BalanceNet Orchestrator: A KQV-based Dynamic Task Allocation for Distributed Deep Learning

Teh-Jen Sun
Department of Artificial Intelligence
Kyung Hee University
Yongin-si, South Korea
dlwmznzl1@khu.ac.kr

Thien-Thu Ngo
Department of Computer Science and Engineering
Kyung Hee University
Yongin-si, South Korea
thu.ngo@khu.ac.kr

Eui-Nam Huh
Department of Computer Science and Engineering
Kyung Hee University
Yongin-si, South Korea
johnhuh@khu.ac.kr

***Abstract*—In Artificial Intelligence(AI), training expansive models with billions of parameters necessitates substantial computational resources. This requirement has led to the adoption of parallel computing frameworks. However, these frameworks often confront node performance imbalances due to disparities in computational capabilities and network conditions. To address this issue, we introduce the BalanceNet Orchestrator(BNO), a dynamic task allocation algorithm designed to equilibrate workloads in parallel training environments. BalanceNet Orchestrator assesses and adjusts to node-specific performance in real time, facilitating optimal workload distribution and resource utilization. This method significantly enhances training efficiency and accelerates model convergence, presenting an efficient approach for training large-scale AI models within parallel training architecture.**

***Keywords*—AI, Parallel training, Distributed, heterogeneous environment**

I. INTRODUCTION

The evolution of Artificial Intelligence(AI) has precipitated significant advancements across various fields, notably in the development and deployment of large-scale models for tasks such as language processing, image recognition, and complex data analysis[1][2]. The burgeoning size and complexity of these models necessitate augmented computational resources, highlighting the limitations of traditional single-node computing systems in terms of efficiency and scalability.

The growing complexity of AI models has led to the adoption of parallel processing techniques within the research community[3]. These techniques aim to decrease the training time of large-scale models by distributing computational tasks across multiple nodes. However, the effectiveness of parallel processing is often hampered by inconsistent computational power, memory, and network capabilities across the nodes[4][5], leading to inefficiencies in workload distribution and synchronization.

To address these challenges, this study introduces the BalanceNet Orchestrator, a novel architecture incorporating the Key-Query-Value(KQV) structure. BalanceNet Orchestrator is engineered to optimize task allocation within parallel processing systems, effectively managing the diversity of node capabilities and promoting more efficient resource allocation and utilization. By dynamically adjusting to real time performance metrics and leveraging the KQV structure, BalanceNet Orchestrator ensures a tailored task distribution to meet the specific needs of each node, significantly enhancing the overall efficiency of parallel learning. Figure 1 provides an overview of the BalanceNet Orchestrator architecture, demonstrating its utility in varied computational environments. This architecture not only analyzes each node's computational power, memory capacity, network bandwidth, and latency in real time, but it also addresses the complexity of interactions between nodes. This comprehensive approach includes tackling challenges such as minimizing the impact of external variables like network latency and alleviating bottlenecks in the learning process, ultimately aiming to enhance the scalability and efficiency of large-scale AI model training.

