Dynamic Expert Migration for Expedited Mixture-of-Experts Training

Zhe Zhang*, Chuan Wu*
*Department of Computer Science, The University of Hong Kong, Email: {zzhang2, cwu}@cs.hku.hk

***Abstract*—Recent development has exhibited that neural network models with extensive parameters can achieve impressive performance in various machine learning tasks. The Mixture-of-Experts (MoE) architecture, which substantially increases the model size with parallel expert networks but not the computation, has been successfully exploited in building large models. Due to the large number of model parameters, distributed training of MoE models is necessary, with experts distributed across devices/machines. A main inefficiency in distributed MoE training is due to the unbalanced transfer and processing of different numbers of tokens to and by the experts, which may incur network congestion and prolonged expert training time. We propose a dynamic expert migration (DEMI) algorithm to optimally distribute experts among workers in each training iteration of model learning, and carefully schedule token transmission and expert training to minimize the overall expert training time. Our algorithm is supported by theoretical guarantees. Extensive evaluation demonstrates that DEMI can reduce 34.6% training time compared to representative baseline approaches and achieve 1.3x more balanced workload among workers.**

I. Introduction

The parameter size of neural network (NN) models has increased from millions to trillions in recent years, e.g., Gemma [1] with 2 billion parameters and PaLM [2] with 540 billion parameters. This large model size enables substantial model performance improvement on many real-world tasks, such as natural language processing and image generation [3–6]. The Mixture-of-Experts (MoE) architecture has been successfully adopted to enlarge model capacity using parallel model structures (aka experts) [7]. Unlike directly scaling NN models to large dense models [8], the MoE architecture is usually added as one or multiple MoE layers in an NN model (e.g., a Transformer model), which can significantly expand the model's capacity while introducing little extra computation [3, 4, 7, 9].

With an MoE model, the training samples (e.g., tokens) are fed to different experts for processing, dynamically selected by a lightweight trainable gating network (or gate function, or router) [9–11]. Each sample is typically routed to the top-k ($k \geq 1$) best experts. Compared with dense training methods (whose processing of each sample involves all parameters in the NN model), the sparse activation of MoE architecture significantly reduces the computation workload. Google has developed a language model family, GLaM, using MoE with a maximum of 1.2-trillion parameters [3], which achieves better zero-shot and few-shot performance than GPT-3 (175-billion parameters), while consuming only 1/3 of the training energy as compared to GPT-3 [12].

979-8-3315-4940-4/25/$31.00 © 2025 IEEE

Although the MoE architecture makes it more feasible to learn giant models, training an MoE model is still time and resource intensive. Expert parallelism is commonly adopted where experts are assigned to run on different workers (devices such as GPUs). For example, the 600-billion-parameter MoE model GShard [9] was trained using 2,048 TPUs in 4 days. To improve expert parallelism and expedite MoE model training, addressing the following two aspects is the key:

First, training workload is unbalanced among the experts. Some experts can receive many more tokens than others, leading to an overload at their workers, while other workers may be quite idle. This significantly lowers hardware utilization and training efficiency. In addition, the popularity of experts may change over the training iterations, necessitating dynamic workload balancing.

Second, the all-to-all communication, where workers receive inputs to and output from the MoE layer (consisting of the gating network and experts) from other workers, is one of the most time-consuming operations in distributed MoE training. Li *et al.* [13] show that the all-to-all communication is a major bottleneck blocking subsequent computation in MoE training. Improving all-to-all communication efficiency is crucial for expediting MoE training.

A number of approaches have been proposed to balance the workload and better utilize available experts, including adding auxiliary losses [11, 14], controlling expert capacity [9] and optimizing the gating network to balance token distribution among experts [15, 16]. Changing routing decisions of the gating network based on workload imbalance rather than prior knowledge of expert's ability may introduce noise in MoE learning, impairing model convergence and accuracy [11, 13, 17]. Other proposals duplicate expert instances (shadow experts) among workers [13, 17]. This increases GPU memory consumption of the MoE system, requires parameter synchronization among expert replicas, and limits model scalability.

This paper focuses on improving the training efficiency of the MoE layer in a large MoE model. Without intervening in token-to-expert selection (i.e., without affecting model convergence), we optimize the scheduling of expert computation and token communication in each training iteration of an MoE layer. Given token routing and computation workloads of experts, we advocate expert migration among workers to optimize computation and communication resource utilization. We conduct detailed token transmission and expert computation scheduling, together with parameter transmissions due to

expert migration, to allow pipelining and overlapping among the computation and communication tasks, and minimize the training time of the MoE layer. Our method is applicable to any workload imbalance among experts, any non-uniform expert structures and heterogeneous worker capacities. Our contributions in this paper can be summarized as follows:

▷ We migrate experts among workers to make full use of resources at each worker, taking into account that the resources required by each expert's training are proportional to the number of tokens routed to the expert in the training iteration. This migration allows experts, that require more computation, memory, and bandwidth resources, to be transferred to workers with available capacities. We periodically monitor the workload of each expert and dynamically adjust the migration plan across different training iterations.

▷ We model the training of experts and the transmission of tokens and expert parameters for expert migration as a mini-task scheduling problem. We abstract token transfer between each pair of workers, expert parameter transfers, and token computation of each expert as mini-tasks, and carefully model the dependencies between transmission and training tasks. We design a novel, progressively time-slot-weighted objective for time and resource scheduling of mini-tasks, aiming to minimize the makespan of completing all tasks, namely the training time of the MoE layer. This weighted objective ensures higher priority for mini-tasks to run in earlier time slots, and encourages timely completion of transmission tasks to reduce expert idle time.

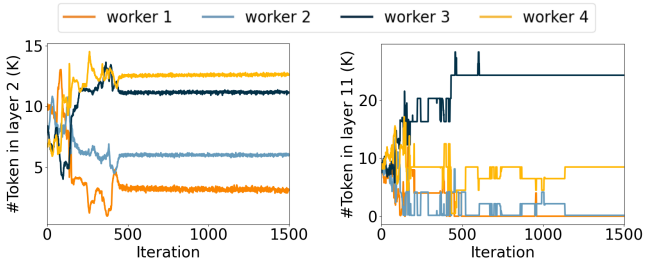
▷ We design an efficient algorithm to solve the expert migration, token transmission, and expert training scheduling problem in each training iteration, generating detailed communication and computation schedules. Our algorithm runs in polynomial time on the number of mini-tasks and experts, enabling fast scheduling of distributed MoE training. The training time of the MoE layer achieved by our algorithm is proven to be bounded by a small constant ratio from the optimal schedule, ensuring high-quality solutions that approach the theoretical optimum.

▷ We implement our system and conduct extensive evaluation in a variety of experimental settings, including configurations with heterogeneous network bandwidth and varying GPUs. Results demonstrate that our algorithm can effectively address workload imbalance among workers and reduce the training time by 34.6% and 37.5% for homogeneous experts and heterogeneous experts, respectively, compared to representative baselines.

II. Related Work and Motivation

A. Mixture-of-Experts Model

An MoE model contains one or multiple mixture-of-experts layers. For example, GShard [9] replaces every feed-forward layer in Transformer [18] with an MoE layer, in both encoder and decoder. An MoE layer [14] contains: (1) multiple expert networks, which can be simply two fully connected layers or heterogeneous delicate structures with different widths or numbers of layers of feed-forward networks [19]; (2) a gating network, which computes the scores of experts for each token and routes the token to a subset of experts based on the scores. A small fully-connected layer is commonly used as the gating network [9, 10, 20]. The top-k algorithm is typically used to route tokens to experts, by which k experts with the highest scores are chosen and the outputs of those experts are weighted by the normalized scores and then transmitted to the subsequent NN layers following the MoE layer. Top-k routing can lead to workload imbalance among workers, where some workers hold experts that receive most of the tokens, while other workers hold experts that receive few [9–11, 14]. We train Transformer-XL [21] with 12 MoE layers on four workers, with each worker holding 4 experts for each MoE layer. As shown in Fig. 1, the number of tokens routed to each worker becomes relatively stable after 500-1200 training iterations. In the second MoE layer, experts on different workers receive varying numbers of tokens; in the penultimate layer, token routing becomes much more biased towards the experts in worker 3, leading to substantially unbalanced computation workload among workers.



