Janus: A Unified Distributed Training Framework for Sparse Mixture-of-Experts Models

Juncai Liu
Tsinghua University
liujc19@tsinghua.org.cn

Jessie Hui Wang
Tsinghua University and
Zhongguancun Laboratory
jessiewang@tsinghua.edu.cn

Yimin Jiang
ByteDance
jymthu@gmail.com

ABSTRACT

Scaling models to large sizes to improve performance has led a trend in deep learning, and sparsely activated Mixture-of-Expert (MoE) is a promising architecture to scale models. However, training MoE models in existing systems is expensive, mainly due to the All-to-All communication between layers.

All-to-All communication originates from *expert-centric* paradigm: keeping experts in-place and exchanging intermediate data to feed experts. We propose the novel *data-centric* paradigm: keeping data in-place and moving experts between GPUs. Since experts' size can be smaller than the size of data, data-centric paradigm can reduce communication workload. Based on this insight, we develop Janus. First, Janus supports fine-grained asynchronous communication, which can overlap computation and communication. Janus implements a hierarchical communication to further reduce cross-node traffic by sharing the fetched experts in the same machine. Second, when scheduling the "fetching expert" requests, Janus implements a topology-aware priority strategy to utilize intra-node and inter-node links efficiently. Finally, Janus allows experts to be prefetched, which allows the downstream computation to start immediately once the previous step completes.

Evaluated on a 32-A100 cluster, Janus can reduce the traffic up to 16× and achieves up to 2.06× speedup compared with current MoE training system.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Distributed training, mixture of experts, deep learning

ACM Reference Format:
Juncai Liu, Jessie Hui Wang, and Yimin Jiang. 2023. Janus: A Unified Distributed Training Framework for Sparse Mixture-of-Experts Models. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3603269.3604869

ACM SIGCOMM '23, September 10–14, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09.
https://doi.org/10.1145/3603269.3604869

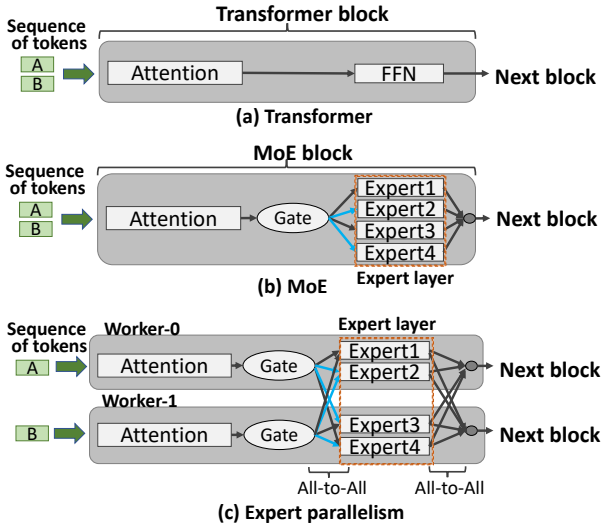


Figure 1: Transformer, MoE and Expert parallelism (blue arrows represent assignment of tokens to experts)

1 INTRODUCTION

Recent trends in deep learning have shown that the capacity of a DNN model is typically improved as its number of parameters increases [7, 11], thus leading to better model quality. Based on this observation, giant models with billions of parameters are often used in the frontiers of Computer Vision (CV) and Natural Language Processing (NLP) [15, 22]. While the improvement of large models' quality (e.g., accuracy) is significant, the computation cost of training them, however, is extremely high. This hinders large models from being more widely adopted.

Following the basic principle of using massive parameters while preserving constant computation cost, sparsely activated models have recently been introduced. Among them, the Mixture-of-Experts (MoE) structure is now one of the most popular ways to implement sparse activation [15, 33] since it does not degrade model capacity like quantization and knowledge distillation. As demonstrated in Figure 1(b), for each input, instead of activating all parameters (i.e. all of experts, which are Feed-Forward Networks typically), an MoE model intentionally selects just a few of them for computation. This leads to sub-linear scaling of FLOPS needed with model size. Recent literature has proven the potential of sparse MoE models versus their dense counterparts [5].

Despite its significant potential in increasing model quality, training a large sparse MoE model is still non-trivial. As the number of

experts increases, the model eventually will exceed the memory limit of a single GPU. To train large MoE models, *expert parallelism* is proposed. As demonstrated in Figure 1(c), the expert layer in an MoE block is split and placed on multiple GPUs. Therefore, during model computation, the All-to-All communication has to be added before and after the expert layer to exchange the intermediate results between all GPUs. Unfortunately, the All-to-All operation is known to be expensive [19] and becomes the primary bottleneck in MoE training.

Given the fact that All-to-All communication is time-consuming during model training, we take a step back and rethink the essential communication pattern during the training of MoE models. Existing training systems keep experts in-place and exchange intermediate data to feed the experts. We name this paradigm as *expert-centric*, where experts are statically placed while intermediate data is moved. On the contrary, *one can also keep intermediate data static and move experts between GPUs*. For simplicity, we name this paradigm as *data-centric* [1], which is a novel yet equivalent way to implement the necessary communication for training MoE models.

Comparing the current *expert-centric* paradigm and the newly presented *data-centric* paradigm, we find that the latter has an advantage on reducing communication workload. The communication size of transferring experts, under certain conditions, could be much smaller than the expert-centric way. This gap can be enlarged when the model dimension is relatively small while the amount of input data is large (e.g., large batch size). The communication workload also becomes balanced among workers since each expert is of the same size. Besides, data-centric paradigm potentially enables two opportunities.

- First, unlike the expert-centric paradigm whose All-to-All communication is strictly blocking (synchronous), moving experts between GPUs can be achieved using non-blocking (or asynchronous) communication primitives such as *push* and *pull*. This further boosts performance since we no longer need to explicitly synchronize all workers (GPUs) during an iteration. Besides, the expert pulled by a worker can be reused by other workers in the same machine. The reusability provides an opportunity to reduce cross-machine traffic.
- Second, since expert weights are deterministic during an iteration of the model training, prefetching experts is possible and it allows us to further improve the overlap between computation and communication. Prefetching technique is not applicable in the expert-centric paradigm, since intermediate data can only be populated on the fly.

While data-centric is a promising paradigm for accelerating MoE model training, as noted before, we find this idea is not revealed in any popular MoE frameworks such as DeepSpeed [31] and Tutel [17]. One possibility is that the data-centric based communication only outperforms the expert-centric way under certain model configs. Hence, we holistically combine the two mechanisms and design *Janus*[2], an MoE training system that is communication-optimal.

Inspired by the opportunities provided by data-centric paradigm, Janus improves system efficiency in the data-centric paradigm from three perspectives. First, Janus treats the request of fetching each expert as an individual task, instead of sending expert requests in batches. In this way, Janus enables asynchronous communication, *i.e.*, a worker can execute the computation of one expert while receiving another expert at the same time, which means parts of the communication time can be hidden by the computation time of experts. Besides, it also enables Janus to implement a hierarchical fetch operation, *i.e.*, requests for the same expert from workers in the same machine can be merged and thus the cross-node traffic is further reduced. Second, to alleviate the contention of bandwidth of bottlenecks, Janus designs a topology-aware priority strategy to carefully arrange the priority of requests of fetching experts. Third, Janus leverages the bandwidth in idle time slots to prefetch experts, which enables that the computation of an expert layer can start immediately once the computation of the former layer completes.

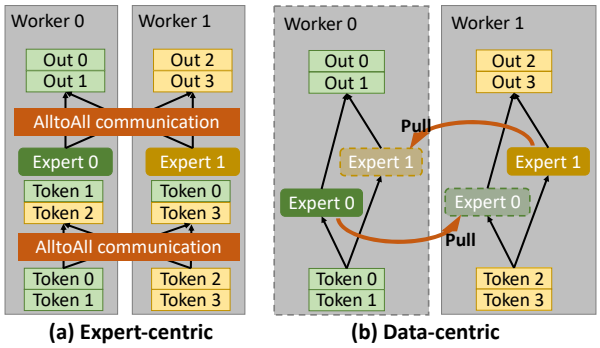


Figure 2: Expert-centric vs Data-centric

Briefly, we make the following contributions:

- We propose a novel communication paradigm for the training of MoE models: data-centric, which could reduce traffic volumes in communication and make the communication workload balanced.
- We schedule the requests of fetching experts in a fine-grained manner to enable asynchronous communication and hierarchical communication, which help us hide communication time and reduce cross-node traffic.
- We design a topology-aware priority strategy to avoid resource contention in intra-node expert exchanges and improve the utilization of the involved links.
- We design a prefetch mechanism, which helps us smooth the bursts of communication traffic and utilize the bandwidth in idle time slots.

