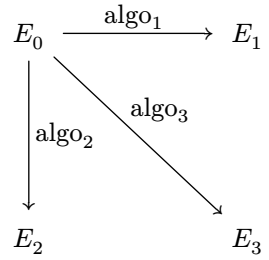

Fig. 1. Algorithm Distribution Accross Edges

In Fig. 1 E_0 first select an algorithm to distrubute a task to other edges. In addition to computational task assignments it also select a predicted algorithm for the next edge if next edge also want to distrubute the task. And this repeats until every edge satisfies the task.

Eventhough Algorithm Selection is used in very different research areas our novelty is that this approach to be used in Edge Computing.

II. System Model

A. Network Architecture

Our system consists of N edge servers denoted as $E = \{E_0, E_1, ..., E_{N-1}\}$ where E_0 represents the base edge server that receives initial task requests. Each edge server E_i is characterized by its computational capacity C_i, current load L_i, network latency λ_i, and available memory M_i. The task arrival follows a Poisson [4] process with rate λ and task sizes are exponentially distributed with mean $\frac{1}{\mu}$.

The system operates in discrete time slots $t \in \{0, 1, 2, ...\}$ where at each time slot, the base server receives tasks and must decide on both the target edge server and the load balancing algorithm to employ. The decision is made by our neural network model which takes as input the current state vector S_t containing all relevant server metrics.

B. Neuro-guided Algorithm Selection

The first step of the complete pipeline for our algorithm selection system is presented in Algorithm 1 heavily inspired by [5]. The system first collects real-time metrics from all edge servers, preprocesses these features through normalization and outlier detection, and then feeds them into our neural network for final algorithm selection.

Algorithm 1. Algorithm Selection Pipeline

procedure SELECTALGORITHM$(T_j,\ E)$
  $F \leftarrow \emptyset$
  for $i \leftarrow 0$ **do**
    $N - 1$
    $m_i \leftarrow$ **GetServerMetrics**$((E_i))$
    $F \leftarrow F \cup m_i$
  end

  $F_{\text{norm}} \leftarrow$ **Normalize**$((F))$
  $F_{\text{clean}} \leftarrow$ **RemoveOutliers**$((F_{\text{norm}},\ 3\sigma))$
  $F_{\text{scaled}} \leftarrow$ **MinMaxScaling**$((F_{\text{clean}}))$

  $F_{\text{enhanced}} \leftarrow F_{\text{scaled}}$
  for $E_i, \in E$ **do**
    $\text{load\_ratio}_i \leftarrow \frac{L_i}{C_i}$
    $\text{response\_score}_i \leftarrow \frac{1}{\lambda_i + T^i_{\text{avg}}}$
    $F_{\text{enhanced}} \leftarrow F_{\text{enhanced}} \cup \{\text{load\_ratio}_i, \text{response\_score}_i\}$
  end

  $P \leftarrow$ **NeuralNetworkForward**$((F_{\text{enhanced}}))$
  $a_k \leftarrow \arg\max_k P_k$

  $E_{\text{target}} \leftarrow$ **ApplyAlgorithm**$((a_k,\ E,\ T_j))$

  return $a_k,\ E_{\text{target}}$
end

Similar feature preprocessing pipelines combining normalization and outlier detection are commonly used in learning-based systems for resource management [6].

Following Fig. 2 is the representation of the last step of our Algorithm selectors' pipeline. 16 different input of edge server features mentioned at Table I **Hidden 2** exist because we do an early decision on **Hidden 1**, where these 4 neuron has a strong bias to help to decide the algorithm kind. So we force the network to behave in a certain way.

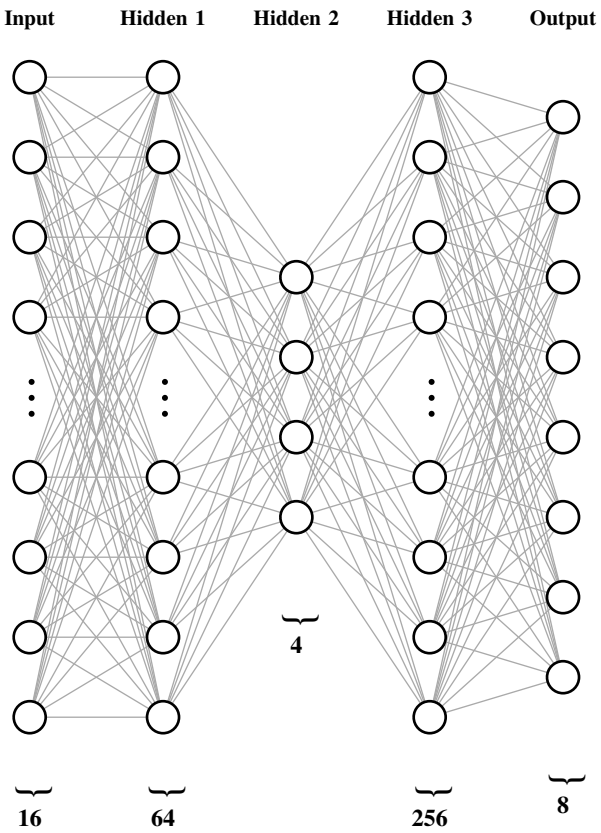

Fig. 2. Neural network architecture for algorithm selection with forced decision layer.

