Workgroup:	Working Group name
Internet-Draft:	draft-aft-ai-traffic-00
Published:	7 October 2024
Intended Status:	Standards Track
Expires:	10 April 2025

Authors: A. Fressancourt *Huawei* L. Iannone *Huawei* D. Lou *Huawei* D. Trossen *Huawei*

Handling inter-DC/Edge AI-related network traffic: Problem statement

Abstract

The growth in terms of number of parameters of LLM models as well as the need to use or train those models with private or protected data will require service providers operating LLM-based services to cooperate to train, specialize or serve LLM-based services accross datacenters. Given their structure, the number of parameters they incorporate and the collective communication librairies they are built with, LLM training and inference (or serving) network traffic has specific characteristics.

In that regard, understanding the specificities of AI-related workloads is critical to determine how to operate AI-based services in a federated setting across datacenters.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

AI has attracted a far bit of attention in the networking community, including the IETF, in regards to not just applications but also needed protocols and technologies for the realization of those applications.

While AI is a large area, this document focuses on the method of (training and inferencing over) Large Language Models (LLM). The starting point being the distributed nature of implementing LLMs, both for training and inferencing. For this, a network of so-called 'workers' is being realized, that over time will train a model, over which in turn inferencing can be performed. This distributed nature involves a number of communication patterns to exchange suitable information, those patterns needing realization in suitable protocols. Differences in those protocols emerge from the deployment choices for AI platforms and the challenges that arise from those deployments.

The training of LLMs in ever growing large-scale data centres (DCs) is a current method to push the boundaries of key performance indicators, most notably the number of parameters included in the training model [LLMSize]. Observations in recent works [AIBackbone], however, point to the fact that distribution across more than one DC will quickly be necessary to continue the growth of LLM training.

LLMs may also start inherently distributed in deployment itself, not because of their size, but because of their use case. An example here is content moderation in fediverse environments [FEDI], where decentralized data governance and computation drives the equally decentralized realization of the workers developing a shared model over time. Other examples of such decentralized use case stems from health (e.g., where independent health trust may train a shared model over strictly locally governed patient data), IoT (where AI-derived features may be derived from localized sensor data), or also network management (where operator data for training may not be possible or legally permitted to be disclosed to a centralized training entity). Realizations of platforms for those use cases often refer to 'federated learning' [FLOWER] to capture the equal basis of participation, both at the data and processing level, of the participating entities of the platform.

The intention of this document is to highlight the possible challenges in the spectrum of realizations for the distributed LLM training and inferencing. For this, we provide more details and examples for use cases in Section 2, followed by a primer of key AI/LLM techniques in

Section 3. Then, a number of challenges to, e.g., resource management, latency, and security, are identified in Section 4 as a starting point for a focussed discussion in the IETF community on challenges AI may pose for networks, networks protocols and technologies.

As the spectrum of realization ranges from centralized intra-DC to highly distributed, federated AI platforms, there is a strong relevance of solving the identified challenges within the IETF, leading to the formulation of a problem statement in Section 5 along those lines and proposing next steps in Section 6 for the IETF community to pursue.

2. Applicability of AI

This section introduces the applicability of AI/LLMs across a spectrum of decentralization for realising machine learning training and inferencing tasks. These two tasks are the most intensive in AI workflows, and they are inherently distributed compute problems given the amount of memory and processing power they require Section 3 will introduce the prominent techniques relevant to implementing those tasks in deployed systems.

It can be observed upfront that the realization of AI/LLM use cases is driven by two main aspects, namely the decentralization of data and that of compute; during the training phase, the inference phase or both. The proposed examples are introduced with regards to those two aspects, for reasons that are detailed in the text.

2.1. xGPT-like Services: an example centralized training use case

Various GPT-like services have gained much attention recently, advocating the training of ultra-large language models, providing chat-like inferencing services to users for, *e.g.*, text generation, generative AI-based image creation, and many others.