We evaluate Janus in a commodity GPU cluster with 32 A100 GPUs. Evaluation results show that Janus can reduce the traffic up to 16× in the All-to-All communication and achieve up to 2.06× speedup on the iteration time in practice, compared with state-of-the-art MoE frameworks.

2 BACKGROUND

2.1 Transformer and MoE Model

Transformer [36] is the state-of-art structure to process sequence and consists of many blocks. Figure 1(a) shows the structure of a

[1] In the NLP literature, the intermediate data is also termed as *tokens*.
[2] God of gates and transitions, representing the behavior of mixture of experts.

Transformer block. A Transformer block consists of two parts: an Attention layer and a Feed-Forward Network (*i.e.* FFN). Typically, an MoE block can derive from a Transformer block.

MoE model has been proven to have a strong capability in various areas [15][19][8]. An MoE model can consist of both Transformer blocks and MoE blocks. Figure 1(b) shows the structure of an MoE block. There is a gate and an expert layer in each MoE block and there are many experts (typically FFNs) in the expert layer. When a token enters an MoE block and is processed after the *Attention* layer, the gate will allocate it to several experts which are specialized in processing it. The number of allocated experts for each token is controlled by one parameter of the gate, *i.e.*, $topK$.

2.2 Expert Parallelism

The size of an MoE model could be so large that it exceeds the capacity of a single GPU. To train a large-scale MoE model on GPUs, expert parallelism [15][19] was proposed and widely used. Figure 1(c) shows the concept of expert parallelism. In expert parallelism, the expert layer is divided into several parts and allocated to GPUs. Each GPU holds some experts in the expert layer, and different GPUs hold different experts. For other parts of an MoE model (*i.e.*, the Attention layer and the gate in Figure 1(c)), each GPU holds an independent copy.

Current implementation of expert parallelism is expert-centric by default. Figure 2(a) illustrates the procedure of expert-centric training. When an MoE block processes sequences of tokens, the gate needs to assign experts for each token, and the token is distributed by the gate to the GPUs that host the allocated experts. This distribution of tokens to GPUs is completed by an All-to-All communication primitive, as the target GPUs of the tokens generated by a GPU are likely to include all GPUs. After the tokens are processed by their assigned experts, the results need to be sent back to their original GPUs, which requires All-to-All communication again. As an MoE model always has multiple MoE blocks, the training of an MoE model could involve many times of All-to-All communication operations.

3 OBSERVATION AND MOTIVATION

3.1 Observation on Expert-centric Paradigm

In this subsection, we profile the time cost of training MoE models in the current expert-centric system and have the following observations.

The communication workload is heavy and imbalanced. We profile the time cost of training three MoE models in four 8-A100 GPU machines when each MoE block has 32 experts and the time cost of training these models in two 8-A100 GPU machines when each MoE block has 16 experts. The configuration of the models is shown in Table 1. We train a large number of iterations and report the average statistics of these iterations in Figure 3. We can see that on average the latency caused by the All-to-All communication in an iteration occupies 38.5% - 68.4% of the time of a whole iteration. Besides, the numbers of tokens assigned to different experts are imbalanced [24]. All-to-All primitive is a kind of synchronous collective communication, which means that the latency is determined by the busiest worker who needs to send and receive the largest amount of tokens. Therefore, the imbalanced workload has a negative influence on the training time.

Table 1: Configuration of models and Traffic reduction of data-centric paradigm (D.C.) compared with expert-centric paradigm (E.C.)

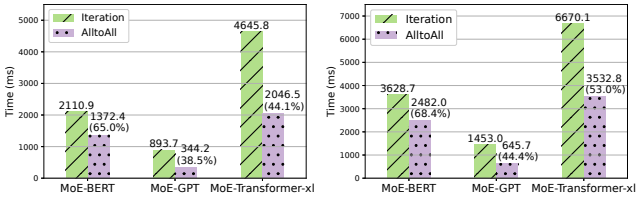
Model	MoE-BERT		MoE-GPT		MoE-Transformer-xl	
Batch size B	256		256		64	
Seq. length S	128		64		512	
Top k in gate	2		4		2	
Expert dim. H	768		768		256	
#MoE block[3]	4		1		12	
#Total block	12		12		12	
#Expert	16	32	16	32	16	32
#GPU	16	32	16	32	16	32
Model size (B)[4]	0.42	0.73	0.23	0.31	0.11	0.21
E.C. Traffic[5](GB)	6	9	1.5	2.25	6	9
D.C. Traffic(GB)	0.56	1.69	0.14	0.42	0.19	0.56

The links between GPUs are heterogeneous. Generally speaking, the GPUs within the same machine are connected by NVlinks, and GPUs in different machines are connected by an RDMA network. We stress test the goodput of the All-to-All primitive on different links in two environments: (1) an 8-GPU machine; (2) four 8-GPU machines connected by an RDMA network. Each machine is equipped with four 200Gbps NICs and each NIC is shared by two GPUs in the machine. The experimental results show the goodput in the former setting reaches 1846.58Gbps while the goodput in the later setting only reaches 101.9Gbps. The intra-machine All-to-All goodput is 18X larger than the inter-machine All-to-All goodput, which suggests that the bandwidth of intra-machine links is not fully utilized during inter-machine All-to-All communication, and the system performance is limited by the bandwidth of inter-machine links.

Cross-GPU links can be underutilized or idle in some time slots. We observe that the links between workers are underutilized or idle in some time slots. This is because no data needs to be exchanged among GPUs when these GPUs are conducting computation of the Attention layer. Once the computation of the Attention layer is completed and the gate has derived the assignment of the tokens to experts, the gate needs to distribute the tokens to the experts, and a traffic burst occurs. If we can do some useful communication in the idle time slots and smooth the bursts, the resource contention can be mitigated and the training process can be accelerated.

All of current MoE training systems adopt the expert-centric paradigm, which means they all have the above issues.

[3]Number of MoE block indicates how many blocks are expanded as MoE blocks among total blocks. For example, MoE-BERT has 12 blocks, among which 4 blocks are expanded as MoE blocks and other 8 blocks are Transformer blocks. The number of MoE block is a hyper-parameter and we set it as 4,1,12 in experiments respectively.
[4]Here, B is the abbreviation of billion, instead of byte.
[5]The traffic is referred to the volume of cross-machine traffic per machine in each iteration in the All-to-All communication. More comprehensive analysis is elaborated in Section 5.1.3.



(a) 16GPU (b) 32GPU

Figure 3: The average latency of a single iteration and the average latency caused by the All-to-All communication in an iteration. (Number in brackets is the percentage of the latency of All-to-All communication.)

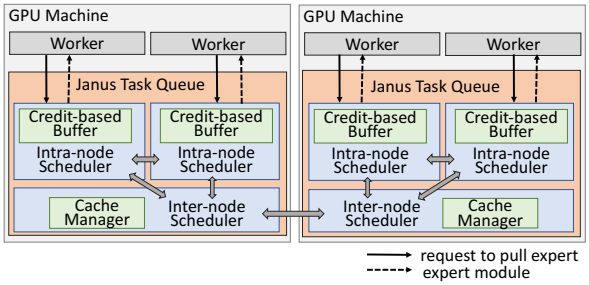


Figure 4: Janus architecture overview.

3.2 Data-centric Paradigm

We notice that the volume of expert modules can be smaller than the volume of tokens. To shorten the time taken by communication, we propose the data-centric paradigm. As Figure 2(b) shows, in our data-centric paradigm, the workers pull experts to the local instead of pushing tokens to other workers. In this way, the communication workload can be reduced, as Table 1 shows. Besides, the computation result in expert-centric paradigm is strictly equivalent to the results in data-centric paradigm. Therefore, *data-centric paradigm does not affect the convergence of training and model accuracy.* In data-centric paradigm, the communication workload also becomes balanced among workers since each expert is of the same size. Besides the advantages of reducing and balancing communication workload, the data-centric paradigm has the following advantages over the expert-centric paradigm.