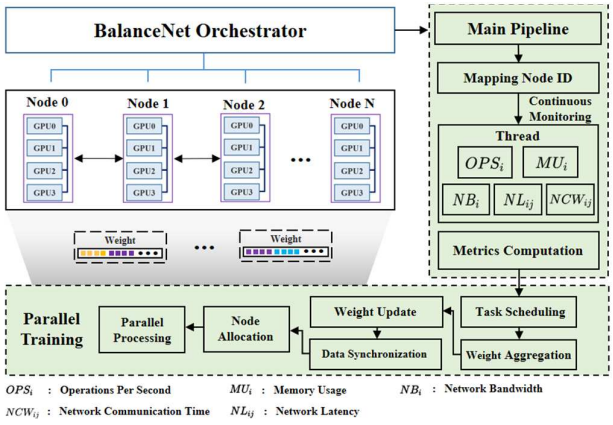

Fig. 1. BalanceNet Orchestrator Overview

II. RELATED WORK

2.1 Parallel Training in Distributed Environments

Parallel training in distributed environments is crucial for training large-scale AI models efficiently. This section covers several parallel processing strategies: data parallelism, distributing data across multiple processors; model parallelism, splitting the model across processors; and pipeline parallelism, dividing the model training into stages[6][7]. Recent studies have focused on combining these strategies to accelerate the training of extensive models[8][9]. Parallel training architectures usually employ Peer-to-Peer(P2P) or master-worker node structures. P2P structures facilitate direct task distribution but increase communication overhead with more nodes. Conversely, master-worker structures, though complex, are more effective for large-scale systems by reducing communication costs[10].

Developing advanced algorithms and techniques to address challenges like load balancing, data synchronization, and resource allocation is essential for optimizing parallel training. These developments are vital for the swift and accurate

training of large-scale AI models[11], continuously evolving and shaping the future of AI model training.

2.2 Task Allocation and Load Balancing Strategies

Efficient task allocation and load balancing are fundamental in distributed systems for optimal system performance. This section focuses on algorithms and methodologies that dynamically distribute and adjust tasks based on the variable performance capabilities of individual nodes. These strategies are essential for addressing performance imbalances, enhancing system efficiency[12][13]. Research in space-search dynamic scheduling techniques is noteworthy, aiming to refine the efficiency of operations across nodes[14].

Building upon the task allocation and load balancing strategies discussed, the development of interfaces for asynchronous communication takes on a significant role[15]. These advanced interfaces are crucial in distributed AI models for scalable and efficient training, facilitating smoother data transfer and synchronization across nodes, a key aspect in parallel computing. Such advancements in communication protocols are instrumental in reducing latency and enhancing throughput, thereby contributing to more effective and rapid model training processes[16].

2.3 Heterogeneous Computing and Resource Optimization

The increasing complexity of AI models necessitates advanced resource optimization in heterogeneous computing environments. This section covers efficient resource management and allocation in systems with diverse node capabilities, focusing on adapting resource usage to workload demands and maximizing underutilized resources like idle CPU and bandwidth in GPU clusters[17]-[19]. The rise of heterogeneous GPU clusters and cloud-based Machine Learning as a Service(MLaaS) solutions is also examined, showcasing their role in enhancing distributed learning tasks and addressing resource allocation challenges[20][21]. These developments indicate a trend towards more adaptive, resilient, and scalable AI infrastructures, improving computational efficiency and model accuracy in AI development[22].

III. The Proposed Systems

3.1. Main Framework

In this study, the 'BalanceNet Orchestrator' architecture proposes dynamic task allocation mechanisms, addressing node performance imbalances within heterogeneous distributed computing environments. This section elaborates on the conceptual structure and theoretical underpinnings of the architecture, exploring strategic approaches to enhance distributed learning efficiency and reduce training times.

The 'BalanceNet Orchestrator' architecture comprehensively evaluates each node's computational power, memory capacity, network bandwidth, and latency in real time, while simultaneously addressing the intricacies of node interactions. Its sophisticated approach dynamically aligns workloads with the capabilities of individual nodes and the complex dynamics of the network, striving to optimize overall system efficiency. Key to this strategy is addressing challenges like mitigating the impact of external factors such as network latency and easing bottlenecks that hinder the learning process.

Figure 2 provides a detailed visual representation of the 'BalanceNet Orchestrator' architecture. It illustrates the sophisticated design and operational functionality, showcasing how the architecture optimizes node performance and efficiently distributes tasks. This depiction clarifies the BalanceNet Orchestrator's component in enhancing parallel training efficiency within distributed environments.

Figure 2 Workflow Description:

(1) Multiple learning nodes initiate the 'BalanceNet Orchestrator' architecture, forming a P2P structure for mapping performance metrics and IDs. These metrics are stored as 'Key', essential for determining each node's workload capabilities.

(2) The system dynamically calculates metrics for each node, essential for ongoing parallel processing. This phase operates continuously on individual threads, adapting to real time performance changes.