(a) Second MoE layer (b) Penultimate MoE layer

Figure 1: #Tokens routed to workers in different MoE layers

To alleviate expert workload imbalance, GShard [9], Switch Transformer [10] and FasterMoE [17] set the capacity of each expert, so that only a limited number of tokens can be fed to each expert. Shazeer *et al.* [14] add an imbalance loss to the NN loss function during training, to balance workload among experts. TA-MoE [11] introduces an auxiliary loss accounting for inter-worker bandwidth, allocating more tokens to experts with higher bandwidth. These approaches may lead to prolonged model convergence or model accuracy decay compared to the original model training [11, 13]. He *et al.* [17] show that GShard requires 153% more iterations to achieve the same loss, while FasterMoE needs 25% more iterations to achieve the same loss as without topology-aware gates.

B. Distributed MoE Training

Most existing MoE training systems [16, 22] evenly distribute experts among workers, not addressing non-uniform expert structures or heterogeneous workers in a training cluster. Our MoE training design supports heterogeneous expert structures and worker capacities.

FasterMoE [17] advocates a dynamic shadow expert mechanism that replicates busy experts on all workers. Parameter synchronization among shadowed experts are required, adding to distributed training latency - their experiments show that the best training speed-up is achieved when only one expert is shadowed. In addition, extra memory on each worker is

needed to host shadowed experts, limiting model scalability (in increasing the number of experts and hence model size). To better decide experts to shadow, Lina [23] estimates token distribution among experts in later MoE layers based on expert selection patterns of previous layers. It requires prior profiling, and the prediction accuracy highly depends on the model structure and datasets.

To inspect training efficiency of expert shadowing, we train a single-layer MoE model with 4 experts on two workers and top-2 gating on the CIFAR-10 classification task [24], and compare training convergence among no expert shadowing, one shadowed expert, full shadowing (every worker has all experts, which is pure data-parallel training). Fig. 2 shows that shadowing one expert leads to the slowest convergence, due to the extra expert replication and gradient synchronization time. The other two achieve similar convergence speeds: MoE training without shadowing does not require gradient synchronization among shadowed experts, while pure data-parallel training minimizes inter-worker token transmission which overcomes the extra gradient synchronization time (it consumes substantially more memory on each worker though).

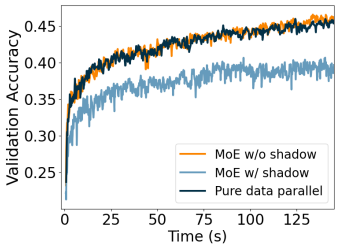
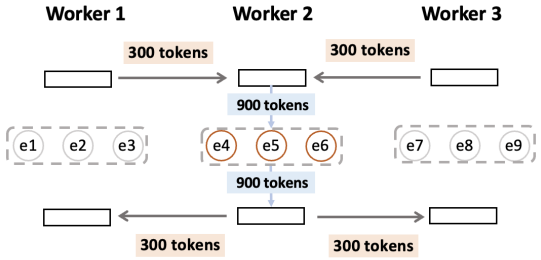


Figure 2: Efficiency downgrade caused by shadowing

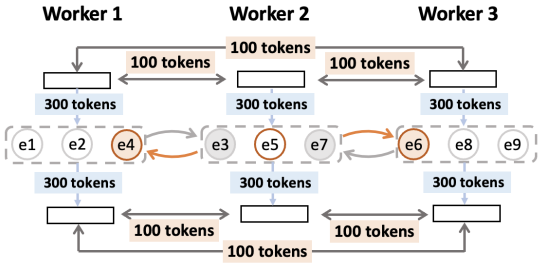
C. Expert Migration

We advocate expert migration instead of expert shadowing. In the example in Fig. 3, each worker hosts 3 experts (expert parallelism) and trains non-MoE parts in the MoE model using data parallelism (on 300 tokens). Suppose the gating network on each worker decides to route 100 tokens to experts e4, e5, and e6, respectively, and e4, e5, and e6 are on worker 2. Without expert migration, worker 1 and worker 3 each need to send 300 tokens to worker 2, wait for worker 2 to compute the results of experts e4, e5, and e6, and receive processed tokens back. Total communication includes transmissions of 600 tokens before and after expert computation, and the expert computation time is the processing time of 900 tokens on worker 2. If we swap expert 4 on worker 2 with expert 3 on worker 1, and swap expert 6 on worker 2 with expert 7 on worker 3, the total communication includes transmissions of 100 tokens between each pair of the three workers before and after expert computation, as well as expert parameter transfers, and the computation workload on each worker is the processing of 300 tokens. Assuming the expert migration time is relatively small, both the computation time and the communication time are about one third of the earlier case.

Janus [25] suggests that each worker should request expert migration when the token assigned to the worker is scheduled to use a specific expert. This process could result in the expert



(a) Before expert migration: unbalanced computation workload and congested inter-worker connection



(b) After expert migration: balanced computation workload and less congested inter-worker connection

Figure 3: Expert migration example

being transferred multiple times during both forward training and gradient updating. SmartMoE [26] advocates expert placement adjustments to roughly balance worker workload, but does not consider communication time for expert migration nor for routing tokens from workers to where the experts are located. Overlapping token communication and expert computation has been adopted in existing MoE training systems (e.g., FasterMoE [17], ScheMoE [27], HiDup [28]), for better resource utilization and training acceleration. Lita [13] divides the all-to-all communication into token granularity and pipelines it with expert computation. Lancet [29] overlaps non-MoE computation with all-to-all communication. We study pipelined expert migration, token transmission and expert training, and jointly schedule communication and computation mini-tasks with efficient algorithm design. The decisions on whether and how to migrate an expert are made automatically.

III. Problem Model

A. System Overview

We aim to optimize the training efficiency of each MoE layer in a large MoE model. The MoE layer consists of a gating network and a set $\mathbb{K}$ of $K = |\mathbb{K}|$ (potentially heterogeneous) expert networks. The K experts are initially distributed randomly among a set $\mathbb{M}$ of $M = |\mathbb{M}|$ workers (usually GPUs) or proportionally according to worker capacities. We suppose the rest of the MoE model (except the MoE layers) is trained using data parallelism with input samples (tokens) distributed among multiple workers in each training iteration. An illustration is given in Fig. 4: tokens 1, 2, 3 and 4 are distributed among the four workers for data-parallel processing before the MoE layer; then through the gating network, tokens 1, 2 and 3 are routed to Expert 2 (on worker 2) for training, while token 4 is sent to Expert 3 (on worker 3); after expert networks'

computation, the processed tokens are sent back to their original workers (e.g., worker 2 sends token 1 back to worker 1) for further processing by the rest of the NN model.

The structures of the experts are allowed to be diverse. Let w_k denote the parameter size of expert $k \in \mathbb{K}$. We allow only one worker to hold an expert at any time. In each training iteration, we advocate expert migration from one worker to another (by sending parameters of the expert) to balance the workload and communication traffic among workers. Let d_k^i denote the size of tokens that worker i routes to expert k in the current training iteration, as decided by the given gating network. We use $b_k^n \in \{0, 1\}$ to denote the original location of each expert at the beginning of the current iteration, i.e., $b_k^n = 1$ if expert k is located at worker n before migration and 0 otherwise.