- **Less synchronization between workers.** In the expert-centric paradigm, the training of MoE blocks requires All-to-All communication while it is synchronous. The All-to-All communication is completed until all of workers receive all of tokens they need. This may cause potential efficiency losses since fast machines have to wait for slow machines. In the data-centric paradigm, all communication is caused by pull operations, and workers do not need to wait for each other when pulling experts. A worker can respond to a pull request and send out the requested expert at any time in an iteration. The communication becomes asynchronous, which indicates that a GPU can proceed its task without the negative influence from other GPUs during an iteration.
- **Data reusability.** In the expert-centric paradigm, tokens sent by a worker to another worker cannot be used by other workers. In the data-centric paradigm, the expert pulled by a worker can be reused by other workers. The reusability provides an opportunity to merge the responses for the pull requests of the same expert and thereby further reducing traffic.
- **Earlier communication.** In the expert-centric paradigm, All-to-All communication happens only after tokens are produced by the previous layer, that is, the model computation reaches the expert layer. In the data-centric paradigm, pull requests of experts can be issued at any time after one iteration starts, *e.g.*, the model computation reaches the first layer. This is because the experts will not be changed within a single iteration.

4 OVERVIEW OF JANUS

Following the principle of the data-centric paradigm, we design Janus. Figure 4 shows the architecture of Janus. The workers are still responsible for executing model computation. When training an MoE model, those experts are divided into several parts and allocated to workers. Each worker is responsible for storing the weights of its experts and update them in each iteration.

When the computation reaches the gate of an MoE block, if communication size of transferring experts is larger than intermediate data. Janus will adopt expert-centric paradigm and call *All-to-All* primitive. Otherwise, Janus will adopt data-centric paradigm, where workers put requests of fetching experts into Janus Task Queue and wait for expert modules. After acquiring experts, workers can execute computation on the expert modules. At the end of an iteration, workers generate gradients of the experts, and they need to ask the Janus Task Queue to send the gradients back to the original workers of these experts to update the weights of the model.

Each machine has a Janus Task Queue, which includes an Inter-node Scheduler and several Intra-node Schedulers (one for each worker). Each Intra-node Scheduler, which is attached to a worker (*i.e.* GPU) and lies in the memory of the GPU, is responsible for receiving requests from its corresponding worker and fetching experts for the worker. It has a component named *Credit-based buffer* to manage the buffer space to save the pulled experts. Inter-node Scheduler, which is located in the CPU memory of the machine, is responsible for fetching experts from other machines upon receiving requests from the intra-node schedulers of the local machine. Besides the communication function, it has a component named *Cache Manager* to cache experts pulled from other machines.

If the requested expert has been pre-allocated to the local machine, Intra-node Scheduler pulls it from the workers who have it. Otherwise, Intra-node Scheduler will pass the request to the Inter-node Scheduler in this machine and wait for the Inter-node Scheduler to return the expert. Inter-node Scheduler will return the expert module from Cache Manager or request from other machines.

Janus Task Queue is the core component in Janus. By appropriately scheduling the requests of fetching experts, it improves the communication efficiency during training MoE models. We briefly introduce the main scheduling strategies here.

First, after receiving requests from workers, the Queue pulls experts one by one in a fine-grained manner instead of pulling all experts simultaneously (5.1). This fine-grained manner enables

that each worker can execute the computation of one expert while receiving other experts at the same time (5.1.1). In this way, parts of communication time can be hidden by the computation time of experts. It also avoids the contention among requested experts and makes some experts arrive earlier. The experts that arrived earlier can be cached in the Cache Manager and shared by workers in the local machine (5.1.2). Besides, the gradient generated from workers in the local machine can be merged in the Inter-node Scheduler before being sent out, which further reduces the cross-node traffic and greatly accelerates communication efficiency.

Second, the Queue arranges the order of requests according to the topology of workers to alleviate the contention of bandwidth of bottlenecks (5.2).

Third, the Queue tries to prefetch experts when possible, which enables that the computation of the expert layer can start immediately once the computation of the former layer completes (5.3). In this way, the underutilized bandwidth in the time slots when GPUs are focusing on computation and no communication is needed is leveraged.

5 SYSTEM DESIGN

5.1 Fine-grained Task Scheduling

In the expert-centric paradigm, a worker has to receive tokens from all different workers before it starts the expert computation. In Janus, each worker usually needs to pull all experts in the expert layer to complete the computation of its tokens on this layer. However, it is unnecessary and infeasible to pull all these experts simultaneously before the computation starts. Once an expert arrives, the computation on this expert can start. Janus splits the request of all non-local experts from a worker into a set of small tasks and only a single expert is required to be pulled in each small task. Thus, Janus can schedule these small tasks in a fine-grained manner.

This fine-grained manner benefits the system in three aspects. First, the computation of an expert can start immediately once the expert is received successfully. Thus, the computation of one expert and the communication of the other expert can overlap, which speeds up the training. Second, it is impossible for the limited GPU memory to host all experts. By requesting experts one by one, we can discard the used experts to host later experts. Third, by scheduling, Janus avoids the resource contention of fetching experts, then some experts can be received earlier and they can be shared with other workers in the local machine. Thus, the cross-node traffic can further be reduced.

In this subsection, we introduce the asynchronous communication mechanism in the Intra-Node Scheduler to implement the first two benefits, and then introduce the hierarchical communication mechanism in the Inter-Node Scheduler to implement the third benefit. At the end of this subsection, we analyze the communication efficiency of Janus mathematically.

5.1.1 Asynchronous Communication Mechanism in Intra-Node Scheduler. In the expert-centric paradigm, All-to-All communication is a kind of synchronous collective communication. As Figure 5(a) shows, in the forward phase, the expert computation (*Ep Comp.* in the figure) in a worker cannot begin until all needed tokens are received. In the backward phase, the gradient computation of

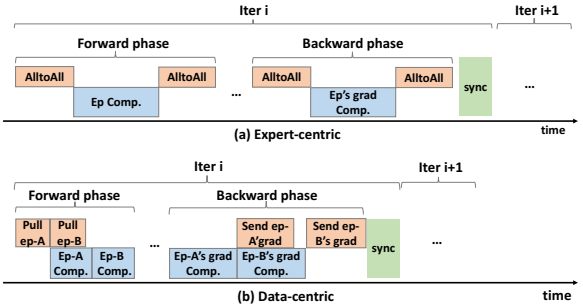


Figure 5: The pipeline of asynchronous communication in fine-grained task scheduling

experts (*Ep's grad Comp.* in the figure) cannot begin either until all needed tokens are received. As for the data-centric paradigm, in the forward phase, workers only need to conduct pull operations for communication among GPUs, and the computation of different experts is independent of each other. After receiving an expert, a worker can perform the expert computation while pulling the next expert at the same time, as Figure 5(b) shows. In the backward phase, a worker can send back the gradient of an expert for update once the gradient is generated, without waiting for generating gradient of other experts in that worker. In this way, the communication and computation can overlap on the timeline and the training time of the MoE block can be shortened. At the end of an iteration, the workers need to synchronize until all expert's weights are updated and then the workers will clear the cache because it is stale. There is no asynchronous operation between workers when updating model with gradients and the asynchronous operation only exists when fetching experts. Thus, *training with asynchronous communication provides the same model accuracy as training with All-to-All communication since no stale cache is used and these two kinds of paradigm are equivalent.*

Specifically, an Intra-Node Scheduler provides a Credit-based buffer to implement the asynchronous communication mechanism. A buffer on the GPU memory is pre-allocated to save the pulled experts, and it has a parameter which is named as *credit*, which represents the number of experts its remaining space can host. Its initial value is set to be C, which is the size of the buffer divided by the size of one expert. A credit is consumed each time a pull operation is requested. When an expert is pulled to the local worker and the expert computation is completed, a credit is released, and the expert is offloaded to the CPU memory and will be reused in backward computation stage by copying it from the CPU memory to the GPU memory. When C credits are used up, all pull tasks are blocked until a new credit is released.

5.1.2 Hierarchical Communication Mechanism in Inter-Node Scheduler. In a multi-machine GPU cluster, an inter-node link (which is an RDMA link going through NIC) has smaller bandwidth than an intra-node link (like NVlink). So we should reduce inter-node traffic as much as possible. Janus implements a hierarchical communication mechanism for this goal.