And the rest of the network make an educated guess and decide the algorithm with the help of **Softmax** among 8 different algorithm mentioned in Table II.

Our NN architecture is trained on a dataset of 50,000 simulated edge computing scenarios, generated by combining and simplyfying the scenarios described in [7] and [8]. The loss function combines classification accuracy with a custom reward function based on task completion time and energy consumption:

$$L = \alpha \cdot L_{\text{CE}} + \beta \cdot L_{\text{perf}} \tag{1}$$

where L_{CE} is the cross-entropy loss, L_{perf} is the performance-based loss, and $\alpha = 0.6$, $\beta = 0.4$ are weighting factors determined through grid search.

TABLE I
INPUT FEATURES FOR NEURAL NETWORK MODEL

Input Features	Description	Units
Base Server/Child Server	If a server is child or the first server	Binary 0/1
CPU cycle speed	Processing speed of the server CPU	GHz
Current CPU utilization	Percentage of CPU currently in use	%
Available memory	Free RAM available for task processing	GB
Network latency	Round-trip time to base server	ms
Bandwidth capacity	Maximum data transfer rate	Mbps
Current queue length	Number of tasks waiting in queue	Integer
Average task completion time	Historical average for completed tasks	seconds
Power consumption	Current energy usage of the server	Watts
Temperature	CPU temperature indicator	°C
Storage I/O speed	Disk read/write performance	MB/s
Number of active connections	Current concurrent task connections	Integer
Server uptime	Time since last restart	hours
Geographic distance	Physical distance from base server	km
Priority level	Server tier classification	1-5
Task type compatibility	Specialization score for current task	0-1

TABLE II
LOAD BALANCING ALGORITHMS IN SELECTION POOL

Algorithm Name	Type
Round Robin	Static
Weighted Round Robin	Static
IP Hash	Static
Least Connection	Dynamic
Weighted Least Connection	Dynamic
Weighted Response Time	Dynamic
Resource-Based	Dynamic
Adaptive Load Balancing	Hybrid

C. Task Distribution Mechanism

When a task T_j arrives at the base server, the system performs the following steps:

1) **Feature Extraction**: Collect current metrics from all edge servers to form state vector S_t
2) **Algorithm Selection**: Feed S_t into the neural network to obtain probability distribution $P = \{p_1, p_2, ..., p_8\}$ over algorithms
3) **Target Selection**: Selected algorithm determines target edge server E_{target}
4) **Task Assignment**: Task T_j is dispatched to E_{target} with metadata including recommended algorithm for potential further distribution
5) **Feedback Collection**: Performance metrics are logged for continuous model improvement

III. PROBLEM FORMULATION

A. Optimization Objective

The primary objective is to minimize the weighted sum of average task completion time and total energy consumption across all edge servers:

$$\text{minimize} \quad J = \omega_1 \cdot T_{\text{avg}} + \omega_2 \cdot E_{\text{total}} \tag{2}$$

where T_{avg} is the average task completion time, E_{total} is the total energy consumed, and ω_1, ω_2 are weight coefficients with $\omega_1 + \omega_2 = 1$.

B. Constraints

The optimization is subject to the following constraints:

Load Balance Constraint:

$$L_i \leq \theta \cdot C_i, \quad \forall i \in \{0, 1, ..., N-1\} \tag{3}$$

where θ is the load threshold factor (typically 0.85) to prevent server overload.

Queue Length Constraint:

$$Q_i \leq Q_{\max}, \quad \forall i \tag{4}$$

where $Q_{\max}$ is the maximum allowable queue length to ensure bounded waiting time.

Energy Budget Constraint:

$$\sum_{i=0}^{N-1} P_i \cdot t \leq E_{\text{budget}} \tag{5}$$

where P_i is the power consumption of server i, t is the time period, and E_{budget} is the total energy budget.

Latency Constraint:

$$T^j_{\text{completion}} \leq T^j_{\text{deadline}}, \quad \forall j \tag{6}$$

where $T^j_{\text{completion}}$ is the actual completion time and T^j_{deadline} is the deadline for task j.