The key success factor of those services is the ingestion of vast amounts of data into a centralized training entity. This data may come from public sources, search input, through license arrangements with, *e.g.*, video providers like YouTube, and many others.

In this use case, centralization pertains to the ownership of the training resources under a single entity. Until now, the prevalent deployment of such centralized training is within a single, large-scale DC with a growing number of GPUs to execute the necessary model training operations over a sustained period of time. However, the growing need for more compute, together with physical and energy limitations in existing DCs, make a growing scale-out of those services beyond the reach of a single DC a likely needed path in the future.

2.2. Decentralized training with centralized inference use cases

In some cases, it is impossible to gather the amount of data necessary to properly train a machine learning model, either for security, privacy or regulatory reasons. The necessity to tackle those cases triggered the development of Federated learning [FederatedLearning], in which several entities can collaborate in a decentralized way to the training of a single large model.

Health is a traditional area for reasoning-based systems. Its richness in data, both historical as well as current (*e.g.*, from patients), lends itself for training LLMs over which inferencing can take place for, *e.g.*, diagnosis, drug development, developing treatment plans and many others.

Individual health organizations are often well equipped in terms of compute resources, albeit not at the scale that suffices to perform centralized LLM training on their own. Thus, federating compute resources is useful for scale and incentivized through the sharing of the resulting LLM in the clinical pathway.

Furthermore, data is also localized, since patients sign up to local health organizations, practises, or health trusts, which in turn manage their data. In many countries, data sharing is performed, also for transferability (*e.g.*, when patients change location and thus local health contacts) but also treatment across health organization. Overall, data governance needs to be strictly adhered to for any application of the data, with a possible development of a LLM being just one such application.

Network management is another use case for federated learning, where the federation is driven by the common goal to develop an increasingly refined model for, *e.g.*, intrusion detection, network failure detection, and others, but where suitable training data is only shared in a limited manner or not at all for confidentiality reasons.

In those use cases, the decentralization of the training stems from the constraints limiting the exchaneg of data between entities for technical feasibility, privacy, confidentiality or regulatory reasons. Once trained, the models including the common knowledge gathered in the training phase can be used by each single entity without necessarily requiring collaboration during the inference phase.

2.3. Federated building management: a decentralized inference use case

Building management, where smart buildings are often equipped with micro-DC capabilities, can be federated to improve energy management in a larger geographic area, utilizing local yet private data to train a suitable (possibly locally limited) LLM. Works like [AIConst] expand this use case into other areas like construction management, architectural design, material design, and others.

Similar to the previous use case, the deployment across shared infrastructures (as well as the sharing of compute resources itself) is a key aspect, utilizing Internet technologies for realizing the necessary exchange within the training and inferencing scenario at hand. Here, the decentralization of the inference task is a necessity given that the goal is to reach a global optimum (*e.g.*, energy savings accross an aea or region) by clustering the capabilities of buildings while keeping the data used for training and inferencing local for security and privacy reasons.

2.4. Key Takeaways

The following key takeaways can be derived from the proposed applicability examples:

- *LLM AI is inherently distributed*: This is due to its scale in required training nodes and model size. Although the distribution may be handled entirely within a central, single data centre, the execution of tasks across, often many thousands, of workers still remains a key aspect in LLM AI.

- *Centralized LLM AI training does not need to touch the Internet*: This is because it may be performed entirely within the limits of a single DC, or at least among a few DCs interconnected by private lines under the control of a single entity.
- *Centralization of compute implies centralization of data*: This is a consequence of the model creation, based on the data, which is centralized and thus requires suitable data transfer towards the compute nodes.
- *Federation allows for decentralization of data*: This is possible with worker nodes that may individually train on data, while contributing to the global model. However, centralization of the federation again leads to the observation in the second item, *i.e.*, data is centralized, too.
- *Inferencing inherently touches the Internet*: This is especially true in any applicability scenario involving end users residing on the Internet. The impact may the creation of very large amounts of traffic to (*e.g.*, centralized) nodes that hold the suitable model information for performing the desired inferencing.