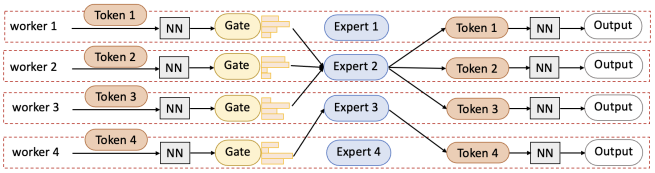


Figure 4: Distributed MoE training

B. Decisions and Constraints

Expert Migration. In each training iteration, we use binary variable s_k^m to indicate whether expert k is located at worker m after possible expert migration. If $s_k^m = 1$ and $b_k^m \neq 1$, expert k is migrated to worker m from the original worker n ($b_k^n = 1$), and all tokens to be routed to expert k will be sent to worker m. The total parameter size of experts migrated from worker n to worker m is $\sum_{k\in\mathbb{K}} b_k^n s_k^m w_k$. We guarantee that an expert is located at only one worker:

$$\sum_{m\in\mathbb{M}} s_k^m = 1, \forall k \in \mathbb{K} \tag{1}$$

Computation and Communication Mini-Tasks. Expert parameter transmission may share communication bandwidth between workers with token transfers, while there are time dependencies among expert migration, token transfer and expert training. We jointly schedule the following tasks involved in the training of an MoE layer: *1. pre-calc token transmissions* (transferring output of NN layers prior to the MoE layer to the experts); *2. expert migration*; *3. expert training*; *4. post-calc token transmissions* (transferring output of the experts back to the respective workers). The inter-task dependencies are: a token cannot be trained on an expert network before the expert is set up on the worker that the token is sent to; a token should be routed back to the original worker for further processing by NN layers following the MoE layer, after it has been trained on its selected experts. Considering one token's training through the MoE layer, task 3 cannot start before task 1 and task 2 are completed, and task 4 cannot start before task 3 is done.

Let r_{k1}^{im} denote the total size of tokens transmitted from worker i to worker m for training on expert k (in task 1 pre-calc token transmissions). r_{k2}^{nm} denotes the parameter size of expert k migrated from worker n to worker m (task 2). r_{k3}^{im} represents the size of tokens trained on expert k on worker m that was routed to worker m from worker i (task 3), and r_{k4}^{mi} denotes the total size of tokens sent from worker m to worker i after training on expert k (in task 4 post-calc communication). Communication and computation workloads for the tasks are:

$$r_{k1}^{im} = d_k^i s_k^m, \forall k \in \mathbb{K}, i, m \in \mathbb{M}, i \neq m \tag{2a}$$
$$r_{k2}^{nm} = w_k s_k^m, \forall k \in \mathbb{K}, n, m \in \mathbb{M}, m \neq n, b_k^n = 1 \tag{2b}$$
$$r_{k3}^{im} = d_k^i s_k^m, \forall k \in \mathbb{K}, i, m \in \mathbb{M} \tag{2c}$$
$$r_{k4}^{mi} = r_{k1}^{im}, \quad \forall k \in \mathbb{K}, i, m \in \mathbb{M}, i \neq m \tag{2d}$$

The total size of tokens on worker m for expert training and the total parameter size of experts on worker m after expert migration should not exceed the memory capacity C_m and Q_m of worker m, respectively:

$$\sum_{k\in\mathbb{K}} \sum_{i\in\mathbb{M}} d_k^i s_k^m \leq C_m, \quad \forall m \in \mathbb{M} \tag{3a}$$
$$\sum_{k\in\mathbb{K}} w_k s_k^m \leq Q_m, \quad \forall m \in \mathbb{M} \tag{3b}$$

Task Co-scheduling. We schedule computation and communication mini-tasks into sufficiently small time slots $t = \{0, 1, ..., T-1\}$, where T is a sufficiently large number such that one training iteration of the MoE layer can be finished in T time slots. We abstract the workload notation of the four types of tasks using a four-tuple (k, a, i, j), where $a \in \mathbb{Z}_{1\sim4}$ corresponds to a task ($\mathbb{Z}_{1\sim4}$ denotes the set of four tasks), k denotes the expert involved, and i and j are the worker indices. Let z_{ka}^{ijt} be the allocated workload of task (k, a, i, j) to execute in time slot t, while the total workload of the task is r_{ka}^{ij} (computed in Eqn. (2a)-(2d)). We ensure that each task is completed within T time slots:

$$\sum_{0\leq t<T} z_{ka}^{ijt} = r_{ka}^{ij}, \ \forall a \in \mathbb{Z}_{1\sim4}, k \in \mathbb{K}, i, j \in \mathbb{M} \tag{4}$$

Let $\mathbb{Z}_m^{t,\text{comp}}$ denote the computation capacity of worker m in time slot t in terms of the maximal size of tokens that can be processed by the worker in the time slot, which is related to CPU cores and GPU resources allocated to MoE layer training on the worker. $\mathbb{Z}_{ij}^{t,\text{comm}}$ represents the communication capacity (corresponding to available bandwidth) from worker i to worker j in time slot t. For example, if a time slot is 2 seconds long and the network bandwidth between worker i and worker j is $1GB/s$, then the communication capacity in t is

Table I: Notation Table

Notation	Description
$\mathbb{M}$	Set of workers
$\mathbb{K}$	Set of experts
$\mathbb{K}(m)$	Set of experts on worker m
$\mathbb{Z}_i^{t,\text{comp}}$($\mathbb{Z}_{ij}^{t,\text{comm}}$)	Computation(communication) capacity in slot t
s_k^m	1 if expert k is on worker m after migration
b_k^n	Constant, 1 if expert k was originally on worker n
w_k	Size of parameters of expert k
d_k^i	Size of tokens sent from worker i to expert k
z_{ka}^{ijt}	Assigned workload of task (k, a, i, j) in slot t
r_{ka}^{ij}	Total workload of task (k, a, i, j)
C_m	Total available memory for tokens on worker m
Q_m	Total expert parameter size limit on worker m
ω_t	Weight of using resources in time slot t

$\mathbb{Z}_{ij}^{t,\text{comm}} = 2GB$. The total workload allocated to each time slot should not exceed communication and computation capacities on each connection and at each worker in the time slot:

$$\sum_{k\in\mathbb{K}} z_{k1}^{ijt} + z_{k2}^{ijt} + z_{k4}^{ijt} \leq \mathbb{Z}_{ij}^{t,\text{comm}},\ \forall i,j \in \mathbb{M}, 0 \leq t < T \quad (5a)$$

$$\sum_{i\in\mathbb{M}}\sum_{k\in\mathbb{K}} z_{k3}^{imt} \leq \mathbb{Z}_{m}^{t,\text{comp}},\ \forall m \in \mathbb{M}, 0 \leq t < T \quad (5b)$$

To formulate inter-task dependencies, e.g., task (k, b, i', j') depends on task (k, a, i, j), we allow the completed portion of task (k, b, i', j') by each time slot t to be no larger than the completed portion of task (k, a, i, j) by time slot $t-1$.

$$\sum_{\tau=0}^{t} z_{k3}^{im\tau} \leq \sum_{\tau=0}^{t-1} z_{k1}^{im\tau},\ \forall k \in \mathbb{K}, i, m \in \mathbb{M}, i \neq m, 1 \leq t < T \quad (6a)$$

$$\sum_{\tau=0}^{t} \frac{z_{k3}^{im\tau}}{d_k^i} \leq \max\{0, B \cdot \sum_{\tau=0}^{t-1} \frac{z_{k2}^{nm\tau}}{w_k} + (1-B)\},$$
$$\forall k \in \mathbb{K}, i, m, n \in \mathbb{M}, b_k^n = 1, n \neq m, 1 \leq t < T \quad (6b)$$

$$\sum_{\tau=0}^{t} z_{k4}^{mi\tau} \leq \sum_{\tau=0}^{t-1} z_{k3}^{im\tau},\ \forall k \in \mathbb{K}, i, m \in \mathbb{M}, 1 \leq t < T \quad (6c)$$

$$z_{k3}^{im0} = 0,\ \forall k \in \mathbb{K}, i, m \in \mathbb{M}, i \neq m \quad (6d)$$

$$z_{k4}^{mi0} = 0,\ \forall k \in \mathbb{K}, i, m \in \mathbb{M} \quad (6e)$$

(6a) guarantees that the tokens can be trained by the respective expert (task 3) after they have been routed to the worker where the expert resides (task 1), if those tokens are not originally on that worker (i.e., $i \neq m$). In (6b), we use a sufficiently large constant B to ensure that the completed percentage of expert k's computation of tokens sent from worker i to worker m (where expert k is located) by time slot t, $\sum_{\tau=0}^{t} \frac{z_{k3}^{im\tau}}{d_k^i}$, can be positive only if expert k has been fully transferred to worker m by $t-1$, i.e., $\sum_{\tau=0}^{t-1} \frac{z_{k2}^{nm\tau}}{w_k} = 1$, if it is originally on another worker n, i.e., $b_k^n = 1, n \neq m$. It enforces the completion of expert migration (task 2) before the expert's computation of received tokens on its worker (task 3). (6c) guarantees that worker m can only send the tokens back to the original worker i (task 4) after the tokens have been processed by expert k on worker m (task 3).