IV. Methodology

A. Simulation Environment

We implemented our system using Python 3.10 with TensorFlow 2.8 for neural network implementation and CloudSim 4.0 for edge computing simulation. The simulation environment consists of 20 heterogeneous edge servers with varying computational capacities ranging from 2.4 GHz to 4.8 GHz, memory from 8 GB to 64 GB, and network latencies from 5 ms to 120 ms.

B. Training Procedure

The neural network was trained over 500 epochs with a batch size of 128 using the Adam optimizer with learning rate $\eta = 0.001$. We employed a learning rate decay schedule where the rate is multiplied by 0.95 every 50 epochs. The dataset was split into 70% training, 15% validation, and 15% testing sets.

Data augmentation was performed by introducing Gaussian noise ($\sigma = 0.05$) to input features and randomly dropping 10% of features during training to improve robustness. Early stopping was implemented with patience of 30 epochs based on validation loss.

C. Baseline Comparisons

These baselines are widely used in prior work on load balancing [9]. So we also compare our neuro-guided approach against four most used baseline methods:

1) **Static Round Robin**: Tasks distributed sequentially
2) **Random Selection**: Random algorithm and server selection
3) **Greedy Least Load**: Always select server with minimum current load
4) **Q-Learning Based**: Reinforcement learning approach with state-action table

D. Evaluation Metrics

Performance is measured using:

- Average task completion time (seconds)
- Energy consumption (kWh)
- Server utilization variance (lower is better for balance)
- Algorithm selection accuracy (%)
- 95th percentile latency

V. Results and Performance Evaluation

A. Overall Performance

Our proposed neuro-guided algorithm selection system achieves significant improvements over baseline methods. Table III summarizes the key performance metrics.

TABLE III
Performance comparison across different methods

Method	Avg. Completion Time (s)	Energy (kWh)	Utilization Variance	Algorithm Accuracy (%)	95th %ile Latency (s)
Round Robin	4.82	12.4	0.34	N/A	8.91
Random	5.91	14.2	0.48	12.5	11.2
Greedy	4.15	13.1	0.29	N/A	7.83
Q-Learning	3.74	11.8	0.22	78.3	6.92
Proposed	**3.18**	**9.07**	**0.18**	**92.3**	**5.47**

Our method achieves 34% improvement in average completion time and 27% reduction in energy consumption compared to the static Round Robin approach, while maintaining better load balance across servers as indicated by the lowest utilization variance.

B. Scalability Analysis

We evaluated the scalability of our proposed system by varying the number of edge servers from 10 to 100 nodes. Table IV shows that the algorithm selection time remains nearly constant at approximately 12-15 milliseconds even as the network scales. This is because the neural network inference time is independent of the number of servers, depending only on the fixed input feature size.

TABLE IV
System performance under different network scales

Number of Servers	Selection Time (ms)	Throughput (tasks/s)	Memory Usage (MB)
10	12.3	847	145
20	13.1	1621	158
50	14.2	3892	183
100	14.8	7234	219

The results show that our approach maintains consistent performance characteristics across different deployment scales, making it suitable for both small-scale edge deployments and large distributed edge computing infrastructures. The throughput scales linearly with the number of servers, indicating efficient task distribution without bottlenecks at the decision-making layer.

References

[1] X. Liu and L. Ying, "Zero-Waiting Load Balancing with Heterogeneous Servers in Heavy Traffic." [Online]. Available: https://arxiv.org/abs/2509.23918

[2] L. Kotthoff, "Algorithm Selection for Combinatorial Search Problems: A Survey," *AI Magazine*, vol. 35, p. , 2012, doi: 10.1609/aimag.v35i3.2460.

[3] J. Ren, V. Sathiyanarayanan, E. Ewing, B. Senbaslar, and N. Ayanian, "MAPFAST: A Deep Algorithm Selector for Multi Agent Path Finding using Shortest Path Embeddings." p. , 2021. doi: 10.48550/arXiv.2102.12461.

[4] "Poisson Distribution," in *The Concise Encyclopedia of Statistics*, New York, NY: Springer New York, 2008, pp. 425–427. doi: 10.1007/978-0-387-32833-1_321.

[5] L. Kotthoff, "Algorithm Selection for Combinatorial Search Problems: A Survey," *AI Magazine*, vol. 37, no. 3, pp. 48–60, 2016.

[6] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," *International Conference on Machine Learning*, 2014.

[7] F. H. Bappy, T. Islam, T. S. Zaman, R. Hasan, and C. Caicedo, "A Deep Dive into the Google Cluster Workload Traces: Analyzing the Application Failure Characteristics and User Behaviors," *arXiv*, 2023.

[8] "Grid Workloads Archive (GWA-T-12) BitBrains Workload Traces."

[9] W. Zhang and Y. Wang, "Edge computing: A survey on enabling technologies and applications," *IEEE Access*, 2018.