With the hierarchical communication mechanism, an Inter-Node Scheduler is responsible for gathering and merging the requests for

Table 2: NOTATION LIST

Notation	Description
n	number of machines
m	number of workers per machine
E	number of experts per worker
H	dimension of a token input to an expert
T	number of tokens generated by a worker
k	the gate parameter *topK*
B	batch size of the training task in each worker
S	sequence length of the training task

the same external expert[6]. We explain how the inter-node traffic is reduced by our hierarchical mechanism in the forward phase and the backward phase, respectively.

Forward computation: In an iteration, an Inter-Node Scheduler will pull experts from other machines and store them in its Cache Manager if the requested experts have not been cached. If the requested expert has been in the Cache Manager, the Inter-Node Scheduler will directly return the expert from the Cache Manager. After the iteration is completed, the experts in the cache will be cleared to release space. It can be seen that with the cache mechanism, an external expert pulled by a worker can be shared by other workers in the same machine. In this way, each machine only pulls the external experts once in an iteration.

Backward computation: In the backward phase, all experts have been in local memory of the CPU or the GPUs, so no inter-node communication is needed. However, after the backward computation which generates gradients, the gradients need to be sent back to the original location of the expert to update their weights. Naively, the gradients generated by each GPU will be sent to the target GPU separately, and then the target GPU reduces (i.e. average) the gradients from all workers.

To reduce the inter-node traffic, the inter-node scheduler first waits and accumulates the gradients from all local workers. After the collection for an expert is completed, the gradients of this expert will be pre-reduced and then sent back to the target GPU. In this way, each machine only sends back the gradients of each expert once in an iteration.

5.1.3 Communication Efficiency Analysis. The fine-grained scheduling of Janus enables the above hierarchical communication mechanism, which significantly reduces inter-node traffic. Since the communication bottleneck of the whole system lies in inter-node communication instead of intra-node communication, we take the volume of inter-node traffic as a metric to evaluate the potential communication efficiency of training systems.

In this subsection, using the metrics, we mathematically analyze the efficiency of the data-centric paradigm and its theoretical gains over the expert-centric paradigm. We first analyze the communication volume in the forward phase in detail and then analyze the communication volume in the backward phase briefly since it is similar to the analysis in the forward phase. Some commonly-use notation is listed in Table 2.

[6]Experts stored in the workers in outside machine. Note that in the expert parallelism, the expert layer is split and each worker only has a part of experts.

Data-centric: In the data-centric paradigm, the inter-node traffic is caused by fetching experts from remote machines, so we need to derive the size of an expert.

In MoE model, each expert module is usually a FeedForward Network (FFN) which is composed of two *Linear* layers. For an FFN module, the first layer includes a matrix with a shape of $H * 4H$, and the second layer includes a matrix with a shape of $4H * H$. Therefore, the size of an FFN module is $8H^2$. Each worker has E experts, then a machine has mE experts. Since each machine needs to broadcast these mE experts to other $n-1$ machines, the communication volume in the data-centric paradigm during training an MoE block is

$$Comm_{DC} = 8H^2Em(n-1).$$

Expert-centric: In expert-centric training systems, the token distribution among experts is usually imbalanced. The time needed to complete communication depends on the machine with the largest volume of data to be sent/received. Obviously, the communication time under imbalanced distribution is almost always longer than the case of balanced distribution. [7]

Now we calculate the size of the tokens to be transferred, which is the inter-node traffic volume. Each worker generates T tokens, then a m-worker (GPU) machine can generate mT tokens. Under the assumption of balanced distribution of tokens, there are $\frac{n-1}{n}$ percentages of tokens to be sent to other machines. In the expert-centric paradigm, an MoE block requires two All-to-All communication operations in the forward computation stage. Therefore, the communication volume of expert-centric paradigm in an MoE block is

$$Comm_{EC} = 2mHT * \frac{n-1}{n}.$$

This completes the analysis of the traffic volume in the forward phase.

As for the backward phase, in the expert-centric paradigm, the system needs to transmit all intermediate results required for generating gradients, and this volume is equal to the volume of the tokens it sends in the forward phase. In the data-centric paradigm, the system can reuse the experts pulled and cache in the forward phase. After calculating the gradient of the expert module, the gradient should be sent back to the original worker. The size of gradients is the same as the expert model pulled, and the communication direction is opposite. Besides, multiple gradients of the same expert in a machine are reduced and merged before being sent back. Therefore, in the data-centric paradigm, the traffic volume in the backward phase is also equal to the volume in the forward phase.

Discussion: We define a metric R to evaluate the theoretical gain of our data-centric paradigm, which is the ratio of the inter-node communication volume under two paradigms. Mathematically, we have

$$R = \frac{Comm_{EC}}{Comm_{DC}} = \frac{T}{4nHE}.$$

[7]One extreme case is that one worker sends most of its generated tokens to the workers in the local machine, which is imbalanced but the inter-node traffic is low. We believe that this kind of case is extreme and uncommon, and assume that the case where the token distribution is balanced among experts is the best case which produces the lower bound of the communication time in the expert-centric paradigm.

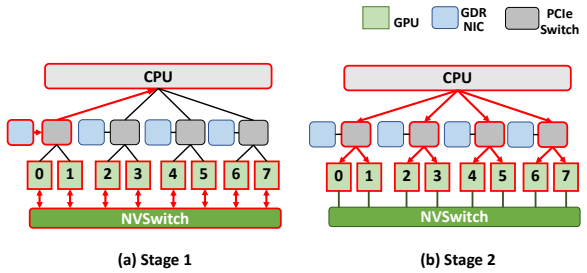


Figure 6: 2-stage scheduling strategy (the activated links and components in the communication are red).

Additionally, the number of tokens can be calculated given the training parameters, including the size of batch, the length of sequence, and the gate parameter *top-K* in the MoE block. The formula is as follows.

$$T = BSk$$

Thus, we have

$$R = \frac{BSk}{4nHE}. \quad (1)$$

Obviously, $R > 1$ indicates that the efficiency of data-centric paradigm is superior to the efficiency of expert-centric paradigm. Theoretically, the communication time in data-centric paradigm is $\frac{1}{R}$ of the time taken in expert-centric paradigm. Since MoE block extends from Transformer block, based on Transformer model[11][36], the expert's dimension H usually lies in range from 256 to 1024. The sequence length S could be 512 or even 2048 for tasks with long sequence [11]. To fully utilize the GPU memory, the batch size B in a worker is usually set as much as possible in a reasonable range, such as from 64 to 512. Obviously, in most cases, $R > 1$ can hold. Examples have been shown in Table 1. In practice, the value of R lies in a wide range since factors like batch and hidden size are task-specific.

Janus is a unified framework of expert-centric paradigm and data-centric paradigm. Janus evaluates R before the training of an MoE model starts. For the MoE blocks where $R \leq 1$, Janus will use the expert-centric paradigm by default. For the MoE blocks where $R > 1$, Janus will use the data-centric paradigm.

5.2 Topology-aware Priority Strategy

In a GPU cluster, the bandwidth of links is heterogeneous. Figure 6(a) shows the internal links in a A100 machine where GPUs are connected by NVlink and its bandwidth is 600GB/s. A GPU is connected to the CPU of the machine by PCIe, whose bandwidth is 64GB/s [3]. A connection across machines goes through GDR NIC, and the bandwidth is 200Gbps. When fetching experts from workers in local or outside machines, ignoring the physical topology easily leads to unnecessary congestion of some links and is harmful to communication efficiency. Therefore, the scheduling priority of the pulling operations needs to be carefully considered based on the topology.

For each worker, the experts to be pulled can be divided into two parts. The first part is internal experts, which are stored in the workers in the same machine. The internal experts are pulled directly via NVlinks. The second part is external experts, which are stored by other machines. The external experts have to be pulled to the CPU memory via an RDMA network and then copied to the GPU memory.

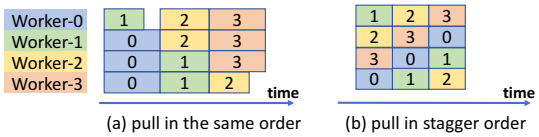


Figure 7: Workers schedule pulling operations in the same order and in the staggered order. Each row represents the scheduling order of a worker. The number $No.$ in the blocks represents fetching experts from worker-$No.$. The length of the block represents the time cost of the corresponding pulling operation.

