AICN Report Draft v0.2

Introduction

This paper is the result of a study item within the IEEE 802 “Network Enhancements for the Next Decade” Industry Connections Activity known as Nendica.

Scope

Study main factors in AI system which impact traffic.
Analyze the major challenges for AI computing network.
Investigate future technologies.
Identify potential standard work.

Purpose

Understand the requirement of AI computing network.
Look for potential standardization opportunity in IEEE802.

Stepping into the AI era

ChatGPT ignites enthusiasm for AI large models

- ✓ OpenAI announced ChatGPT in November 2022. It gained over 100 million users within 2 months which is the fastest-growing consumer software application in history.
- ✓ What is ChatGPT?
  ChatGPT is short for Chat Generative Pre-trained Transformer. “Chat” is its function. “Generative” represents it uses generative AI technology. ChatGPT has significantly improved in its ability to make conversations, generate high-quality content, and understand language. It is based on GPT-3.5, and further tune the model based on human needs.
- ✓ ChatGPT is a milestone of generative AI. (RNN->Transformer->ChatGPT)
  In the past few years, generative AI has been developing slowly due to the disadvantage of RNN(recurrent neural network). It was not until the "Transformer" architecture emerged in 2017 and solved the problems of the traditional RNN model that generative AI began to develop fast.
  With the geometric growth of model parameters and the exploration of training methods, the emergence of ChatGPT marks that the generic large model breaks through the traditional development dominated by small scale models in the field of NLP.

AI large models show emergent abilities

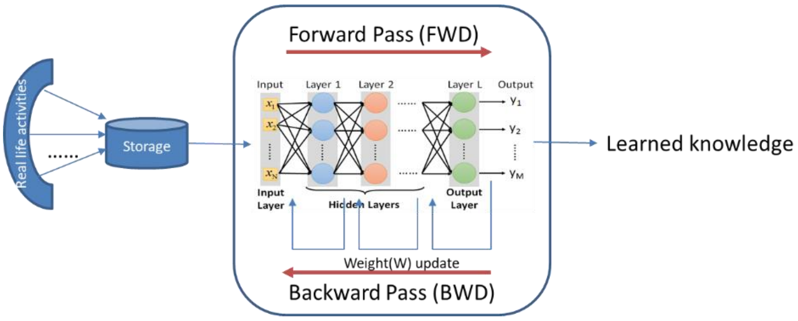
The so-called ‘emergent abilities’ in the field of AI large models refer to when a model breaks through a certain scale, its performance significantly improves, showing amazing and unexpected abilities. Generally speaking, models in the range of tens of billion to hundreds of billion parameters may experience ability emergence.
As Google and Stanford said, emergent abilities that are not present in smaller-scale models

but are present in large-scale models, which are qualitative changes resulted by quantitative changes. Few-shot prompted tasks by five language model families are analyzed in their research. The result is that the model achieves random performance until a certain scale, after which performance significantly increases to well-above random. [7]

AI Large models require distributed system

AI working process

The technology behind AI large models is deep learning which uses DNN-based(Deep Neural Network) architecture. A neural network consists of input layer, hidden layers and output layer. Neurons are linked together in the network. The training includes several steps, as shown in below figure. First, feeding the data into neural networks. Then, in forward propagation, calculations with parameters are performed in each neuron, from the input layer to the output layer through the neural network. After completing the calculations in the output layer, loss function is used to calculate how far the output value is from the real value. The deviation which can be understood as gradient then is used as a feedback signal to update parameters in the backward propagation. This is one iteration. The training is run iteration by iteration until the parameters that produce satisfied output are calculated.



AI large models have to be trained on vast amounts of data. Transformer architecture[9] is the most popular architecture today for AI large model training. It is a set of neural networks consisting of an encoder and a decoder with self-attention capabilities. Analysis in this paper is mainly based on transformer-based architecture.

Distributed AI system

✓ Why needs distributed system?

Training large-scale AI models is indeed a complex task that requires significant computational resources. A single AI accelerator struggle to handle the immense workload of training such models. This is because large models, like those with billions of parameters, require a substantial amount of memory and compute power that surpasses the capabilities of a single AI accelerator. Take GPT-3 of 175B parameters as example. It requires about 2.8TB memory to store the model and 430ZFLOPS to train it. A single Nvidia A100 [10] product which was release in the same period as GPT-3 is 80GB. So 35 pieces of A100 are needed to store the model. Considering from compute power perspective, A100 delivers 312 TFLOPS of performance. As a result, it needs 43.8 years to train the model with one A100 or one day with 16000 pieces of A100.

To address this, distributed systems and parallel techniques have been used to support the training process.