(3) Nodes are organized based on performance, involving steps like model initialization, task sequencing, and weight initialization. This process concludes with Batch Size

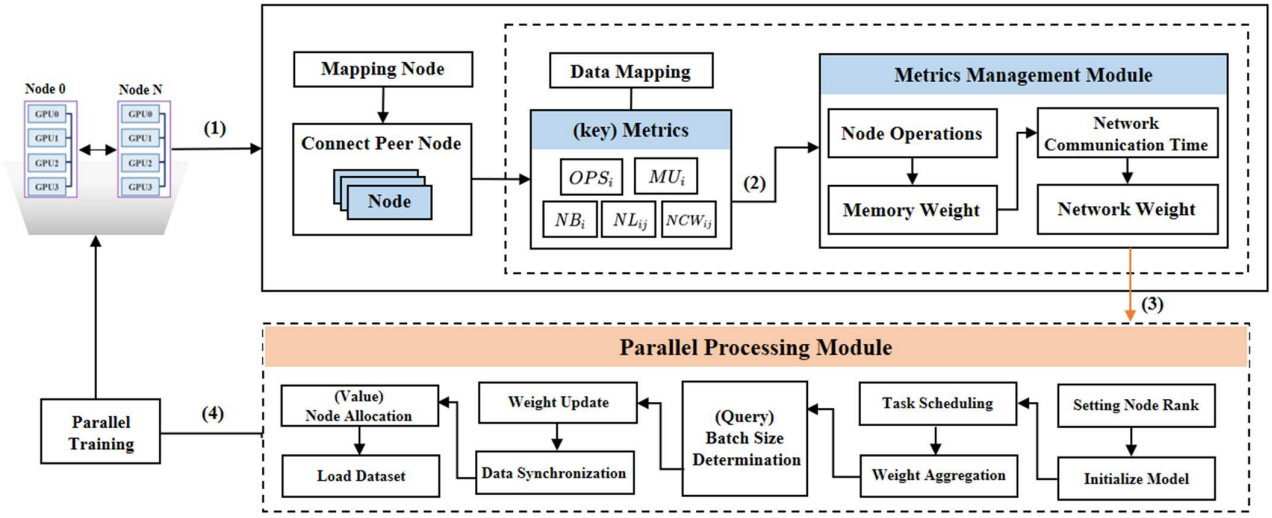

Fig. 2. BalanceNet Orchestrator Architecture

Determination for each node, aligning with the 'Query' component of the KQV model, which dictates task assignment based on node capabilities.

(4) Finally, the system concludes with dataset loading, leading to the commencement of parallel training. This step represents the dynamic allocation of nodes, corresponding to the 'Value' in the KQV model, ensuring optimal task distribution based on node performance and task requirements.

This detailed workflow explanation underscores the intricate design and the significance of each component within the 'BalanceNet Orchestrator' architecture. Figure 2 adeptly demonstrates how each phase interconnects to enhance the overall system's performance, playing a crucial role in the effective management of nodes and optimization of task allocation within complex distributed learning environments. Such a systematic approach significantly boosts the efficiency of distributed learning systems, paving the way for more sophisticated and efficient approaches in AI model training methodologies.

3.2 KQV-Based Mathematical Architecture of the 'BalanceNet Orchestrator' Algorithm

In this study, we introduce a KQV-based 'BalanceNet Orchestrator' Algorithm that incorporates a mathematical Architecture to address the issue of performance imbalance among nodes within parallel training environments. The adoption of the KQV structure is rooted in its ability to effectively handle the dynamic and complex nature of distributed learning systems. By breaking down the performance assessment into three distinct but interrelated components—Key, Query, and Value—the algorithm can precisely evaluate and address the unique needs of each node.

This algorithm dynamically assigns workload to each node by continuously monitoring computational capabilities, memory usage(MU), and network latency(NL), optimizing task distribution in real time. It quantifies three primary performance indicators—Key, Query, and Value—which collectively determine the optimal batch size to be allocated to each node. This approach enables a more nuanced and adaptive task distribution, significantly enhancing the efficiency and scalability of parallel training processes. Specifically, the Key metric assesses the computational capability, the Query addresses the current demand or requirement of the node, and the Value represents the node itself, allowing for a tailored workload distribution that optimally utilizes each node's resources.