Algorithm 1 Priority Scheduling for Pulling Internal Experts

Input: Local rank of the worker r; Number of workers per machine m; Number of experts per worker E.
for i $\in [(r+1) * E, m * E)$ **do**
  pulling i-th internal experts;
end for
for i $\in [0, r * E)$ **do**
  pulling i-th internal experts;
end for

Figure 6 illustrates our scheduling strategy, which has two stages. In the first stage, two kinds of operations are conducted. The first kind of operation is that each GPU pulls internal experts from other local GPUs to its GPU memory with the help of its local intra-node scheduler. The second kind of operation is that the machine pulls external experts to its CPU memory with the help of the inter-node scheduler. Figure 6(a) shows this stage. We can see that links involved in two kinds of operations are different, which means these two operations can be done in a parallel manner without any interference. In the second stage, Janus copies the external experts that have been cached from CPU memory to the memory of each local GPU.

Among the above pulling operations, when pulling internal experts from other local GPUs and pulling external experts from Cache Manager, the resource contention can be mitigated via proper arrangements. We explain how to arrange the priority of pulling different experts as follows.

Priority strategy on pulling internal experts from other local GPUs: For the second kind of pulling operations in stage 1 (*i.e.*, pulling internal experts from other local GPUs), if all workers pull internal experts in the same order, the GPU that hosts the requested expert will receive the requests from different workers at the same time, which means the traffic demand is unbalanced in timeline and bottlenecks will occur on each GPU in turn. Figure 7(a) shows this phenomenon. At the beginning, worker-1,2,3 all pull experts from worker-0, which makes the egress of worker-0 become a bottleneck. Therefore, Janus needs to schedule these operations in a staggered order in the first stage.

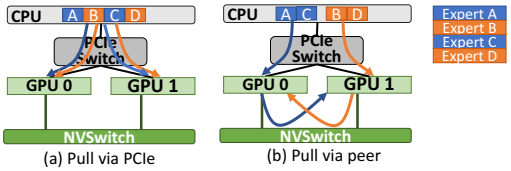


Figure 8: PCIe-Switch-aware Scheduling. (before: pull via PCIe directly. after: pull via peer in best-effort manner)

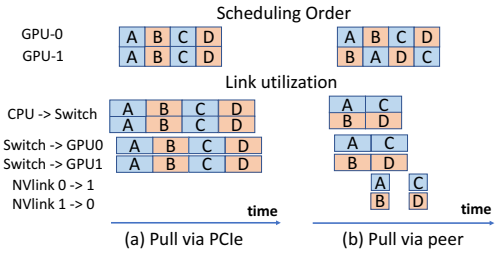


Figure 9: Scheduling order and link utilization in the PCIe-Switch-aware Scheduling.

Suppose each worker has its index, denoted by $r \in [0, m-1]$. We use $rank(i)$ to denote the local rank of the worker in which the i-th internal expert is located. The priority of pulling the i-th expert into the worker r is as follows.

$$P_i^r = \begin{cases} rank(i) - r, & rank(i) > r \\ rank(i) + m - r, & rank(i) < r \end{cases}$$

Note that smaller P_i^r denotes higher priority. The order of pulling experts is shown in Algorithm 1.

Figure 7(b) shows the scheduling results with the above priority assignment. It can be seen that, for each worker, only one other worker pulls experts from it at the same time. The resource competition on the egress bandwidth of the GPU is relieved.

Priority strategy on pulling external experts from Cache Manager: The traffic needs to go through the PCIe links when copying experts from the CPU memory to the memory of GPUs, i.e. from Cache Manager to workers. In an A100 machine, one PCIe switch is connected to two workers (GPUs). If both workers request the same experts from the CPU memory, the expert will go through PCIe switch twice. Thus, the efficiency of the link connected with PCIe switch and CPU memory is sub-optimal. We arrange the order of these pulling operations and leverage the NVlink to improve efficiency.

Figure 8 shows our scheduling method. We first divide the experts on the Cache Manager into two groups. Each one of the two workers connected by a PCIe Switch is responsible for one of the two groups. One worker firstly pulls an expert in the group it is responsible for from the CPU memory via PCIe, and then pulls an expert in the other group from its peer worker via NVlink. In this way, the workload on the link connected with CPU and PCIe Switch is reduced.

Figure 9 illustrates the scheduling results and the utilization of links with our priority strategy. The task order between peers

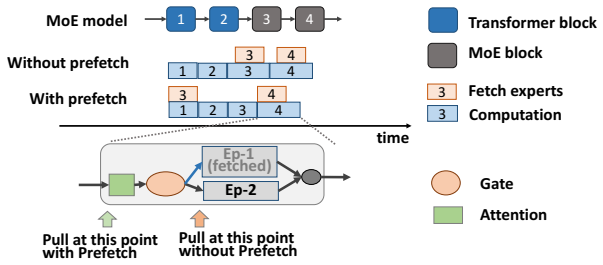


Figure 10: The concept of prefetch.

becomes interleaving, so the idle NVlink can be leveraged, and the tasks can be completed in parallel. With this management on the priority, one worker can cache an expert in time, which will be pulled by its peer via NVlink in the next scheduling interval. Besides, each expert only occupies the GPU memory in two scheduling intervals, which is friendly to the limited GPU memory.

5.3 Provident Prefetch Strategy

An MoE model can have both Transformer blocks and MoE blocks, as Figure 10 shows. When a worker performs computation in the Transformer blocks or the Attention layer in the MoE blocks, it does not need to exchange information with other workers, so its communication links could be idle (at least underutilized) in these time slots. Data-centric paradigm allows Janus to pull experts before the model computation reaches the gate in the MoE block, since experts' weight will not be changed in an iteration. Therefore, at the beginning of an iteration, Janus can start to pull all external experts to the local CPU memory to leverage the idle time slots of cross-node links. If there is a credit for the credit-based buffer, Janus can also prefetch internal experts to the credit-based buffer to leverage the idle time slots of NVlinks. Figure 10 compares prefetch mechanism and non-prefetch mechanism in terms of the timeline of communication and computation of each block (Credit Size is 1 in this example). With prefetch mechanism, the expert computation can start immediately once the computation in the Gate completes, without waiting for fetching experts for a long time. Thus, the training can be sped up.

6 IMPLEMENTATION

Janus not only integrates the communication-efficient Janus Task Queue, but also provides complete implementations of MoE block in the application layer so that developers can easily apply the MoE block in their MoE model. The core implementation of Janus contains 4.3K lines of code. We currently implement Janus as a plugin in PyTorch [28], which is plug-and-play and convenient for developers to use. By *import janus.layer.MoE*, developers define their MoE block, just like using *torch.nn.Linear* to construct their models, and the details of cross-worker communication are transparent to developers.

We introduce the implementation of the basic operations or components which are non-trivial to be implemented as follows.

Implementation of integrating data-centric paradigm into computation graph: How to integrate our data-centric paradigm into the expert parallel system to replace the original All-to-All

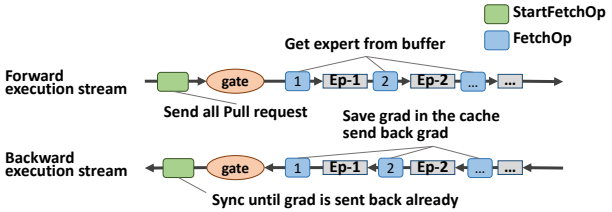


Figure 11: Integrating data-centric paradigm into computation graph

primitive? Janus inherits the class *torch.autograd.Function* and create two types of *Op*: *StartFetchOp*, *FetchOp*, which are inserted into the computation graph of the Expert layer to solve this problem, as Figure 11 shows.

Class *torch.autograd.Function* allows to customize the forward and backward behavior of the *Op*, which can be inserted into the computation graph. *StartFetchOp* is inserted into the graph when the data stream just enters the MoE block. In this *Op*, an optimized fetching order will be calculated and all of the "*pull*" requests will be sent out in the forward phase. At this time, it begins to receive expert modules. Since the PreFetch mechanism is applied, the "*pull*" requests can be sent out before the computation in gate by Janus. *FetchOp* is inserted before the computation of each of expert modules, In this *Op*, Janus will poll the credit-based buffer to acquire the received expert module in the forward phase.

In the backward phase, the order of execution stream is opposite to that in the forward phase. In *FetchOp*, Janus will calculate the gradient of the expert on the buffer and write the result into the cache in Cache Manager and then the gradient of the expert will be sent to the GPU on the corresponding worker. Finally, in *StartFetchOp*, workers synchronize until gradient is already sent back.