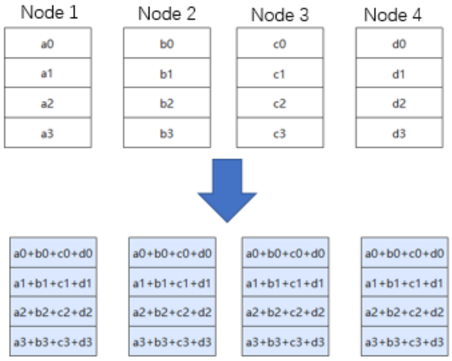
✓ Collective communication in distributed AI system

Collective communication is a cornerstone in distributed AI system. It is used to manage and coordinate the exchange of information across the various AI accelerators involved in a computational task. This often involves the synchronization of gradient updates during the training across multiple AI accelerators. The efficiency of these collective communications is paramount as they can significantly impact the speed and scalability of AI model training. [11] shows the training time breakdown of several large models on TPUv4 which is Microsoft AI accelerator product. Although those models have different types and different amounts of data communication depending on the model architecture and how partitioning is performed, they all spend a substantial percentage of the training time on data communication, that is between 20% to 50%. When model size grow bigger, proportion of communication time in training will increase if there is no further optimization.

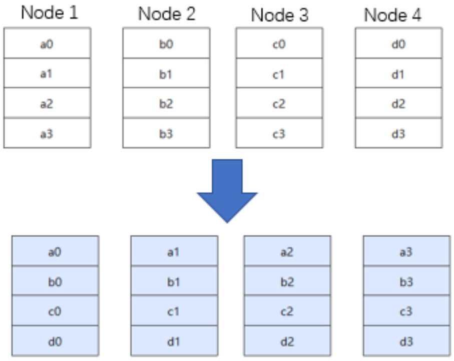
Fundamental collective communication operations used in AI training include Allreduce and Alltoall. This will be shown in subsequent section “Parallel processing in distributed AI system”.

- Allreduce:

  Data from all members of a group is aggregated using a specified operation, such as sum, maximum, or minimum, and then the result is distributed back to all members of the group.

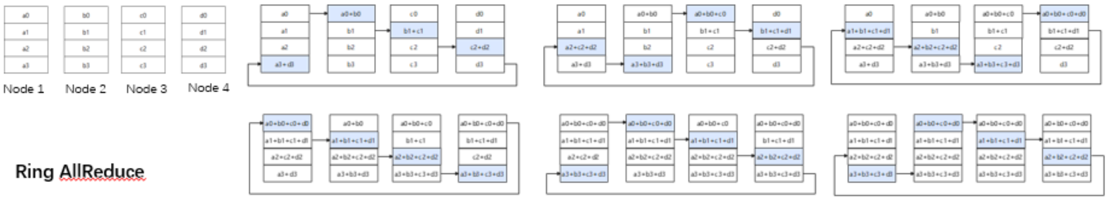


- Alltoall



Implementations of collective communication operations can vary based on the underlying hardware and network topology, but the goal remains the same. That is to reduce communication overhead and latency while ensuring accurate and timely data exchanging

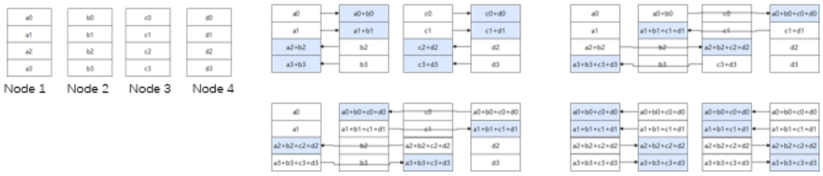
Ring Allreduce and half doubling Allreduce are 2 implementations for Allreduce operation. For ring allreduce, the 4 nodes form a ring. In each round, every node sends data to the next node, and receives data from previous node. After 6 rounds, allreduce is done. This implementation is simple, but may have latency issue when there are a lot of nodes, which is unfriendly to small messages.



Ring AllReduce

Half doubling reduces the communication times, meaning less rounds thus less time to finish allreduce. But the nodes have to change its connect frequently. In round 1, node 1 talks with node 2. In round 2, node talks with node 3. This changing introduces some cost.

Half Doubling AllReduce



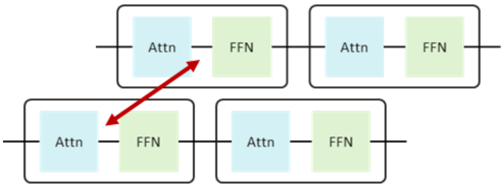
Different implementations have their own pros and cons. In practice, it is difficult to use one implementation to satisfy all cases. The decision needs to be made based on comprehensive considerations such as the physical network topology, communication message size.