The performance indicators K_i, memory weight MW_i, network communication weight NCW_{ij}, and network latency NT for node i are defined as follows:

$$K_i = \frac{OPS_i}{\sum_{k=1}^{N} OPS_k} \qquad (1)$$

K_i represents the computational capacity of node i relative to the total computational capacity of all nodes in the network. This metric is critical for assessing each node's capability to handle computational tasks in the distributed system.

$$MW_i = 1 - \frac{MU_i}{\sum_{k=1}^{N} MU_k} \qquad (2)$$

MW_i evaluates the memory availability of node i in relation to the overall memory usage across all nodes. This indicator helps to determine how much memory each node can utilize without overburdening the system

$$\mathrm{NL} = T_{\text{response}} - T_{\text{request}} \qquad (3)$$

NL measures the overall network latency, calculated as the difference between the response time and request time in the network communication. This metric is crucial for determining the efficiency of data transfer and synchronization across the distributed system.

$$NCW_{ij} = 1 - \frac{NL_{ij}}{\max_{k=1}^{N} NL_{ik}} \qquad (4)$$

NL_{ij} specifically denotes the network communication time between nodes i and j, capturing the latency in data transfer specific to that node pair. This metric is essential for optimizing node-to-node communication within the distributed network.

With these definitions in, OPS_i represents the operations per second for node i, MU_i denotes the memory usage, indicating the amount of memory utilized by the node, and, NL_{ij} is the network communication time between nodes i and j, reflecting the latency in data transfer between nodes. N signifies the total number of nodes.

The batch size BS_i to be allocated to node i is calculated by considering the performance indicators and workload requirements as follows:

$$BS_i = max\left(\left\lfloor \frac{K_i + MW_i + \sum_{j=1}^{N} NCW_{ij}}{3} \times B \right\rfloor, 1\right) \qquad (5)$$

In Equation (5), B signifies the baseline batch size, establishing a standard for the initial task allocation among nodes. The floor function $\lfloor \cdot \rfloor$ rounds the computed value down to the nearest smaller integer, ensuring that the batch size remains an integer. The max function guarantees that the batch size for each node is never less than 1, ensuring every node receives at least a minimum workload. Here, K_i represents the relative computational capacity of node i, and MW_i reflects its available memory. The sum $\sum_{j=1}^{N} NCW_{ij}$ denotes the collective network communication weight for node i in relation to all other nodes, incorporating network latency into the task allocation process.

This formula dynamically adjusts the batch size for each node, considering the computational power(K), memory usage(MU), and Network Communication Weight(NCW). By doing so, it enhances the overall efficiency of distributed learning and reduces training time. This mathematical approach allows the 'BalanceNet Orchestrator' Algorithm to effectively address the complexities inherent in distributed environments, leading to a more efficient and balanced workload distribution across the network.

3.3 Operational Implementation of the 'BalanceNet Orchestrator' Algorithm for Parallel training

In this section, we discuss the implementation of the 'BalanceNet Orchestrator' algorithm, designed to address performance imbalances among computational nodes in

parallel training operations. This algorithm dynamically calibrates the allocation of batch sizes based on a real time analysis of each node's computational throughput, memory allocation, and network communication latency. The master node acts as a central aggregator of performance metrics, using this information to recalibrate workload distribution across the network, thus optimizing the overall throughput of the parallel training paradigm.