Implementation of pull-based communication: Data-centric paradigm is a pull-based communication. We combine the *send* API and *recv* API in the *BytePS* communication library [18][2] to build our *pull* operation.

Specifically, Janus adopts socket to pass messages on the control plane and adopts RDMA connection to pass data on the data plane. When a worker requests an expert from another worker, the requester sends a request to the target worker through the socket, and calls the *recv* API to receive data. The target worker listens to the port of socket all the time. After receiving the request, the target worker calls the *send* API to send data to the requester through the RDMA connection.

7 EVALUATION

7.1 Experimental Setup

We evaluate Janus on a cluster with 32 GPUs on 4 machines. Each machine is equipped with 8× NVIDIA A100 SXM 80GB GPUs and 200Gbps NIC. Each machine also has 500GB of memory and 40 CPU cores. GPUs are connected by NVlink and NVSwitch within one machine, as Figure 6 shows.

We evaluate Janus on 3 models: MoE-BERT, MoE-GPT and MoE-Transformer-xl [10]. BERT can be regarded as a representative of

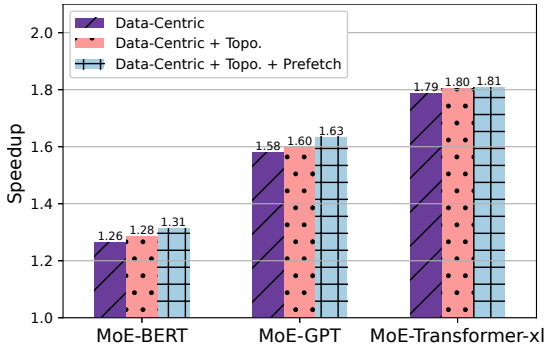


Figure 12: Speedup of the optimizations separately

the category of pre-trained models based on transformer encoder. GPT and Transformer-xl can be regarded as representatives of the category of models based on transformer decoder. The configuration of models is shown in Table 1. All of these models have 12 blocks, each of which is a normal Transformer block or MoE block. The 2nd, 5th, 8th, and 11th blocks in the MoE-BERT model are MoE blocks. The 11th block in the MoE-GPT model is an MoE block. All of the 12 blocks in the MoE-Transformer-xl model are MoE blocks. Each MoE block has 32 experts. Thus, the number of experts for each block pre-allocated to each worker (i.e. E) is 1.

We compare Janus with a state-of-art MoE training system named Tutel [4] in the evaluation experiments. Tutel supports expert parallelism with expert-centric paradigm and has many communication optimizations in All-to-All primitives. As a unified implementation of expert parallelism, Janus supports expert-centric paradigm and data-centric paradigm. Janus uses paradigm based on the theoretical gain R. When $R \leq 1$, the expert-centric paradigm in Janus has less difference from the expert-centric paradigm in Tutel, so we mainly focus on the cases in which all MoE blocks or at least 1 MoE block in models satisfied $R > 1$.

7.2 Ablation Study

7.2.1 Ablation of Optimization. We evaluate the contribution of each strategy in this subsection and observe how the speedup on the iteration time gradually improves when the three scheduling strategies are gradually applied to the system. Note that the fine-grained task scheduling is the fundamental design in the data-centric paradigm in Janus, which is necessary to make the system work. Topology-aware priority strategy and prefetch strategy are optional choices that are designed to further improve performance. So we label the system with only fine-grained task scheduling as *Data-Centric* in the figure. In this subsection, the baseline of the speedup is the expert-centric paradigm in Janus.

We visualize the results in Figure 12. The result shows that the data-centric paradigm is the main contributor to efficiency improvement. The Speedup on training MoE-BERT, MoE-GPT, MoE-Transformer-xl in the data-centric paradigm without Topology-aware optimization and prefetch strategy, can already reach 1.26×, 1.58× and 1.79×. This is because the traffic volume has been greatly reduced in this paradigm. The topology-aware scheduling strategy and the prefetch strategy both make an incremental and effective

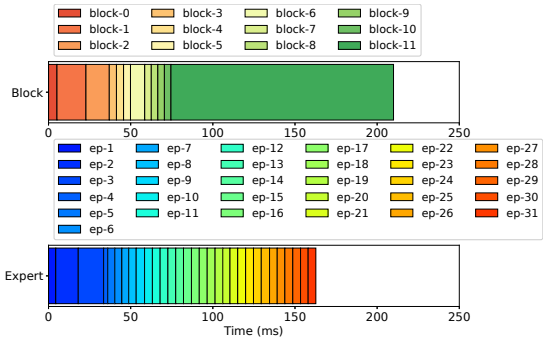


Figure 13: Time breakdown of the computation of each block and pulling each expert on MoE-GPT

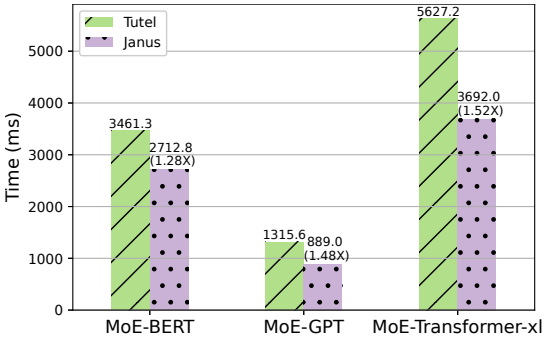


Figure 14: End-to-end performance of Janus and Tutel

contribution. With all of the optimizations, the Speedup on training MoE-BERT, MoE-GPT, MoE-Transformer-xl can reach 1.31×, 1.63× and 1.81×.

7.2.2 Computation-Communication Overlap of Prefetch Strategy. To show the details of how the prefetch strategy improves efficiency in practice (*i.e.*, the computation- communication overlap achieved by the prefetch strategy), we take the MoE-GPT model as an example and observe its forward phase in an iteration when the prefetch mechanism is applied and Topology-aware optimization has not been applied.

We visualize the result in Figure 13. The sub-figure above shows the timestamps when each of 12 blocks completes the computation. Each color represents a block. The sub-figure below shows the timestamps when each of experts has been pulled completely. Each color represents an expert. The result shows that the MoE blocks (i.e., the 11th-block) indeed takes a longer time than normal transformer block. This is because the former blocks are normal transformer blocks in this model and there is no need to exchange data with other workers for these blocks. When the model completes the computation of the first 11 blocks, the worker has already pulled 12 experts. The computation-communication overlap is around 74.9 ms, which is the time cost saved by the prefetch strategy. The accelerated forward phase in an iteration takes 210.4 ms. In this way, the forward phase can be sped up by 1.36× and the prefetch strategy is effective.

We can also notice that pulling the 2^{nd} and 3^{th} experts (i.e. $ep-2$ and $ep-3$) takes a little bit longer time than pulling other experts. This is because the topology-aware priority strategy has not been applied here. This worker and other workers request the 2^{nd} and 3^{th} experts from worker-2 and worker-3 at the same time, and thus the egress of these 2 workers has congestion. This experimental result further confirms the phenomenon we describe in Figure 7 and topology-aware scheduling is necessary.

7.3 End-to-End Performance

The end-to-end performance, which is measured by the time cost of an iteration, can directly represent the efficiency of the system. We compare the end-to-end performance of Janus with Tutel in this subsection.

The time cost of an iteration not only includes the time taken by communication, but also includes the time cost of forward computation and backward computation of the model. Janus mainly optimizes the communication part in an iteration. In this model configuration, based on formula 1, the theoretical gain in communication of MoE-BERT, MoE-GPT and MoE-Transformer-xl are $R = 5.33$, $R = 5.33$ and $R = 16$, respectively.

Figure 14 shows the end-to-end performance of Janus and Tutel. In terms of end-to-end performance, Janus can greatly reduce the time cost in an iteration and achieve 1.28×, 1.48×, 1.52× speedup on these three models, respectively. This result suggests that Janus is highly efficient and outperforms Tutel when $R \geq 5.33$.

7.4 Sensitivity analysis

To explore how the parameters in configuration affect the training efficiency, we explore the sensitivity of batch size, length of sequence in this subsection.