✓ Parallel processing in distributed AI system

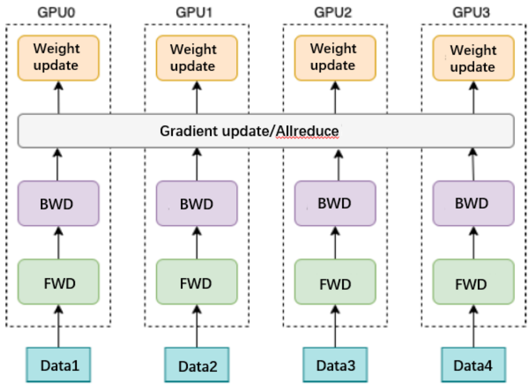
Parallel processing techniques are essential components of distributed systems, utilized to enhance performance and scalability. It mainly includes data parallelism, tensor parallelism, pipeline parallelism, expert parallelism and sequence parallelism. They correspond to orthogonal partitioning along the dimensions of batch size and hidden size, and introduce collective communications in the system.

DP:

The training data is divided into multiple batches. Batches are independent from each other. Each batch is processed on a separate accelerator simultaneously. In this approach, each accelerator receives a portion of the data and computes the gradients on its own before they are combined to update the model parameters.

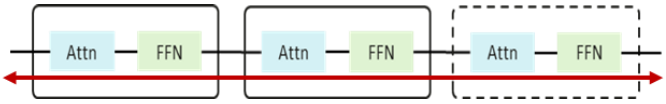


In the forward computation stage, each computing device uses its own data to calculate the loss value. Since the data read by each computing device is different, the loss value obtained on each computing device is often different. In backward computation stage, each computing device calculates the gradient based on the loss value calculated forward, and uses the AllReduce operation to calculate the average of the accumulated gradient, thereby ensuring that the gradient value used to update parameters on each computing device is the same. In the parameter update phase, the parameters are updated using the average gradient.



PP:

A model is divided into multiple stages and each stage is executed on a separate accelerator. The output of each stage is passed on to the next stage by using **send/recv operation**.

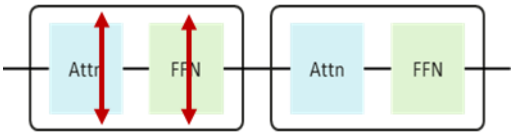


Original PP only solves the memory problem of large model, but cannot accelerate training, because it introduces point to point communication between accelerators, and during the communication, accelerators are idle.

There are many solutions to optimize pipeline parallelism, such as Gpipe [8]. It splits minibatch into microbatches, in order to reduce ‘bubble’ in the pipeline. Compared with original PP, Gpipe increase accelerator utilization from 1/N to M/(N+M-1), where N is the number of accelerator, M is the number of microbatch.

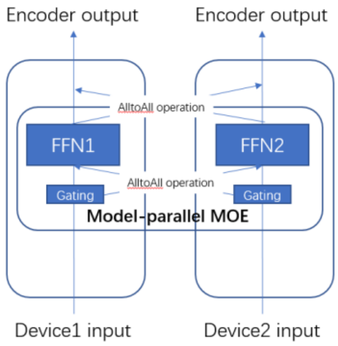
TP:

Tensor parallelism focuses on intra-layer parallelization within the model. For example, in Transformer architecture, it uses matrix-matrix multiplication operation. The weight matrix is split along one of its dimensions (usually row or column) and then assigned to multiple accelerators for individual computation. Each accelerator is responsible for only a portion of the entire matrix, and all accelerators synchronize through AllReduce operation to merge the result.



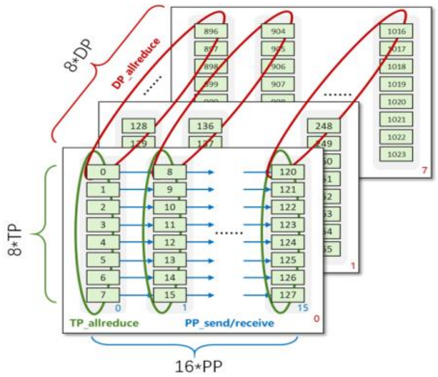
EP

The principle of EP is to divide complex tasks into multiple sub-tasks and assign them to specific "expert" for processing. It becomes popular in large model due to its scalability and flexibility. In MOE(Mixture of Expert) architecture, it integrates multiple experts and a gate. The gate is responsible for selecting the most suitable expert to deal with specific tasks based on the input data features. Each expert may need to process a portion of all input data, and their outputs need to be aggregated. This introduces Alltoall operations between devices according to Gshard[13]. Alltoall operation consumes 31.18% time of MOE layer in DeepSpeed-MOE system[12].