The algorithm operates as outlined below:

Algorithm 1 'BalanceNet Orchestrator' for Parallel training

```
Procedure DYNAMIC_KQV_TASK_ALLOCATION(rank, world_size, stop_event, update_interval)
    while NOT stop_event.IS_SET() do
        OPS[rank], MU[rank], NL[rank] = COLLECT_CURRENT_METRICS(rank)
        if rank IS FIRST_NODE then
            AGGREGATE_ALL_METRICS(OPS, MU, NL, world_size)
            K = [OPS[i] / SUM(OPS) for i in range(world_size)]
            MW = [1 - MU[i] / SUM(MU) for i in range(world_size)]
            NCW = [1 - NL[i] / MAX(NL) for i in range(world_size)]
            for node_id in range(world_size) do
                BS = MAX(FLOOR((K[node_id] + MW[node_id] + NCW[node_id]) / 3 * BASE_BATCH_SIZE), 1)
                UPDATE_NODE_BATCH_SIZE(node_id, BS)
            end for
            DISTRIBUTE_BATCH_SIZES(world_size)
        else
            SEND_METRICS_TO_FIRST_NODE(OPS[rank], MU[rank], NL[rank], rank)
    end while
end procedure
```

Where:

- SUM(OPS) is the sum of operations per second across all nodes.
- SUM(MU) is the sum of memory usage across all nodes.
- MAX(NL) is the maximum network time across all nodes.
- BASE_BATCH_SIZE is a predefined baseline batch size.
- FLOOR is the floor function, rounding down to the nearest integer.
- MAX ensures that the batch size does not fall below 1.

By strategically optimizing task allocation based on each node's unique performance metrics, the algorithm efficiently reduces processing bottlenecks and accelerates the model's convergence. This detailed description provides an in-depth understanding of the algorithm's operational implementation, offering critical insights into the architectural design and optimization of parallel training architecture.

3.4 Integrated Parallel training Methodology of the 'BalanceNet Orchestrator' Architecture

The BalanceNet Orchestrator algorithm, central to our study, utilizes a unique Key-Query-Value structure for dynamic task allocation, focusing on real-time server performance metrics to efficiently distribute computational tasks in distributed computing environments. To demonstrate the BalanceNet Orchestrator's effectiveness in parallel training, we incorporated the Distributed Data Parallel(DDP) processing framework and the NVIDIA Collective Communications Library(NCCL) in our experimental setup. It's important to note that DDP and NCCL were used as tools to facilitate the experimental validation of our algorithm, rather than being integral components of the BalanceNet Orchestrator architecture itself.

DDP provided the necessary infrastructure for parallel execution across multiple nodes, enabling us to test and evaluate the task allocation efficiency of the BalanceNet Orchestrator in a controlled, parallel computing environment. Similarly, NCCL was utilized primarily for its high-performance communication capabilities in multi-node and multi-GPU settings, crucial for handling the large-scale data transfers and complex computations in our experiments. This strategic use of DDP and NCCL underscores the versatility and adaptability of the BalanceNet Orchestrator in enhancing parallel training processes within various distributed computing.

IV. EXPERIMENTAL EVALUATION

4.1 Experimental Environment

This study's experimental setup involved a system running Ubuntu 20.04.6, equipped with four NVIDIA RTX 2080 Ti GPUs, with each GPU dedicated to one of the four computing nodes. This homogeneous hardware environment ensures consistent computational capabilities across nodes, a critical factor for accurately evaluating the BalanceNet Orchestrator's parallel training performance.

4.2 Models and Dataset

In this study, our objective was to assess the BalanceNet Orchestrator's efficacy in dynamic task allocation within parallel training environments, using the KQV structure. For this purpose, we selected a range of convolutional neural network models including CNN, VGG16, ResNet, and AlexNet due to their varied architectural complexities and computational demands. These models, commonly used in computer vision tasks, were chosen to demonstrate the algorithm's ability to efficiently handle diverse workloads.

Furthermore, the MNIST dataset, widely recognized as a standardized benchmark in the field of computer vision, was utilized. This choice was strategic, not due to a focus on computer vision, but to provide a familiar and efficient platform for evaluating our algorithm's task allocation and workload distribution capabilities. The widespread recognition of MNIST ensures the reproducibility of our results and facilitates a clear evaluation of the BalanceNet Orchestrator's performance in a parallel processing context.