To analyze the sensitivity of batch size, we fix $S = 256, k = 4$ for MoE-BERT, $S = 128, k = 8$ for GPT and $S = 256, k = 2$ for transformer-xl, and then observe the end-to-end performance when batch sizes are 64 and 128. Figure 15 shows the experimental result. Please note that this configuration is different from the configuration in Figure 14 (*i.e.* Table 1). The figure shows that the iteration time in both systems increases with the increase of the batch size. This is reasonable because the computation workload also increases with the increase of the batch size. However, Tutel (*i.e.* expert-centric paradigm) is more sensitive than Janus (*i.e.* data-centric paradigm in this experiment). With the increase of the batch size, the iteration time in Tutel can greatly increase and the speedup of Janus on Tutel also increases. This is because the traffic volume in All-to-All communication also increases besides the computation workload.

To analyze the sensitivity of length of sequence, we fix $B = 256, k = 4$ for MoE-BERT, $B = 32, k = 8$ for GPT and $B = 64, k = 2$ for transformer-xl, and then observe the end-to-end performance when the lengths of sequence are 256 and 512. Figure 16 shows the sensitivity on sequence length. From the figure we can see that the iteration time in both systems increases with the increase of the sequence length since the computation workload also increases. Tutel is more sensitive than Janus and the reason is the same as the analysis on the sensitivity on batch size.

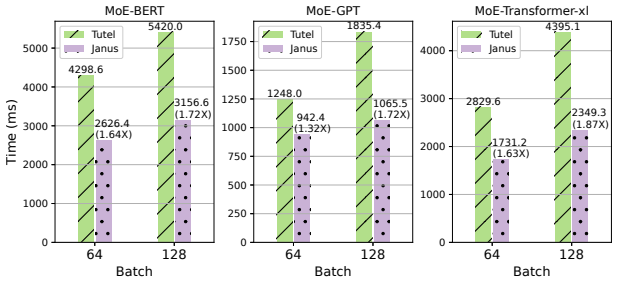


Figure 15: End-to-end performance for different batch sizes

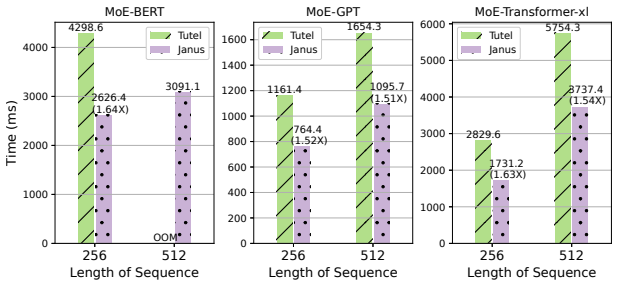


Figure 16: End-to-end performance for different lengths of sequence (OOM means out of GPU memory)

Besides, there is an error (*i.e.* out of GPU memory) when training MoE-BERT model in Tutel when $S = 512$. This is because there are so many tokens to be exchanged in the All-to-All communication and the GPU has no more available memory as buffer to receive these tokens. While Janus does not have this problem since only experts instead of tokens are transferred in the data-centric paradigm and the volume of experts is smaller than the volume of tokens.

7.5 Unity of data-centric paradigm and expert-centric paradigm in Janus

Janus is a unified training framework of expert-centric paradigm and data-centric paradigm. It can support both paradigms at the same time when training a model. In this subsection, we take Pyramid-Residual MoE (*i.e.* PR-MoE) [31] as an example to show how Janus unifies both paradigms.

PR-MoE model is a category of MoE models in which the shallow MoE blocks has a small number of experts and the deep MoE blocks have a large number of experts. For example, in PR-MoE-transformer-xl model, we make each of the first two MoE blocks has 16 experts and each of the last two MoE blocks have 64 experts. Thus, with 16-GPU expert parallelism, each GPU holds 1 expert for each of the first two blocks (*i.e.* $E = 1$) and holds 4 experts for each of the last two blocks (*i.e.* $E = 4$). We set $B = 32, S = 256, k = 2$ when training the PR-MoE-transformer-xl model, and the theoretical gain is $R = 4$ for the first two blocks and $R = 1$ for the last two blocks. In practice, we notice that the bandwidth utilization has a gap to 200 Gbps (the capability in NIC) when copying remote experts to local CPU cache because of the limit of the PCIe link between PCIe switch and CPU. Conservatively, we consider

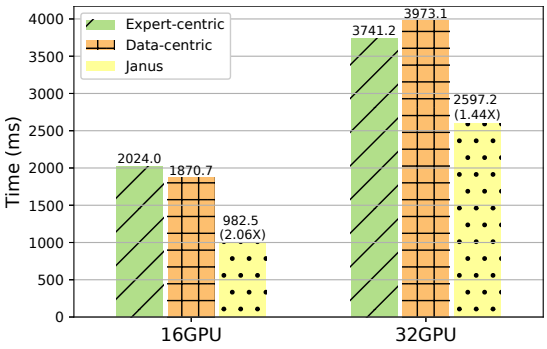


Figure 17: End-to-end performance on the PR-MoE-Transformer-xl for different paradigm

expert-centric paradigm would be better than current implementation of data-centric paradigm when $R = 1$. Thus, for training the PR-MoE-transformer-xl model, only expert-centric paradigm or only data-centric paradigm is sub-optimal. Janus system unifies both paradigms. It automatically uses data-centric paradigm for the first two MoE blocks and expert-centric paradigm for the last two MoE blocks to achieve better performance.

To observe the scalability of Janus and the effect of the number of GPUs, we also conduct a PR-MoE-transformer-xl experiment on a 32-GPU cluster. In this case, we make the batch size $B = 64$ and each of the first two MoE blocks has 32 experts while each of the last two MoE blocks has 128 experts. Thus, with 32-GPU expert parallelism, each GPU still holds 1 expert for each of the first two blocks (*i.e.* $E = 1$) and holds 4 experts for each of the last two blocks (*i.e.* $E = 4$).

Figure 17 shows the experimental result. From the figure, we can see that Janus outperforms the pure expert-centric paradigm and the pure data-centric paradigm. Compared with the expert-centric paradigm, it has 2.06× and 1.44× speedup on the 16-GPU cluster and 32-GPU cluster, respectively. With the increase of the number of GPU (*i.e.* the number of machine), the iteration time of expert-centric, data-centric paradigm and unified paradigm all increases. The speedup reduces as expected as we describe in Equation 1.

Please note that the number of GPUs (*i.e.* 16 and 32) in this experiment is not a small parameter. For simplicity, this paper is based on the context that only data parallelism and expert parallelism are used in training. And 16 or 32 is a parameter for the degree of data parallelism (also for the degree of expert parallelism). When training a giant model, like 175B GPT-3 model, on 1024 GPUs, all tensor parallelism (TP), pipeline parallelism (PP) and data parallelism (DP) have to be applied and a typical setting for degree of parallelism is TP=8, PP=8 and DP=16.

8 RELATED WORK

Mixture-of-Expert model: Giant models with large scales of parameters have shown surprising model quality in various areas. For example, BERT [11] achieves high performance on language comprehension and GPT [30] achieves high performance on language generation.

To further increase the size of models and improve the performance of models, MoE has been widely applied [21][13] [29][39][25].

Switch Transformer [15] proposed by Google is the early investigation of MoE. GShard [19], proposed later by Google, increases the size of MoE to a trillion scale. PaLM [8] and GaLM [14] proposed by Google achieve excellent results on language tasks, while M6-T [37] proposed by Alibaba achieves good results on multi-modal tasks. Users can train these models using our system, and improving the training efficiency of these MoE models is the goal of our system.

MoE training system: Expert Parallelism is proposed by Google to train Switch Transformer [15], which could have 1.6 trillion parameters. DeepSpeed-MoE [31] and PathWay [6] utilize a variety of parallelism, including expert parallelism, to train MoE models. BASE layers [20], which is a part of FairSeq [26], is another implementation of expert parallelism of MoE training systems. Some research works have been proposed to improve the training efficiency of MoE models. Tutel [17] proposed by Microsoft designs adaptive parallelism switching and adaptive pipelining to handle dynamic workloads of MoE. Tutel also proposed hierarchical All-to-All communication to improve communication efficiency. FasterMoE [16] proposed the concept of shadow expert to deal with the imbalanced workload and proposed a smart fine-grained scheduler to achieve asynchronous All-to-All communication. It achieves better performance with the help of Megatron-LM [23]. SE-MOE [34], proposed by Baidu, focuses on solving the limitation of storage and improving access efficiency. SE-MOE also proposed hierarchical All-to-All communication to improve communication efficiency. Alpa [38] proposed a training strategy for MoE models from the perspective of automated parallelism. All of these previous works are based on the expert-centric paradigm naively. In this paper, we propose the data-centric paradigm, a novel and equivalent way to implement the necessary communication for training MoE models. Janus unifies expert-centric paradigm and data-centric paradigm, and it is communication-optimal both in theory and in practice.