Hybrid parallelism

In practice, a combination of parallelism techniques is often employed to harness the full potential of distributed computing systems, maximizing resource utilization and expediting training processes. Below figure shows an example of DP, PP, TP combination. In this scenario, the AI cluster comprising 1024 accelerators employs these parallelism techniques. The model's layers are partitioned across 8 accelerators via tensor parallelism. The entire model undergoes division into 16 pipeline stages based on the sequential order of layers. Concurrently, the dataset is segmented into 8 batches.



It's worth noting that the specific partitioning strategies employed may vary, depending on various factors such as infrastructure capabilities and model architecture. These decisions are implementation-dependent, tailored to optimize system performance and efficiency.

Characteristics of communication in AI training processes

In distributed AI systems, AI training processes often exhibit unique traffic patterns, characterized by uneven and sporadic data flows across spatial and temporal dimensions. These patterns reflect the inherent complexity of AI algorithms, which require extensive data processing capabilities. Despite the complexity, this traffic tends to follow a consistent trajectory within the network. With prior knowledge and better understanding of how AI tasks are distributed, these patterns become increasingly predictable.

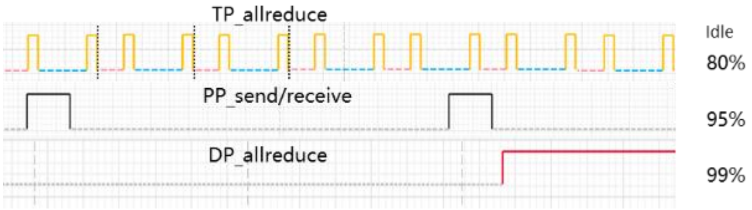
Sparsity of traffic in space

The communication relationship between AI accelerators is not fully meshed, but rather determined by the specific requirements of the AI job. Once AI job begins, the communication relationship becomes fixed throughout the task's execution. This relationship is primarily influenced by two key factors: the architecture of the model being trained and the partitioning strategy employed. Each AI accelerator only communicates with a limited number of peer points, calculated based on the model's partitioning. In a 5-dimension hybrid parallelism model partitioning scenario, this count reaches its maximum as (number of TP - 1) + (number of SP – 1) + (number of DP – 1) + 1 for each accelerator.

When training dense models such as GPT-3 with a trillion parameters using thousands of accelerators, only a small fraction—ranging from 0.57% to 1.5%—of AI accelerator pairs encounter communication traffic. While in the case of sparse models like GPT-4 with a quadrillion parameters trained across tens of thousands accelerators, the proportion of communication pairs experiencing traffic diminishes even further, ranging from just 0.024% to 0.86%.

Sparsity of traffic in time

Distributed AI systems leverage parallel strategies to accelerate AI tasks, with the advancement of these tasks hinging upon effective communication between AI accelerators. Various forms of parallelism employ distinct communication methods: tensor parallelism and data parallelism utilize Allreduce communication, pipeline parallelism involves point-to-point send/receive traffic, and sequence parallelism utilizes Allgather communication. Each form of parallelism delineates a logical plane within the network. The communication between AI accelerators shows sporadic behavior, resembling a square waveform, within each logical plane. Notably, TP Allreduce occurs more frequently than PP send/receive, with PP send/receive being more common than those of DP, as illustrated in the figure. Nonetheless, despite these fluctuations, the network links remain mostly idle for extended durations. This underscores the intermittent nature of communication demands in distributed AI systems.



Huge amount of traffic for communication

Despite the sparse pattern of communication observed in the AI training process, the overall traffic amount remains substantial. Taking GPT-3 175B model as an example, the amount of traffic generated by various types of communication in each iteration is shown in the following table. While the absolute amount of communication traffic may differ based on factors such as model architecture and partitioning strategy, it invariably increases with the scale of the AI model. As AI models grow larger in parameter size and dataset, the demands on network bandwidth for communication likewise increase.