4.3 Experimental Methodology

This study's experimental design focused on evaluating three task allocation strategies in parallel training environments: the BalanceNet Orchestrator algorithm, n/1 Equal Distribution, and Random Distribution. These strategies were applied across various convolutional neural network models, including Baseline CNN, VGG16, ResNet, and AlexNet, to assess their effectiveness in a parallel learning context. The primary aim was to analyze and compare the efficiency of these strategies, particularly focusing on weight update times as a key performance indicator in parallel training systems.

For each neural network model, we performed a total of 2400 weight updates to ensure a comprehensive evaluation. To accurately assess the performance of each task allocation strategy, we averaged the 'weight update time' metrics every 100 updates. This approach provided a granular insight into the efficiency and effectiveness of the BalanceNet Orchestrator in comparison to the other strategies over an extended period. This methodology was crucial for identifying trends and patterns in the performance data, thus offering a robust evaluation of the algorithm's impact on training efficiency and speed.

4.4 Experimental Results and Analysis

In our extensive evaluation of the BalanceNet Orchestrator architecture, as detailed in Figures 3 to 7, we scrutinized the performance across four convolutional neural network models: Baseline CNN(Figure 3), AlexNet(Figure 4), ResNet50(Figure 5), and VGG16(Figure 6). Each model underwent assessment under three different task allocation strategies: the BalanceNet Orchestrator method, n/1 Equal Distribution, and Random Distribution. The principal metrics for this analysis were training time and computational efficiency, particularly focusing on communication time during weight updates across nodes. This emphasis aligns with our study's objective to enhance efficiency in distributed training environments by reducing node-to-node communication overhead and balancing computational loads effectively.

4.4.1 Baseline CNN Model Analysis

In this study, the Baseline CNN model served as an initial test bed for evaluating the BalanceNet Orchestrator. While this model possesses a relatively simple architectural design, it was critical for establishing a foundational understanding of our algorithm's impact in a controlled environment. The primary focus was not on the intricacy of the Baseline CNN itself, but rather on its role as a benchmark to demonstrate the BalanceNet Orchestrator's capability to enhance computational efficiency in a straightforward setting.

The application of the BalanceNet Orchestrator to the Baseline CNN model yielded a significant reduction in computational time, recorded at 0.103ms. This reduction is indicative of the algorithm's potential to streamline task distribution processes, even in less complex neural network frameworks. The efficiency observed with the Baseline CNN model is a key indicator of the BalanceNet Orchestrator's broader applicability and effectiveness. This foundational analysis sets the stage for subsequent, more comprehensive evaluations with advanced models like AlexNet and VGG16. It confirms our hypothesis that the BalanceNet Orchestrator can be a valuable tool for optimizing performance across a spectrum of neural network architectures, thus highlighting its practical utility in diverse AI models.

4.4.2 AlexNet Model Analysis

The AlexNet model, with an average performance metric of 0.212ms for the BalanceNet Orchestrator, highlighted its capacity to handle complex workloads, outperforming the n/1 Equal Distribution(0.217ms) and Random Distribution (0.244ms). This performance advantage is attributed to the algorithm's effective management of AlexNet's unique architecture and computational demands.

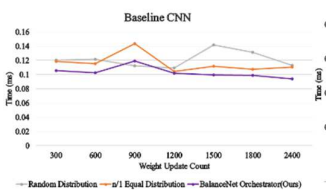

Fig. 3. Baseline CNN

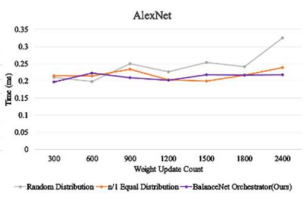

Fig. 4. AlexNet

4.4.3 ResNet50 Model Analysis

ResNet50's performance under the BalanceNet Orchestrator, with an average of 0.263ms, demonstrated the algorithm's effectiveness in managing deep architectures. Compared to the n/1 Equal Distribution(0.323ms) and Random Distribution (0.283ms), this result highlights the algorithm's ability to optimize task allocation in complex networks.