C. Optimization Problem

We aim to minimize the total makespan of completing all tasks in each training iteration of the MoE layer (i.e., minimize the per-iteration training time of the MoE layer) in our distributed MoE training system. For this purpose, we assign increasing weights to the time slots, i.e., $\{\omega_0, \omega_1, \cdots, \omega_{T-1}\}$ with $\omega_t < \omega_{t+1}$, $t = 0, 1, \ldots T-2$, and formulate the minimization objective as $\sum_{t=0}^{T-1}(\omega_t \sum_{a\in\mathbb{Z}_{1\sim4}} \sum_{k\in\mathbb{K}} \sum_{i,j\in\mathbb{M},} z_{ka}^{ijt})$. The weights can be regarded as increasing penalties for carrying out mini-tasks in later time slots, e.g, $\omega_t = t+1$ or $\omega_t = 2^t$. By minimizing this objective, we strive to schedule all tasks to complete as early as possible. Our optimal communication and computation scheduling problem for expediting MoE training is formulated as follows:

$$\text{minimize} \sum_{t=0}^{T-1}(\omega_t \sum_{a\in\mathbb{Z}_{1\sim4}} \sum_{k\in\mathbb{K}} \sum_{i,j\in\mathbb{M}} z_{ka}^{ijt}) \quad (7)$$

$$(1), (2a) - (6e) \quad (7a)$$

$$z_{ka}^{ijt} \geq 0, r_{ka}^{ij} \geq 0, s_k^i \in \{0,1\} \quad (7b)$$
$$\forall a \in \mathbb{Z}_{1\sim4}, i, j \in \mathbb{M}, k \in \mathbb{K}, 0 \leq t < T$$

In the optimization problem, constraint (6b) can be converted to the following linear constraints, by introducing an auxiliary variable $g_k^{imt} \geq 0$, $\forall k \in \mathbb{K}, i, m \in \mathbb{M}, 1 \leq t < T$:

$$\sum_{\tau=0}^{t} \frac{z_{k3}^{im\tau}}{d_k} \leq g_k^{imt}$$

$$g_k^{imt} \geq B \cdot \sum_{\tau=0}^{t-1} \frac{z_{k2}^{nm\tau}}{w_k} + (1-B)$$

Then problem (7) is a mixed-integer program with all linear constraints, fractional decision variables z_{ka}^{ijt}, r_{ka}^{ij}, and binary variables s_k^m. To solve the problem, we relax each expert migration decision variable $s_k^m \in \{0,1\}$ to a fractional value, i.e., $0 \leq \hat{s}_k^m \leq 1$, use a standard linear programming algorithm such as the interior point method to obtain a solution in polynomial time, and then convert the fractional solution to an approximate binary solution.

IV. Algorithm

We now design an efficient algorithm to convert the fractional expert migration decisions of problem (7) to binary solutions, and schedule resources and execution of computation and communication mini-tasks in each training iteration of the MoE layer. Our DEMI scheduling algorithm is given in Alg. 1.

Given the optimal fractional solution of expert migration $\hat{s}_k^m$ to the relaxed linear program (7), we use $\hat{s}_k^m$ as the probability of migrating expert k from worker n ($b_k^n = 1$) to worker m, i.e., we decide s_k^m is 1 with probability $\hat{s}_k^m$ and 0 with probability $1 - \hat{s}_k^m$ (line 1). If any memory constraint in (3a)-(3b) is violated with the above expert migration decisions, we greedily move the expert from the worker whose memory capacity is violated to the most under-loaded worker until all memory constraints are satisfied (line 2), assuming that the total available device memory in our system is relatively large as compared to the total parameter size of experts and the size of tokens. We then compute task workloads r_{ka}^{ij} based on equations (2a) through (2d) using s_k^m (line 3). A priority of each task (k, a, i, j) is decided for task workload scheduling, using the time-slot-weighted total task workload $\sum_{t=0}^{T-1} \omega_t \hat{z}_{ka}^{ijt}$ ($\hat{z}_{ka}^{ijt}$ is the workload of task (k, a, i, j) scheduled to execute in t, as solved by the relaxed linear program (7)) and scaling the workload by $r_{ka}^{ij}/\hat{r}_{ka}^{ij}$ ($\hat{r}_{ka}^{ij}$ is the workload solved by program (7) and r_{ka}^{ij} is the workload of task (k, a, i, j) computed at line 3) to estimate the weighted workload finished till time slot t with integer expert migration decisions s_k^m, i.e., $\xi(k, a, i, j) = r_{ka}^{ij}/\hat{r}_{ka}^{ij} \sum_{t=0}^{T-1} \omega_t \hat{z}_{ka}^{ijt}$ (line 4). A lower value of $\xi(k, a, i, j)$ gives a higher priority as it suggests scheduling the corresponding task in earlier time slots.

We schedule unfinished tasks in each time slot, starting from the first time slot. Specifically, we further maintain a time-slot specific task priority, equaling $\xi(k, a, i, j)$ minus weighted workloads of the task that have been executed so far ($\sum_{\tau=0}^{t-1} w_\tau z_{ka}^{ij\tau}$) and the weighted remaining workload, assuming that all the remaining workload of the task is scheduled in

the current time slot ($w_t \tilde{r}_{ka}^{ij}$) (line 10). The rationale behind this priority setting is to ensure that remaining tasks with larger workloads should be scheduled earlier, since the weights of the time slots are increasing over time.

In each time slot, we schedule unfinished tasks in increasing order of the priority. For task 1 (pre-calc token transmissions) or task 2 (expert migration) without dependency on precedent tasks, they can be scheduled up to total task workload or what the communication capacity in the current time slot allows (lines 16-19). Task 3 of expert training cannot be scheduled before the corresponding task 2 is finished (lines 20-21). If the tokens and experts are both originally on the worker, we can schedule expert training on those tokens (lines 22-24); otherwise, we can schedule expert training on received tokens ($\sum_{\tau=0}^{t-1} z_{k1}^{ji\tau} - z_{k3}^{ij\tau}$) at the respective worker, up to the computation capacity of the worker (lines 25-28). Task 4 of post-calc token transmission can be scheduled on tokens whose expert training has been done ($\sum_{\tau=0}^{t-1} z_{k3}^{ji\tau} - z_{k4}^{ij\tau}$), up to the communication capacity of the respective inter-worker connection (lines 29-33). We continue scheduling the tasks until all tasks are completed. The algorithm produces binary expert migration decisions, $s_k^m \in \{0,1\}$, as well as time and workload execution schedule of communication and computation tasks, z_{ka}^{ijt}, $\forall 0 \le t < T$, $a \in \mathbb{Z}_{1\sim4}, i, j \in \mathbb{M}, k \in \mathbb{K}$.

Alg. 1 is carried out for each MoE layer separately during MoE model training, and adapts expert migration, computation and communication schedules according to the routing decisions of the gating networks in the current training iteration. The MoE layer training process using Alg. 1 proceeds as follows: The gating network decides the experts for processing each batch of tokens. All workers exchange the expert selections of tokens in their batches with other workers, each worker runs Alg. 1, and the same random seed ϕ guarantees that the computed schedules are the same at all workers. Then the obtained expert migration, computation and communication schedules are carried out by each worker for MoE training.