	Typical operation	Traffic amount
Tensor Parallelism (TP)	Allreduce	100s GB
Pipeline Parallelism (PP)	Send/receive	100s MB~10s GB
Data Parallelism (DP)	Allreduce	GB
Expert Parallelism (EP)	Alltoall	10s GB

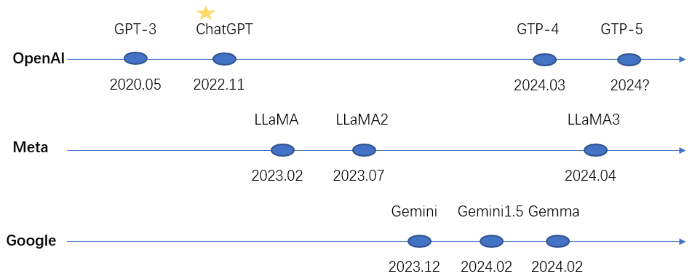
Requirements and Challenges of AI computing Networks

Total compute of distributed AI system = single GPU compute * Scale * Efficiency * Availability

- Scale: number of AI accelerators
- Efficiency: percentage of theoretical peak FLOPS
- Availability: percentage of system working time

Scale

In the last 4 years, industry giants and startups have accelerated AI large models development, such as GPT from OpenAI, LLaMA from Meta and Gemini/Gemma from Google.

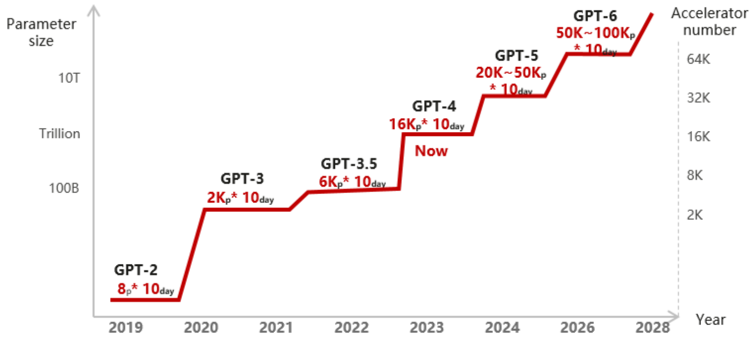


The evolvement of models follows the LLM scaling law, increasing the size of model to get better and better performance. Take GPT series as example. GPT3 is 175B parameters with 300B tokens. When it comes to GPT4 2 years later, parameter size grows to 1.8T with 13T tokens. GPT5 is not released yet, but It is stated to be much more powerful than GPT4 by OpenAI. The parameter size is estimated to be close to 10T with 30T tokens.

The larger the model is, the more computing power is required to train the model. Roughly, F=6PD. F is computing power, P is parameter size, and D is token number. [1] So, compared with the model 4 years ago, the computing power needs increases 10000 times.

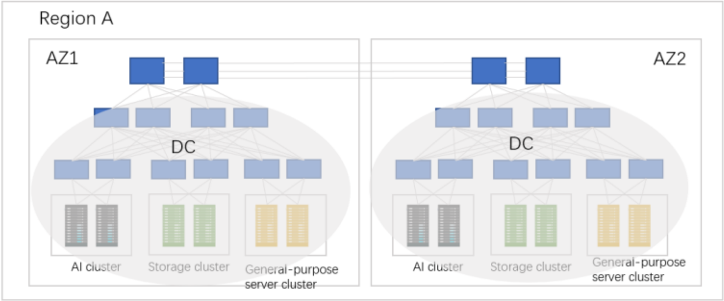
Although the accelerators used for AI training are also constantly evolving, like Nvidia's latest GPU B200 which reaches 2.5PFLOPS[5], almost 8 times that of its previous product A100, the speed of evolvement is lag behind the growth of required computing power. So it is seen that the scale of AI cluster increases from thousands of accelerators to tens of thousands accelerators, even to hundreds of thousands accelerators[2].

Below figure depicts the development of model size, cluster scale, and its relationship. The x-axis is the year of development. The y-axis on the left is parameter size of AI models. The y-axis on the right is the number of accelerators in AI cluster. To train GPT-3 within 10 days, it needs about 2000 accelerators 4 years ago. Now, the mainstream large-scale AI cluster is built with tens of thousands accelerators. In future, it is expected to use hundreds of thousands accelerators to train larger models, such as GPT-6.



With the scale of AI cluster growing bigger and bigger, power consumption and network cost become challenges.