4.4.4 VGG16 Model Analysis

For the VGG16 model, the BalanceNet Orchestrator's performance metric of 0.263ms reflects its capability to adapt to unique architectural demands, providing a more efficient task allocation compared to the other strategies.

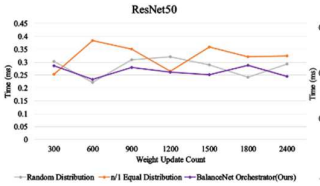

Fig. 5. ResNet50

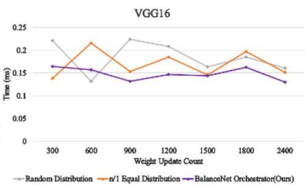

Fig. 6. VGG16

4.4.5 Analysis Summary

Our comprehensive analysis, as illustrated in Figure 7, clearly demonstrates the BalanceNet Orchestrator's consistent superiority over traditional task allocation strategies across all tested models. In the Baseline CNN model, for instance, the BalanceNet Orchestrator achieved a remarkable reduction in computational time to 0.103ms, surpassing the n/1 Equal Distribution and Random Distribution methods. Moreover, it maintained uniform communication speeds at every weight update, indicating a balanced and efficient processing across the network. This pattern of enhanced performance was similarly observed in the AlexNet, ResNet50, and VGG16 models, where the BalanceNet Orchestrator's average performance metrics were 0.212ms, 0.263ms, and 0.263ms respectively. These figures not only indicate significant improvements in efficiency but also showcase the BalanceNet Orchestrator's ability to optimize weight update times and computational processes in various complex architectures.

A key factor in the success of the BalanceNet Orchestrator is its dynamic task allocation mechanism, based on the real-time analysis of node-specific metrics within the Key-Query-Value framework. This meticulous approach, which takes into account each node's capabilities and interrelations, illustrates the sophisticated nature of the BalanceNet Orchestrator's task allocation strategy. It also highlights the crucial role of adaptive task distribution in improving the efficiency of parallel training processes in distributed computing environments.

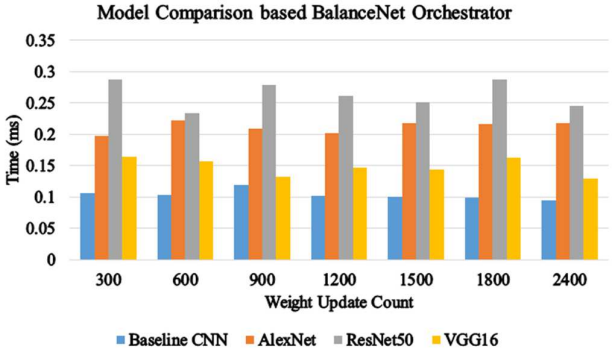

Fig. 7. Model Comparison with BalanceNet Orchestrator

While the current study predominantly focuses on convolutional neural network models within computer vision tasks, we are planning to broaden our research horizons. Future research will aim to apply the BalanceNet Orchestrator across a diverse range of AI models, including those used in natural language processing and other areas. This expansion is not just about validating the algorithm's flexibility and scalability; it's about establishing the BalanceNet Orchestrator as a versatile and universally applicable framework for efficient parallel training in varied and heterogeneous computing environments.

V. Conclusion

In this study, the 'BalanceNet Orchestrator' architecture, grounded in the Key-Query-Value dynamic task allocation model, has been presented as a transformative solution for parallel training in heterogeneous computational environments. The BalanceNet Orchestrator 's adept handling of node performance disparities and its real time adaptive task distribution strategy have shown to significantly reduce weight update latency and training durations, enhancing the efficiency of network communications. This algorithm not only presents a sophisticated approach to resource allocation in parallel processing systems but also sets a foundation for future advancements in AI, particularly in scaling up for large-scale projects. Looking ahead, the plan is to broaden the scope of BalanceNet Orchestrator to encompass a wider range of AI models, further solidifying its role in advancing the field of efficient parallel processing architectures.

Aknowledgment

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2023-RS-2023-00258649) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation) and the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.RS-2022-00155911, Artificial Intelligence Convergence Innovation Human Resources Development (Kyung Hee University)). Professor Eui-Nam Huh is the corresponding author.