Theoretical Analysis. We next analyze the performance of Alg. 1 in terms of time complexity and optimality as compared to true optimum of the mixed-integer linear program in (7).

Theorem 1. *Alg. 1 produces feasible expert migration and task scheduling strategies in each training iteration in $O(M^2T + KT)$ time, where T is the number of time slots, M is the number of workers and K is the total number of experts.*

Proof. When scheduling tasks, Alg. 1 first checks the dependency (lines 16, 20, 22, 25, 29) and then allocates resources based on the completion ratio of preceding tasks and the available resources (lines 17, 23, 26, 30). Thus all the schedules are valid and Alg. 1 produces feasible expert migration and task scheduling solutions to problem (7).

Alg. 1 iterates through each time slot, which is bounded by T. When scheduling the tasks, Alg. 1 updates the priority for each unscheduled task and iterates through the tasks to check the completion ratio of preceding tasks and schedule available task workloads. Given that every worker sends tokens at most to all other workers, we have at most M^2 mini-tasks for tasks

Algorithm 1: Expert Migration, Computation and Communication Task Scheduling in Each Training Iteration of an MoE Layer (DEMI)

Given: optimal solutions of relaxed linear program of (7): expert migration plan $\hat{s}_k^j \in [0,1]$, workloads $\hat{r}_{ka}^{ij}$ and schedules $\hat{z}_{ka}^{ijt}$, $\forall 0 \le t < T, a \in \mathbb{Z}_{1\sim4}, i, j \in \mathbb{M}$, $k \in \mathbb{K}$, random seed ϕ.

// Convert expert migration decision $\hat{s}_k^m$ to integer, $\forall j \in \mathbb{M}, k \in \mathbb{K}$

1 Set $s_k^m = 1$ with probability $\hat{s}_k^m$ and $s_k^m = 0$ otherwise.

2 Adjust s_k^m if the memory constraint on worker m is violated.

// Adjust task workload $\hat{r}_{ka}^{ij}$ based on s_k^m

3 Compute r_{ka}^{ij} using (2a)-(2d) and s_k^m, $\forall a \in \mathbb{Z}_{1\sim4}, i, j \in \mathbb{M}, k \in \mathbb{K}$.

// Set priority for each task. Lower $\xi(\cdot)$ gives higher priority.

4 $\xi(k,a,i,j) := \frac{r_{ka}^{ij}}{\hat{r}_{ka}^{ij}} \sum_{t=0}^{T-1} \omega_t \hat{z}_{ka}^{ijt}$, $\forall a \in \mathbb{Z}_{1\sim4}, i, j \in \mathbb{M}, k \in \mathbb{K}$

5 $S :=$ set of tasks (k,a,i,j) with workloads $r_{ka}^{ij} > 0$, $\forall a \in \mathbb{Z}_{1\sim4}, i, j \in \mathbb{M}, k \in \mathbb{K}$.

// Initialize the variables for allocating resources for each task

6 $z_{ka}^{ijt} = 0, \forall a \in \mathbb{Z}_{1\sim4}, i, j \in \mathbb{M}, k \in \mathbb{K}, 0 \le t \le T$

// Initialize the remaining workload for every tasks

7 $\tilde{r}_{ka}^{ij} = r_{ka}^{ij} \forall a \in \mathbb{Z}_{1\sim4}, i, j \in \mathbb{M}, k \in \mathbb{K}$

8 $t := 0$ // time slot index

9 **while** *S is not empty* **do**

10 Update priority of tasks in S with $\xi_t(k,a,i,j) := \xi(k,a,i,j) - (\sum_{\tau=0}^{t-1} w_\tau z_{ka}^{ij\tau}) - w_t \tilde{r}_{ka}^{ij}$

11 Sort tasks (k,a,i,j) in S in increasing order of priority $\xi_t(k,a,i,j)$.

12 *// Initialize the remaining resources for each time slot t*

13 $\tilde{Z}_{ij}^{t,\text{comm}} = \mathbb{Z}_{ij}^{t,\text{comm}} \forall a \in \mathbb{Z}_{1\sim4}, i, j \in \mathbb{M}, k \in \mathbb{K}, 0 \le t \le T$

14 $\tilde{Z}_{j}^{t,\text{comp}} = \mathbb{Z}_{ij}^{t,\text{comp}} \forall a \in \mathbb{Z}_{1\sim4}, i, j \in \mathbb{M}, k \in \mathbb{K}, 0 \le t \le T$

15 **for** *task (k,a,i,j) in S* **do**

16 **if** *a = 1 or a = 2* **then**

// Task 1 and Task 2 have no dependencies

// Allocate workload up to total task workload or communication capacity

17 $z_{ka}^{ijt} := \min\{\tilde{Z}_{ij}^{t,\text{comm}}, \tilde{r}_{ka}^{ij}\}$

18 $\tilde{r}_{ka}^{ij} := \tilde{r}_{ka}^{ij} - z_{ka}^{ijt}$

19 $\tilde{Z}_{ij}^{t,\text{comm}} := \tilde{Z}_{ij}^{t,\text{comm}} - z_{ka}^{ijt}$

20 **else if** *a = 3 and $b_k^j \neq 1$ and $r_{k2}^{ij} \neq 0$* **then**

// expert migration task is unfinished

21 continue

22 **else if** *a = 3 and $b_k^j = 1$ and $i = j$* **then**

// tokens and experts are already on worker m

23 $z_{k3}^{ijt} := \min\{\tilde{Z}_{j}^{t,\text{comp}}, \tilde{r}_{k3}^{ij}\}$

24 $\tilde{Z}_{j}^{t,\text{comp}} := \tilde{Z}_{j}^{t,\text{comp}} - z_{k3}^{ijt}$

25 **else if** *a = 3* **then**

// Task 3 depends on Task 1's progress

26 $z_{k3}^{ijt} := \min\{\tilde{Z}_{j}^{t,\text{comp}}, \sum_{\tau=0}^{t-1}(z_{k1}^{ji\tau} - z_{k3}^{ij\tau})\}$

27 $\tilde{r}_{k3}^{ij} := \tilde{r}_{k3}^{ij} - z_{k3}^{ijt}$

28 $\tilde{Z}_{j}^{t,\text{comp}} := \tilde{Z}_{j}^{t,\text{comp}} - z_{k3}^{ijt}$

29 **else**

// Task 4 depends on Task 3's progress

30 $z_{k4}^{ijt} := \min\{\tilde{Z}_{ij}^{t,\text{comm}}, \sum_{\tau=0}^{t-1}(z_{k3}^{ji\tau} - z_{k4}^{ij\tau})\}$

31 $\tilde{r}_{k4}^{ij} := \tilde{r}_{k4}^{ij} - z_{k4}^{ijt}$

32 $\tilde{Z}_{ij}^{t,\text{comm}} := \tilde{Z}_{ij}^{t,\text{comm}} - z_{k4}^{ijt}$

33 **end**

34 **if** *$\tilde{r}_{ka}^{ij}$ = 0* **then**

35 Remove task (k,a,i,j) from set S.

36 **end**

37 **if** *$\tilde{Z}_{ij}^{t,comm} = 0$ and $\tilde{Z}_{j}^{t,comp} = 0$* **then**

38 break

39 **end**

40 **end**

41 $t = t + 1$ // move to the next time slot

42 **end**

Return: Expert migration decisions $s_k^j \in \{0,1\}$, $\forall j \in \mathbb{M}$, $k \in \mathbb{K}$ and scheduling of communication and computation tasks $z_{ka}^{ijt}, \forall 0 \le t < T$, $a \in \mathbb{Z}_{1\sim4}$, $i, j \in \mathbb{M}, k \in \mathbb{K}$

1, 3 and 4. For expert migration, each expert can be migrated to at most one worker; therefore, the total number of mini-tasks for task 2 is bounded by K. Since we handle mini-task scheduling in each time slot, the time complexity of Alg. 1 is $O(M^2T + KT)$. □

Theorem 1 guarantees that the time complexity of Alg. 1 is upper-bounded by a small constant ratio of the MoE communication (which is proportional to MK). Since computation is much faster than communication, the running time of Alg. 1 has minimal impact on training time.