The power requirements for large-scale AI clusters are substantial due to the significant computational resources(mainly accelerators) needed. A report by Schneider Electric points out the introduction of new GPU generations has led to a significant increase in power consumption, despite yielding higher productivity gains. An AI cluster with 22,500 H100 GPUs demands 40,000 kW of power, which could power around 31,000 average U.S. homes[3]. In the typical public cloud deployment, illustrated in the figure below, it comprises regions, available zones (AZs), and data centers (DCs). AZs are distinct, physically separate locations within a cloud region, each functioning as a logical data center supported by one or more physical data centers. Each AZ is equipped with its own power system, networking infrastructure, and connectivity. AZs within a region are interconnected in order to enhance the resilience, fault tolerance, and reliability of cloud-based applications and services by providing redundancy and isolation. However, the deployment of large-scale AI cluster within an AZ is restrained by the power supply of the location. As AI clusters continue to expand in size and scope, it becomes increasingly challenging to identify locations with the required power capacity. Microsoft has already met the problem. Microsoft engineer complains that they cannot put more than 100K H100s in a single state without bring down the power grid. [2]

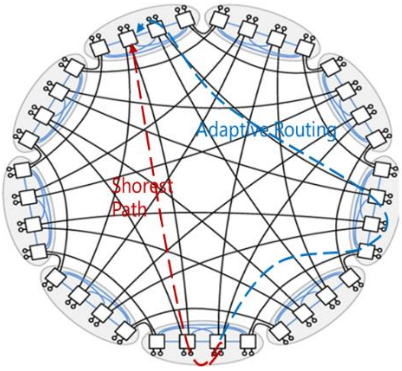


It has to do AI training across different datacenters/AZs. Therefore, long distance DCI transmission is involved. When training is running within a single location, it is relatively easier to control the traffic in AI cluster. The focus is how to get large bandwidth. People build regular topology, use non-blocking network, and run jobs in AI cluster. But when the training job is across different locations, those friendly pre-conditions will disappear. First, AI traffic cross AZs is mixed with other application traffic. The burst of AI huge traffic can easily conflict with other applications' traffic, causing network congestion. Second, distance of DCI is much longer. It can be tens of kilometers or hundreds of kilometers depending on where the datacenters are located. Current congestion control will react slowly because it takes longer time to sense the congestion and take proper reaction. Third, it turns to be over-subscribed network. The oversubscription ratio could be 1:10. Thus bandwidth bottleneck will impact the performance.

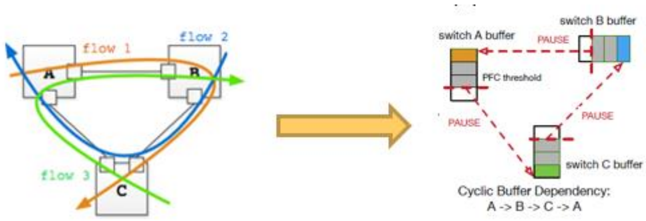
Another challenge is the network cost of AI cluster. Fat-tree topology, while popular in traditional datacenter network encounters cost issues when scaled up for large AI clusters. Fat-tree is good for diverse traffic pattern. Its structured approach considers the worst case patterns. But that is not efficient for AI traffic which is predictable. As network scale grows, the hierarchical nature of fat-tree requires a significant number of switches and extensive cabling, leading to increased hardware expenses. According to HOTI 2023 keynote speech by Nvidia[4], network cost per node(<5k endpoints) in fat-tree topology is about 2 times of cost per node in dragonfly topology. Moreover, the complexity of managing such a vast network adds to operational costs, as adding or removing nodes affects the overall topology and may require significant reconfiguration.

This has led industry to explore optimized topologies in order to have a more cost-effective model for AI cluster. Such works include direct topologies and optical interconnects. Dragonfly/Dragonfly+ and Torus are typical direct topologies. They are scalable and flexible, requiring fewer switches than the traditional fat-tree topologies. Optical interconnects provide higher bandwidth, lower latency, and improved energy efficiency compared to traditional electrical interconnects. Some examples are 3D torus and optical spine switches used by google[6], and Nvidia proposed Dragonfly+ that uses OCS (optical path switching) to provide interconnect between groups[4].

But those optimized topologies have their own challenges when deployed for Ethernet/IP traffic. There are multiple redundant paths between nodes in the new topologies. Unlike the FatTree topology, where paths follow a consistent pattern and have uniform lengths, the paths in these optimized topologies can vary in length and routing behavior. Take Dragonfly topology for example. In a dragonfly network, nodes are organized into groups, and each group is connected to a set of switches. As illustrated in the figure below, the red path represents the preferred shortest path between two nodes. Once the shortest path becomes congested, adaptive routing is executed, shifting the traffic to alternative paths such as the blue path. The blue path provides a viable alternative route to ensure continued connectivity under congested conditions, but it is longer and has different routing behavior.