References

[1] Keuper, Janis, and Franz-Josef Preundt. "Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability." 2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC). IEEE, 2016.

[2] Chen, Mingzhe, et al. "Distributed learning in wireless networks: Recent progress and future challenges." IEEE Journal on Selected Areas in Communications 39.12: 3579-3605, 2021.

[3] Woodworth, Blake E., Kumar Kshitij Patel, and Nati Srebro. "Minibatch vs local sgd for heterogeneous distributed learning." Advances in Neural Information Processing Systems 33: 6281-6292, 2020.

[4] Hashemi, Sayed Hadi, Sangeetha Abdu Jyothi, and Roy Campbell. "Tictac: Accelerating distributed deep learning with communication scheduling." Proceedings of Machine Learning and Systems 1: 418-430, 2019.

[5] Ghosh, Avishek, et al. "Robust federated learning in a heterogeneous environment." arXiv preprint arXiv:1906.06629, 2019.

[6] Ben-Nun, Tal, and Torsten Hoefler. "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis." ACM Computing Surveys (CSUR) 52.4: 1-43, 2019.

[7] Verbraeken, Joost, et al. "A survey on distributed machine learning." Acm computing surveys (csur) 53.2: 1-33, 2020.

[8] Rudin, Nikita, et al. "Learning to walk in minutes using massively parallel deep reinforcement learning." Conference on Robot Learning. PMLR, 2022.

[9] Song, Zhuoyang, et al. "Parallel Learning for Legal Intelligence: A HANOI Approach Based on Unified Prompting." IEEE Transactions on Computational Social Systems, 2023.

[10] Shoeybi, Mohammad, et al. "Megatron-lm: Training multi-billion parameter language models using model parallelism." arXiv preprint arXiv:1909.08053, 2019.

[11] Yook, Dongsuk, Hyowon Lee, and In-Chul Yoo. "A survey on parallel training algorithms for deep neural networks." The Journal of the Acoustical Society of Korea 39.6: 505-514, 2020.

[12] Hashemi, Sayed Hadi. Timed execution in distributed machine learning. Diss, 2020.

[13] Shi, Shaohuai, Xiaowen Chu, and Bo Li. "Exploiting simultaneous communications to accelerate data parallel distributed deep learning." IEEE INFOCOM 2021-IEEE Conference on Computer Communications. IEEE, 2021.

[14] Jia, Zhihao, Matei Zaharia, and Alex Aiken. "Beyond Data and Model Parallelism for Deep Neural Networks." Proceedings of Machine Learning and Systems 1: 1-13, 2019.

[15] Peng, Yanghua, et al. "A generic communication scheduler for distributed DNN training acceleration." Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019.

[16] Tang, Zhenheng, et al. "Communication-efficient distributed deep learning: A comprehensive survey." arXiv preprint arXiv:2003.06307, 2020.

[17] Zhang, Jinghui, et al. "PipePar: Enabling fast DNN pipeline parallel training in heterogeneous GPU clusters." Neurocomputing 555: 126661, 2023.

[18] Yi, Xiaodong, et al. "Optimizing distributed training deployment in heterogeneous GPU clusters." Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies, 2020.

[19] Park, Jay H., et al. "{HetPipe}: Enabling large {DNN} training on (whimpy) heterogeneous {GPU} clusters through integration of pipelined model parallelism and data parallelism." 2020 USENIX Annual Technical Conference (USENIX ATC 20), 2020.

[20] Jiang, Yimin, et al. "A unified architecture for accelerating distributed {DNN} training in heterogeneous {GPU/CPU} clusters." 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 2020.

[21] Weng, Qizhen, et al. "{MLaaS} in the wild: Workload analysis and scheduling in {Large-Scale} heterogeneous {GPU} clusters." 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), 2022.

[22] Zhang, Zhen, et al. "Is network the bottleneck of distributed training?." Proceedings of the Workshop on Network Meets AI & ML, 2020.