Theorem 2. *Per-iteration MoE layer training time achieved by Alg. 1 is upper bounded by a constant ratio of the training time by optimal solutions of integer linear problem (7).*

Proof. We prove that the number of time slots needed to complete all tasks by the optimal solution OPT of integer linear problem (7) is lower-bounded by the ideal OPT_LB defined in (9). The lower bound is calculated as the minimum time required to complete all tasks without considering their dependencies.

$$\begin{aligned}\text{OPT} \geq & \max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k1}^{ij} + r_{k2}^{ij} + r_{k4}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil + \max_{m\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}, i\in\mathbb{M}} r_{k3}^{im}}{\mathbb{Z}_{m}^{\text{comp}}} \rceil \\ \triangleq & \text{OPT\_LB}\end{aligned} \tag{9}$$

On the other hand, the number of time slots used by Alg. 1, i.e., ALG, is upper bounded by ALG_UB defined in (10). The upper bound is reached when all tasks are executed sequentially: tokens are first transmitted to the workers using $\max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k1}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil$ time slots; then expert migration is done using $\max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k2}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil$ time slots; next, workers carry out expert training and use $\max_{m\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}, i\in\mathbb{M}} r_{k3}^{im}}{\mathbb{Z}_{m}^{\text{comp}}} \rceil$ time slots to finish the computation; processed tokens are then sent back to the workers within $\max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k4}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil$ time slots.

$$\begin{aligned}\text{ALG} \leq & \max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k1}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil + \max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k2}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil \\ & + \max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k4}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil + \max_{m\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}, i\in\mathbb{M}} r_{k3}^{im}}{\mathbb{Z}_{m}^{\text{comp}}} \rceil \\ \triangleq & \text{ALG\_UB}\end{aligned} \tag{10}$$

Then we can derive that the ratio of the training time with our algorithm as compared to OPT, is upper-bounded by 3.

$$\begin{aligned}&\frac{\text{ALG}}{\text{OPT}} \leq \frac{\text{ALG\_UB}}{\text{OPT\_LB}} \\ =&(\max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k1}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil + \max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k2}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil \\ &\quad + \max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k4}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil + \max_{m\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}, i\in\mathbb{M}} r_{k3}^{im}}{\mathbb{Z}_{m}^{\text{comp}}} \rceil) \\ &/(\max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k1}^{ij} + r_{k2}^{ij} + r_{k4}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil + \max_{m\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}, i\in\mathbb{M}} r_{k3}^{im}}{\mathbb{Z}_{m}^{\text{comp}}} \rceil) \\ <&(\max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k1}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil + \max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k2}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil + \max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k4}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil) \\ &/(\max_{i,j\in\mathbb{M}} \lceil \frac{\sum_{k\in\mathbb{K}} r_{k1}^{ij} + r_{k2}^{ij} + r_{k4}^{ij}}{\mathbb{Z}_{ij}^{\text{comm}}} \rceil) \\ \leq & 3\end{aligned}$$

□

V. Performance Evaluation

A. Methodology

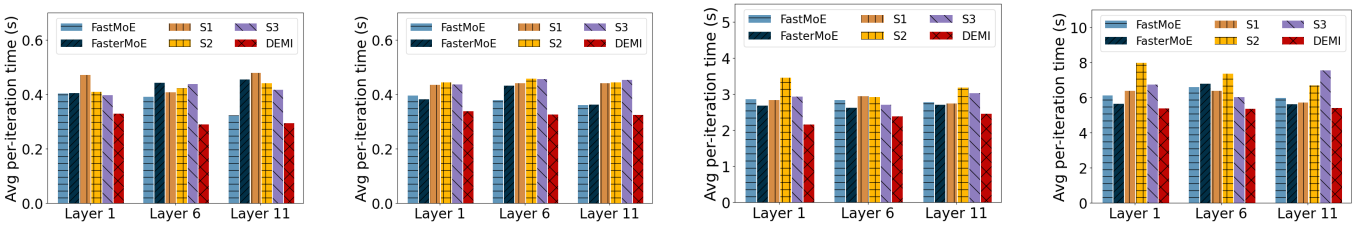
Experimental setup We implement our MoE training system on PyTorch 2.3.1 using Python 3.11.5 on Ubuntu 22.04 with CUDA 12.5 and NCCL 2.20. We evaluate our system on a cluster with four servers including two V100 servers and two A100 servers, featuring 100 Gbps intra-server and 50 Gbps inter-server connections, to assess performance across heterogeneous GPUs and bandwidths. Each V100 server is equipped with four NVIDIA V100 GPUs, and each A100 server holds four A100 GPUs.

We run MoE versions of a 12-layer Transformer-XL model [21] where the feed-forward layer in each Transformer block is replaced with an MoE layer. The Transformer-XL model has a hidden dimension size of 500 and 10 attention heads. Unless otherwise stated, the expert network is constructed with two feed-forward layers, with a hidden size of 1000 and a ReLU activation function [30]. The size of each expert network is 7.64MB. The overall Transformer-XL model with 16 experts has 5.94GB parameters, and reaches 11.67GB with 128 experts. The standard top-2 gating mechanism is used. We run one worker process on one GPU, and keep the number of experts the same across different workers for baselines without expert migration, with the number of experts on each worker varying from 4 to 16 in different experiments.

We experiment under four main settings: (1) 16 experts on four workers on one V100 server, (2) 64 experts on four workers on one V100 server, (3) 128 experts on eight workers spanning one V100 server and one A100 server, and (4) 128 experts on 16 workers spanning two V100 servers and two A100 servers. We run model training on the Enwik8 [31] dataset. The batch size is 200 sequences and input sequence length is 20 tokens for experiments with 16 experts and 64 experts. For experiments with 128 experts, we use an input sequence length of 200 tokens. We train the model using the Adam optimizer [32], with an initial learning rate of 2.5e-4 and a cosine learning rate scheduler.

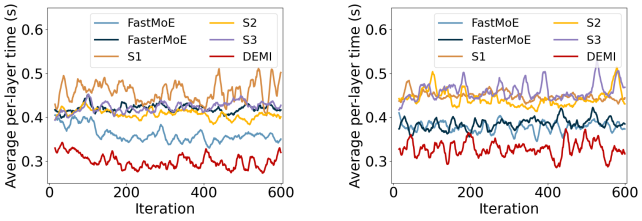
Hyper-parameters We set the number of time slots, T, in our DEMI algorithm to 5. By default, the weights of the time slots are set as $\omega_t = 2^t$. We set large limits for token capacity and parameter capacity on the workers to make the experiments comparable with baselines: the token capacity is set to half of the total token size, e.g., $C_\cdot = 20 \times 200 \times 2 \times 16/2 = 64,000$ times the token dimension for experiments with 16 experts; the parameter capacity is set to half of the total parameter size of all experts, e.g., $Q_\cdot = 8w_k$ for experiments with 16 homogeneous experts. By default, we may conduct expert migration in every training iteration.