Such irregularity of paths increases risk of PFC deadlock. PFC deadlock is caused by CBD(cyclic buffer dependency), as shown in the figure. Flow 1, 2 and 3 use the same priority queue. The paths of the 3 flows overlap. Once any one of switches A, B, and C starts PFC due to congestion, PFC deadlock may occur. In fat-tree, it uses equal-cost paths and does not form CBD. Only when a link failure happens, the traffic is re-routed causing CBD. It is easy to identify the re-routing by observing the direction of traffic path. 802.1Qcz[7] provides topology recognition function for fat-tree topology assisting to determine traffic path direction, thereby breaking CBD. However, the topology recognition is not applicable for the optimized topologies. One reason is direct topologies do not have layers as fat-tree. Another reason is optical interconnects may be reconfigured on demand to suit traffic pattern which changes the topology. Furthermore, the traffic re-routing is no longer an indicator of CBD. Because traffic paths are not regular in optimized topologies and adaptive routing makes it more complicated. Developing a method adaptive to different topologies is critical to solve PFC deadlock issue.



Efficiency

<< Communication costs hinder linear expansion of computing power

- Communication overhead
- Bandwidth constraints

>>

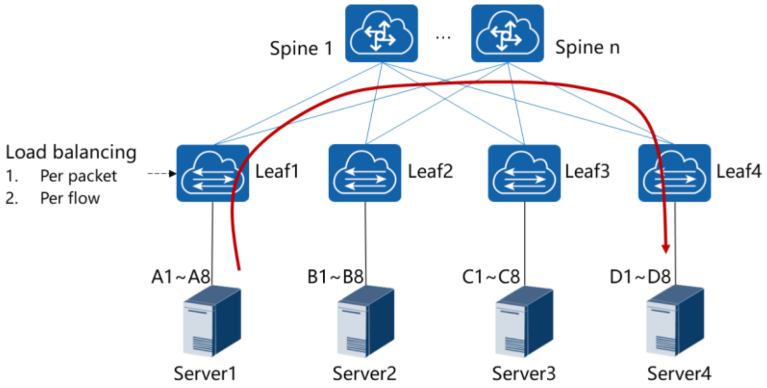
- ✓ Load balancing

<<Describe How LB benefit improving efficiency
AI workload feature: few Elephant flows with Low entropy.→ECMP works badly→ a more fine-grained load balancing scheme. >>

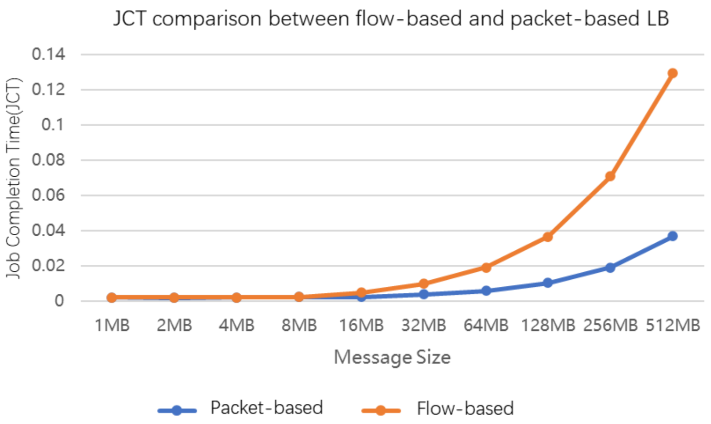
Modern datacenter networks generally provide multiple forwarding paths for each end pairs. Load balancing (LB) is a kind of technologies aiming at fully utilizing these redundant paths. LB can effectively relief the congestion hotpot intra network and network fault, raising the overall throughput by distributing flows or packets among multiple paths.

The effectiveness of a LB scheme is tightly related to the network traffic pattern. As analyzed in the former chapter, the AI traffic is mainly composed of a small number of large bandwidth flows. It's hard for the most conventional LB algorithm ECMP to evenly distribute few elephant flows restricting by its flow-based granularity. It is almost coming into a consensus that AI network need a more fine-grained load balancing scheme to service these elephant flows. Per-packet LB solution is widely considered as the technology trend to avoid per-flow LB's drawbacks for AI network.

The work of [14] conduct a simple experiment to verify that per-packet LB performs better on Job completion time (JCT) than per-flow.
Experiment settings: The topology is the classic two-layer clos network. There are 4 servers. Each server has 8 GPUs and 8 NICs. Running 8 jobs that is between A1 and D1, A2 and D2, ⋯ A8 and D8 respectively.