DeepSpeed Zero3 [32] also proposes collecting parameters from other GPUs on-the-fly to support very large models. One may think the basic design idea is similar to Janus, but the core of Janus targets on exploring the optimal communication pattern in MoE training, which is essentially different from Zero3. Besides, at implementation level, Janus incorporates MoE-specific optimizations such as asynchronous communication, hierarchical communication, and the prefetch mechanism to further improve training efficiency.

9 DISCUSSION

In this section, we briefly discuss the application scope and the future work of Janus.

As analyzed in Section 5.1.3, the R metric is a critical value for data-centric's performance gain, which scales linearly with the input size (*i.e.*, number of tokens) but decreases as the model size becomes larger. It is well known that Transformer-based Large Language Models (LLMs) are becoming increasingly huge in terms of model size. However, recent NLP literature is also exploring longer context length for LLMs (e.g., 100K in Anthropic Claude [1] and even 1B in LONGNET [12]) so that LLMs learn the ability to understand and interact with human even after long conversation. Given this fact, training LLMs usually requires long sequence length. We use a concrete example to elaborate this impact. The sequence length in training GPT-3 (hidden size 12288) is usually 2048, and the global batch size can reach more than 1M [7]. Supposed the degree of data parallelism is 128, k as 1 in MoE gate, and each worker holds only an expert ($E = 1$), we get the $R = 20.35$. Therefore, data-centric paradigm not only can accelerate current GPT-3 style MoE model, but can also expect higher gain in the near future as long context length is becoming increasingly important.

Limited by the GPU memory, training a large MoE model often requires tensor parallelism [35] that partitions the large matrix multiplication in Transformer into smaller chunks and distributes them to several GPUs for parallel execution. For simplicity, this paper is based on the context that data parallelism and expert parallelism are used in training. However, Janus also supports tensor parallelism and the implementation is flexible to adapt to popular frameworks such as Megatron-LM[35].

Although Janus is primarily designed for training, the same design principles can be applied to inference as well, since the communication pattern is similar. Thus, Janus can improve the overall efficiency of LLMs' training and inference pipelines, including pre-training, supervised fintuning [9], RLHF (Reinforcement Learning with Human Feedback) [27] and online serving. Overall, our future work is to provide efficient MoE framework support for the entire LLM production and deployment processes through Janus.

10 CONCLUSION

MoE has been shown to achieve remarkable performance across a variety of applications. However, training a large-scale MoE model is non-trivial. To train large-scale MoE models efficiently, we propose the concept of data-centric paradigm and design Janus.

Data-centric paradigm creatively argues that data can be in-place and model can be moved in expert parallelism, while people always consider model is in-place and data is moved in expert parallelism. Data-centric paradigm provides a new dimension in parallelism.

Janus unifies expert-centric paradigm and data-centric paradigm and it is communication-optimal. The data-centric paradigm can reduce traffic volume and make communication workload balanced under certain conditions. To improve the system efficiency, we carefully design the scheduling strategies for the requests of fetching experts, including asynchronous communication, cache mechanism in the hierarchical communication, topology-ware priority management and prefetching mechanism. With asynchronous communication, expert computation can be overlapped with the operations of fetching experts. With the cache mechanism, we reduce the inter-node communication traffic. By carefully arranging the priority of requests, we alleviate the contention on the bandwidth of intra-node links. We achieve further overlap between communication and computation with prefetch mechanism. All of these strategies are helpful to reduce training time. Our experiment has proven the effectiveness of our system.

This work does not raise any ethical issues.

Acknowledgments. We sincerely thank our shepherd Amar Phanishayee and anonymous reviewers for their valuable feedback on this paper. This work is sponsored by National Natural Science Foundation of China (No.62072269). Jessie Hui Wang is the corresponding author. Juncai Liu is with Institute of Network Sciences and Cyberspace (Tsinghua University), and also with Beijing National Research Center for Information Science and Technology.

REFERENCES

[1] Anthropic claude. https://www.anthropic.com/index/introducing-claude.

[2] BytePS. https://github.com/bytedance/byteps.

[3] NVIDIA A100. https://www.nvidia.com/en-us/data-center/a100/.

[4] Tutel. https://github.com/microsoft/tutel/.

[5] Mikel Artetxe, Shruti Bhosale, Naman Goyal, Todor Mihaylov, Myle Ott, Sam Shleifer, Xi Victoria Lin, Jingfei Du, Srinivasan Iyer, Ramakanth Pasunuru, et al. Efficient large scale language modeling with mixtures of experts. *arXiv preprint arXiv:2112.10684*, 2021.

[6] Paul Barham, Aakanksha Chowdhery, Jeff Dean, Sanjay Ghemawat, Steven Hand, Daniel Hurt, Michael Isard, Hyeontaek Lim, Ruoming Pang, Sudip Roy, et al. Pathways: Asynchronous distributed dataflow for ml. *Proceedings of Machine Learning and Systems*, 4:430–449, 2022.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[8] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[9] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

[10] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, and Furu Wei. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*, 2023.

[13] Ke Ding, Xin Dong, Yong He, Lei Cheng, Chilin Fu, Zhaoxin Huan, Hai Li, Tan Yan, Liang Zhang, Xiaolu Zhang, et al. Mssm: a multiple-level sparse sharing model for efficient multi-task learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2237–2241, 2021.

[14] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR, 2022.

[15] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

[16] Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 120–134, 2022.

[17] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, et al. Tutel: Adaptive mixture-of-experts at scale. *arXiv preprint arXiv:2206.03382*, 2022.

[18] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed dnn training in heterogeneous gpu/cpu clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 463–479, 2020.

[19] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[20] Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, pages 6265–6274. PMLR, 2021.

[21] Dingcheng Li, Xu Li, Jun Wang, and Ping Li. Video recommendation with multi-gate mixture of experts soft actor critic. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1553–1556, 2020.

[22] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[23] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[24] Xiaonan Nie, Shijie Cao, Xupeng Miao, Lingxiao Ma, Jilong Xue, Youshan Miao, Zichao Yang, Zhi Yang, and Bin Cui. Dense-to-sparse gate for mixture-of-experts. *arXiv preprint arXiv:2112.14397*, 2021.

[25] Xiaonan Nie, Pinxue Zhao, Xupeng Miao, and Bin Cui. Hetumoe: An efficient trillion-scale mixture-of-expert distributed training system. *arXiv preprint arXiv:2203.14685*, 2022.

[26] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

[27] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[29] Zhen Qin, Yicheng Cheng, Zhe Zhao, Zhe Chen, Donald Metzler, and Jingzheng Qin. Multitask mixture of sequential experts for user activity streams. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3083–3091, 2020.

[30] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[31] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International Conference on Machine Learning*, pages 18332–18346. PMLR, 2022.

[32] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[33] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[34] Liang Shen, Zhihua Wu, WeiBao Gong, Hongxiang Hao, Yangfan Bai, HuaChao Wu, Xinxuan Wu, Haoyi Xiong, Dianhai Yu, and Yanjun Ma. Se-moe: A scalable and efficient mixture-of-experts distributed training and inference system. *arXiv preprint arXiv:2205.10034*, 2022.

[35] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[37] An Yang, Junyang Lin, Rui Men, Chang Zhou, Le Jiang, Xianyan Jia, Ang Wang, Jie Zhang, Jiamang Wang, Yong Li, et al. M6-t: Exploring sparse expert models and beyond. *arXiv preprint arXiv:2105.15082*, 2021.

[38] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Joseph E Gonzalez, et al. Alpa: Automating inter-and intra-operator parallelism for distributed deep learning. *arXiv preprint arXiv:2201.12023*, 2022.

[39] Simiao Zuo, Xiaodong Liu, Jian Jiao, Young Jin Kim, Hany Hassan, Ruofei Zhang, Tuo Zhao, and Jianfeng Gao. Taming sparsely activated transformer with stochastic experts. *arXiv preprint arXiv:2110.04260*, 2021.