Metrics. We evaluate the per-iteration training time of each MoE-layer in the model. We also measure a Workload Imbalance Degree, D, for each MoE layer, defined as the square root of the sum of squares of workloads of all workers, normalized by the total workloads of all workers: $D = \frac{\sqrt{\sum_{i=1}^{M}(F_i/C_i)^2}}{\sum_{i=1}^{M}(F_i/C_i)}$, where C_i and F_i are the workload capacity and allocated workload on worker i, respectively, and workload is measured



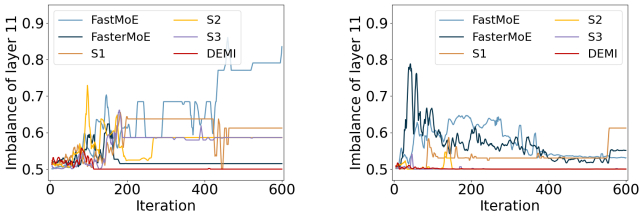
(a) 16 experts (4 workers) (b) 64 experts (4 workers) (c) 128 experts (8 workers) (d) 128 experts (16 workers)

Figure 5: Per-iteration time of different MoE layers



(a) 16 experts (4 workers) (b) 64 experts (4 workers)

Figure 6: Average per-layer iteration training time (presented with rolling average of 15 training iterations)



(a) 16 experts (4 workers) (b) 64 experts (4 workers)

Figure 7: Workload imbalance of penultimate MoE layer

as the size of tokens computed with experts on that worker. A smaller D indicates more balanced workloads across workers. When the workloads are perfectly balanced, i.e., $F_1/C_1 = F_i/C_i, \forall i \in \mathbb{M}$, D is $\frac{\sqrt{M(F_1/C_1)^2}}{M(F_1/C_1)} = 1/\sqrt{M}$. When the workloads are seriously unbalanced, with one worker (e.g., worker 1) taking all the workloads, D is $\frac{\sqrt{(F_1/C_1)^2}}{F_1/C_1} = 1$.

Baselines We compare our algorithm with the following:

- FastMoE [22], which evenly distributes the experts to workers at the beginning of training and does not change expert placement during training.
- FasterMoE [17], which dynamically decides whether to shadow one expert (the expert with the maximum total number of tokens) to all workers in each training iteration. We do not use its topology-aware gates for fair comparison, given that the change of gates affects model accuracy.
- Expert migration strategy S1 that minimizes the total number of tokens transmitted across workers, by placing expert k on the worker receiving the largest number of tokens for the expert, i.e., worker $\arg\max_i d_k^i$.
- Expert migration strategy S2 that minimizes the total amount of token transmission and expert migration traffic, by solving an optimization problem $\min \sum_{m\in\mathbb{M}} \sum_{k\in\mathbb{K}(m)} \sum_{i\in\mathbb{M}, i\neq m} (d_k^i s_k^m + w_k s_k^i)$ under constraint (1).
- Expert migration strategy S3 that minimizes both the total token transmission and expert migration traffic and workload imbalance among workers, by finding optimal expert migration decisions s_k^i that minimize $\sum_{m\in\mathbb{M}} \sum_{k\in\mathbb{K}(m)} \sum_{i\in\mathbb{M}, i\neq m} (d_k^i s_k^m + w_k s_k^i) + \max_{m\in\mathbb{M}} \sum_{k\in\mathbb{K}, i\in\mathbb{M}} d_k^i s_k^m$, where $\max_{m\in\mathbb{M}} \sum_{k\in\mathbb{K}, i\in\mathbb{M}} d_k^i s_k^m$ is the maximum workload among workers, without considering the dependency between computation and communication in MoE training. S3 extends the strategy in SmartMoE [26] by considering both workload balance and transmission time of tokens and experts.

B. Training Speedup

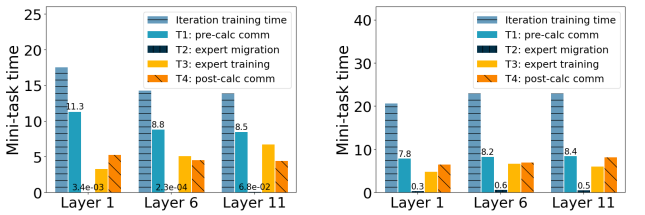
We train the model under four settings for 600 iterations. Fig. 5 shows the iteration time of the first, middle and penultimate MoE layer in the model, and our method achieves greater speed-up in later layers for settings (1) and (2) (1.62× and 1.4× speed-up compared to baselines for penultimate MoE layer with 16 experts and 64 experts, respectively). This is because workload imbalance is more significant in later layers (Fig. 1). Under settings (3) and (4), the speedup is similar for different layers, and our method achieves on average 1.2× speedup on two servers with eight workers and 1.42× speedup on four servers with 16 workers. Fig. 6 further shows per-iteration training time of each MoE layer, averaged over the 12 MoE layers, under settings (1) and (2). Our dynamic expert migration reduces 34.6% and 28.5% of training time compared to baselines under 16 experts and 64 experts, respectively.

C. Workload balance among workers

Fig. 7 presents the Workload Imbalance Degree for the penultimate layer under settings (1) and (2) (homogeneous workers). Our algorithm achieves, on average, 1.3× and 1.14× better workload balance among workers as compared to baselines under 16 and 64 experts, respectively. We also observe that our algorithm may incur slightly biased workloads at the start of training, but maintain workload balance after around 120 and 50 iterations under 16 and 64 experts, respectively. This is mainly due to significant changes in token distribution among experts in the initial stage of training. By considering expert migration and token transmission time together, our algorithm might migrate only parts of the experts, thus not making the workload completely balanced.

D. Time breakdown of mini-tasks

Fig. 8 shows the time taken for each mini-task in model training under settings (1) and (4), where the time is presented as the ratio to the execution time of Alg. 1 for computing the schedules. Especially, the time is recorded on the worker



(a) 16 experts on 4 workers (b) 128 experts on 16 workers

Figure 8: Time breakdown of mini-tasks

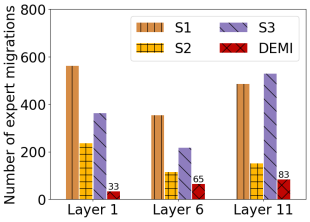


Figure 9: Number of expert migrations (16 experts)

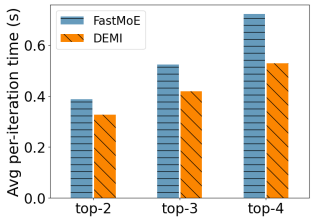
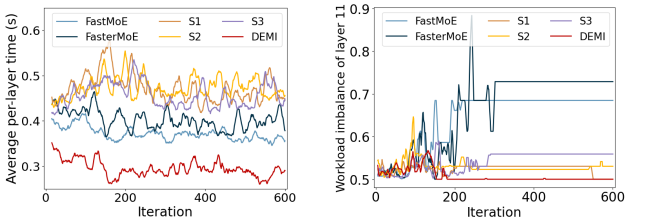


Figure 10: Different top-k gates (64 experts)



(a) Average per-iteration time (b) Workload Imbalance

Figure 11: Heterogeneous experts (16 experts)

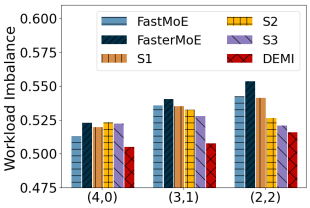


Figure 12: Heterogeneous workers (16 experts)

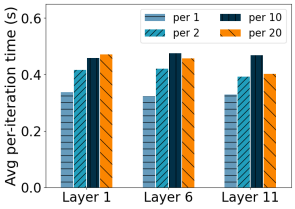


Figure 13: Different migration frequencies (64 experts)

with rank 1. We find that task 1 occupies most of the time at worker 1. When executing task 1, a worker needs to wait for the completion of token transmissions from other workers before starting its task 3; for task 4, the worker only needs to receive all local tokens back from other workers. Worker 1 might receive more tokens for processing but sends fewer tokens to other experts for processing. Task 2 has a relatively short makespan due to infrequent expert migration. Task 3 has shorter duration compared to Task 1 and can partially overlap with Tasks 1, 2, and 4. We also observe that the overhead of running Alg. 1 is less than 7% in each iteration, as shown by the ratio of the iteration time to Alg. 1's computation time.

E. Number of expert migrations

Fig. 9 shows the total number of expert migrations in different MoE layers over 500 training iterations under setting (1). Compared to S1, S2, and S3, our algorithm reduces the number of expert migrations by 83%, 58%, and 73%, respectively, by considering the whole time span of token transmission, expert migration and expert training in decision making. We also observe that there are more expert migrations in the later layers compared to the front layers. This is because changes in token expert selections in the front layers can affect layer computation results, which in turn changes expert selection of tokens in later layers. This makes the "hot" experts change more frequently in later layers.