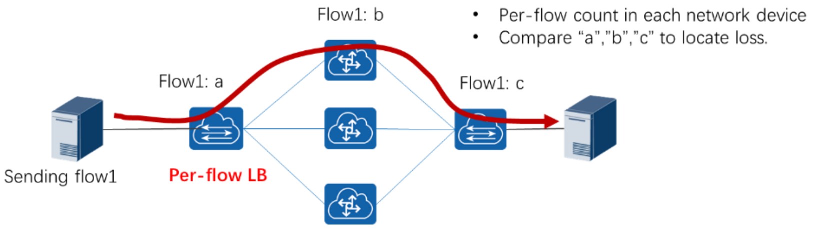


Results: Figure shows the JCT of per-flow and per-packet LB under different message size. The per-packet LB achieve shorter JCT obviously with the message size increasing. When the message size is 512 MB, JCT of per-packet LB is about one-third of flow-based LB.



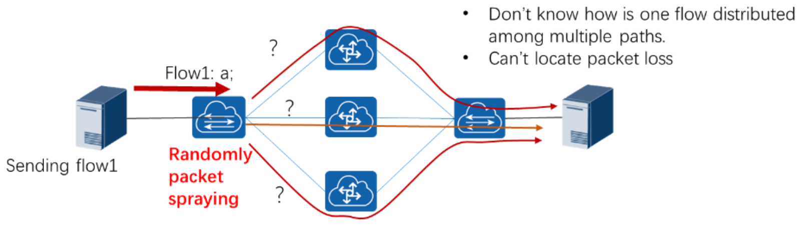
The fine-grained per-packet LB achieve better load balancing effectiveness, but resulting in packets of a flow arriving at receiver out of order. The change from network in-order to out-of-order delivery makes troubles, which mainly in three aspects.

1) Re-ordering: End-side device or NIC may need to re-order out-of-order packets, which causes severe scalability challenges especially for hardware-based protocol like RDMA.
2) Packet loss can't be detected fast: The receiver can't quickly distinguish packet loss or delay based on packet sequence number, that lowering the efficiency of packet loss recovery. In flow-based LB scenario, packets are expected to arrive in order and the loss recovery protocol of RNIC (Go-back-N or selective repeat protocol) interprets an out-of-order packet as an indication of packet loss, which can quickly activate retransmission when loss happen [15]. If network enable packet spraying, out-of-order packets are normal, hence RNIC can only rely on the timeout mechanism to detect packet loss that is obviously inefficient. That means the performance penalty of packet loss in per-packet LB network may be more than per-flow LB network.
3) Packet loss can't be located under silent network faults: Silent network faults is a kind of network device faults that can't detected by device itself and don't generate any alarm information. This kind of faults may be caused by chip soft failure, forwarding table entry failure and so on. The silent failures can cause silent packet loss that do significant damage to performance. People usually adopt In-band flow measurement technologies (e.g., IOAM, IFIT) to accurately locate silent packet loss of a flow under the condition that a flow is only forwarded by one path. The basic idea is illustrated in the below figure. Each device count per-flow in certain method. As the forwarding path is determined, the packet loss can be located by comparting the flow count of each device of this path.



- Per-flow count in each network device
- Compare "a","b","c" to locate loss.

Under packet spraying, packets of a flow may randomly be forwarded by all available paths. It's hard to locate the silent packet loss using the existing in-band flow measurement as having no knowledge about how is one flow forwarded among multiple paths.



Availability

Components in large scale system frequently fail.

✓ Fast Failure recovery

AI fabric's requirement on failure recovery

Future technologies

Approaches to avoid PFC deadlock

Packet-spray load balancing

Standard considerations

References

[1] https://zhuanlan.zhihu.com/p/688178908
[2] https://mp.weixin.qq.com/s/y6B9vM13byV8KT9A3fRiDA
[3] https://www.powerelectronicsnews.com/schneider-electric-predicts-substantial-energy-consumption-for-ai-workloads-globally/
[4] https://www.youtube.com/watch?v=napEsaJ5hMU
[5] https://www.theregister.com/2024/03/18/nvidia_turns_up_the_ai/
[6] https://arxiv.org/pdf/2304.01433
[7] Jason Wei et al. "Emergent Abilities of Large Language Models" Transactions on Machine Learning Research (2022)
[8] Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." Advances in neural information processing systems 32 (2019).
[9] https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)
[10] https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf
[11] Shibo Wang et al. "Overlap Communication with Dependent Computation via Decomposition in Large Deep Learning Models" ASPLOS (2023)

[12] Xiaonan Nie et al. “HetuMoE: An Efficient Trillion-scale Mixture-of-Expert Distributed Training System”
[13] Dmitry Lepikhin et al. “GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding”
[14] 1-24-0004-05-ICne-load-balancing-challenges-in-ai-fabric.
[15] Song C H, Khooi X Z, Joshi R, et al. Network Load Balancing with In-network Reordering Support for RDMA[C]//Proceedings of the ACM SIGCOMM 2023 Conference. 2023: 816-831.