F. Varying hyper-parameters

We adjust the gating mechanism in each MoE layer under setting (2). Fig. 10 shows that selecting more experts for each token (top-3 or top-4) increases MoE layer training time sublinearly and our method achieves 1.19× (top-2), 1.25× (top-3), and 1.37× (top-4) speedup as compared to FastMOE.

To evaluate heterogeneous experts, we adopt another type of experts, each with 3 layers, a hidden size of 1000 and a ReLU activation. Under setting (1), we adopt eight 2-layer experts and eight 3-layer experts in each MoE layer. Fig. 11 shows that our algorithm reduces the training time by 37.5% and achieves up to 1.31× better workload balance.

We then adjust the memory capacities C_m of workers to render heterogeneous workers under setting (1). We compare the workload imbalance degree of our method with baselines in three different settings of heterogeneous workers in Fig. 12: (a) 4 fast workers each with 14,000 token capacity for each MoE layer; (b) 3 fast workers each with 14,000 token capacity and 1 slow worker with 7,000 token capacity; (c) 2 fast workers each with 14,000 token capacity and 2 slow workers each with 7,000 token capacity. We find that our method reduces the workload imbalance by up to 23% compared to baselines.

We further evaluate the impact of reduced expert migration frequency under experimental setting (2), by running Alg. 1 once every 2, 10 or 20 training iterations. Fig. 13 shows that with less frequent expert placement adjustments, the average iteration time increases for the front layers, and the time first increases and then decreases for the middle and last layers. The latter is possibly due to the reduction in scheduling time and expert migration time, which partially compensates for the increased time spent on unbalanced workloads.

VI. Conclusion

This paper designs an efficient dynamic expert migration and computation-communication co-scheduling algorithm for distributed Mixture-of-Experts (MoE) model training. We abstract MoE layer training into mini-tasks including token transmission between workers, expert parameter transmission, and expert computation, and model dependencies between computation and communication mini-tasks. We design an efficient algorithm to minimize completion time of all mini-tasks. Extensive experimental evaluation demonstrates that our method achieves significant efficiency improvements over state-of-the-art baselines. Our approach reduces the training time of MoE layers by 34.6% under homogeneous expert setting and 37.5% under heterogeneous expert setting. Additionally, our method achieves 1.3× better workload balance among workers, leading to more efficient parallelization and resource utilization.

ACKNOWLEDGMENT

This work was supported in part by grants from Hong Kong RGC under the contracts 17204423 (GRF), C7004-22G (CRF), C5032-23G (CRF), T43-513/23-N (TRS) and CRS_PolyU501/23.

REFERENCES

[1] T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, and et al., "Gemma: Open models based on gemini research and technology," *CoRR*, vol. abs/2403.08295, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.08295

[2] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, and et al, "Palm: Scaling language modeling with pathways," *J. Mach. Learn. Res.*, vol. 24, pp. 240:1–240:113, 2023. [Online]. Available: http://jmlr.org/papers/v24/22-1144.html

[3] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, and et al, "Glam: Efficient scaling of language models with mixture-of-experts," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, vol. 162. PMLR, 2022, pp. 5547–5569.

[4] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation AI scale," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, vol. 162. PMLR, 2022, pp. 18 332–18 346.

[5] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[6] Z. Xue, G. Song, Q. Guo, B. Liu, Z. Zong, Y. Liu, and P. Luo, "RAPHAEL: text-to-image generation via large mixture of diffusion paths," *CoRR*, vol. abs/2305.18295, 2023.

[7] OpenAI, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[8] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, E. Zheng, R. Child, R. Y. Aminabadi, J. Bernauer, X. Song, M. Shoeybi, Y. He, M. Houston, S. Tiwary, and B. Catanzaro, "Using deepspeed and megatron to train megatron-turing NLG 530b, A large-scale generative language model," *CoRR*, vol. abs/2201.11990, 2022.

[9] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[10] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *J. Mach. Learn. Res.*, vol. 23, pp. 120:1–120:39, 2022.

[11] C. Chen, M. Li, Z. Wu, D. Yu, and C. Yang, "Ta-moe: Topology-aware large scale mixture-of-expert training," in *NeurIPS*, 2022.

[12] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, and et al, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[13] J. Li, Y. Jiang, Y. Zhu, C. Wang, and H. Xu, "Lita: Accelerating distributed training of sparsely activated models," *CoRR*, vol. abs/2210.17223, 2022.

[14] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[15] Y. Zhou, T. Lei, H. Liu, N. Du, Y. Huang, V. Zhao, A. M. Dai, Z. Chen, Q. V. Le, and J. Laudon, "Mixture-of-experts with expert choice routing," in *NeurIPS*, 2022.

[16] M. Lewis, S. Bhosale, T. Dettmers, N. Goyal, and L. Zettlemoyer, "BASE layers: Simplifying training of large, sparse models," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 139. PMLR, 2021, pp. 6265–6274.

[17] J. He, J. Zhai, T. Antunes, H. Wang, F. Luo, S. Shi, and Q. Li, "Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models," in *PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022*. ACM, 2022, pp. 120–134.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017, pp. 5998–6008.

[19] G. Jawahar, S. Mukherjee, X. Liu, Y. J. Kim, M. Abdul-Mageed, L. V. S. Lakshmanan, A. H. Awadallah, S. Bubeck, and J. Gao, "Automoe: Heterogeneous mixture-of-experts with adaptive computation for efficient neural machine translation," in *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, A. Rogers, J. L. Boyd-Graber, and N. Okazaki, Eds. Association for Computational Linguistics, 2023, pp. 9116–9132.

[20] Y. Zhou, T. Lei, H. Liu, N. Du, Y. Huang, V. Zhao, A. M. Dai, Z. Chen, Q. V. Le, and J. Laudon, "Mixture-of-experts with expert choice routing," in *NeurIPS*, 2022.

[21] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," in *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pp. 2978–2988. [Online]. Available: https://doi.org/10.18653/v1/p19-1285

[22] J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang, "Fastmoe: A fast mixture-of-expert training system," *CoRR*, vol. abs/2103.13262, 2021.

[23] J. Li, Y. Jiang, Y. Zhu, C. Wang, and H. Xu, "Accelerating distributed moe training and inference with lina," in *Proceedings of the 2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*, J. Lawall and D. Williams, Eds. USENIX Association, 2023, pp. 945–959. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/li-jiamin

[24] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[25] J. Liu, J. H. Wang, and Y. Jiang, "Janus: A unified distributed training framework for sparse mixture-of-experts models," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ser. ACM SIGCOMM '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 486–498. [Online]. Available: https://doi.org/10.1145/3603269.3604869

[26] M. Zhai, J. He, Z. Ma, Z. Zong, R. Zhang, and J. Zhai, "Smartmoe: Efficiently training sparsely-activated models through combining offline and online parallelization," in *2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*. USENIX Association, 2023, pp. 961–975.

[27] S. Shi, X. Pan, Q. Wang, C. Liu, X. Ren, Z. Hu, Y. Yang, B. Li, and X. Chu, "Schemoe: An extensible mixture-of-experts distributed training system with tasks scheduling," in *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM, 2024, pp. 236–249.

[28] S. Zhang, L. Diao, C. Wu, S. Wang, and W. Lin, "Accelerating large-scale distributed neural network training with SPMD parallelism," in *Proceedings of the 13th Symposium on Cloud Computing, SoCC 2022, San Francisco, California, November 7-11, 2022*. ACM, 2022, pp. 403–418.

[29] C. Jiang, Y. Tian, Z. Jia, S. Zheng, C. Wu, and Y. Wang, "Lancet: Accelerating mixture-of-experts training via whole graph computation-communication overlapping," in *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*, P. B. Gibbons, G. Pekhimenko, and C. D. Sa, Eds. mlsys.org, 2024.

[30] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[31] M. Hutter, "The human knowledge compression contest," 2012.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980