The next section outlines the system aspects of LLM AI, in light of the above takeaways, *e.g.*, on centralization, while Section 4 more directly takes these takeaways into account when formulating key challenges for networking, stemming from the insights in this section.

3. How ML systems are built

3.1. Lifecycle of a typical large language model

In the last few years, the impressive AI boom has been associated with the development of Large Language Models (LLMs). LLMs are models, or representations of systems that are observed or measured with data. Models have an underlying structure, consisting in a parametrized graph linking together small operators performing operations on **tensors**, or small arrays of finite dimensions. Models are built, or **trained**, using a reference data set (or training set), consisting in data labelled with its underlying structure or meaning.

Before its use for training, the data is collected and pre-processed to structure it and chunk it into **tokens**, the basic data units used as input and outputs of models. After the training data set has been prepared, the model is trained. During this training phase, the model is parametrized, i.e., the parameters ruling the strength of the relationships between the tensor operators constituting the model are modified so the model is able to abstract the behaviour of the system from the dataset representing it.

After the training phase, the model can be used during an **inference** phase to derive information or make predictions from new data presented in the form of a sequence of tokens. Inference operations are not necessarily done by the same nodes that have trained the model. Models can be transferred to other worker nodes to perfom inference tasks, sometimes placed close to the users making requests. Besides, those transferred models can be re-trained or fine-tuned in order to better serve requests in the context they are done. This can be done for instance with private or locally relevant data.

3.2. The distributed nature of Machine Learning systems

To improve the accuracy of LLMs, those models incorporate an increasing amount of parameters, and are trained using ever larger datasets. Modern LLMs use 125 million up to 405 billion parameters, where each parameters can be encoded with 4 bits to 4 bytes depending on the required precision. This increase in models' sizes has important consequences on the power consumption and on the memory footprint of the systems used to train those models.

From a power consumption perspective, the increase in computing power needed to accomplish the training tasks of large models requires the deployment of more powerful Tensor Processing Units (TPUs) of Graphics Processing Units (GPUs) with higher power and cooling demands. It is becoming physically unsustainable to bring the power required to datacenters hosting those machine learning worker nodes. A possible way to address this challenge is to distribute machine learning workloads beyond the realm of a single datacenter, possibly between two or a few interconnected datacenters separated by long to mid range private interconnections.

From a memory perspective, this means that storing a model and its parameters will cost from 62,5 MB (for a model using 125 million parameters with 4-bits parameters) to 1620 GB (for a model using 405 billion parameters with 4 bytes parameters). The memory footprint of modern large language models makes those models difficult to manipulate on a single node. Besides, as mentionned in [SchedulingSharding], the pre-training of Llama3 with 405 billion parameters required from 8192 to 16384 GPUs, which were obviously hosted on multiple, connected nodes.

To cope with the memory and computing needs of workloads associated with LLM operations, the data preparation, training and inference tasks mentionned in Section 3.1 can be distributed among nodes with specific functionnal roles (task execution, result aggregation, orchestration...). The detailed description of those functional roles is given in Appendix A.2.

From a networking perspective, some elements are important to mention:

- Some roles (aggregator or orchestrator) tend to concentrate network traffic, thus creating some issues in properly managing the incast traffic.
- The parallel execution of training and inference tasks by worker nodes is following specific parallelism modes presented in Figure 1 and detailed in Appendix A.3. Those modes have various requirements in terms of volume of data exchanged and latency (see [SchedulingSharding] for instance).

  a. **Data parallelism**, in which data is split into chunks to be used by different worker nodes to train the model. This parallelism mode can cope with rather large latencies (several 100s of ms) but requires exchanging a large volume of data.

  b. **Pipeline model-parallelism**, in which a model is separated into stages consisting in a few consecutive layers of the entire model. Each stage is allocated to a separate worker node, and intermediate states are exchanged between successive worker nodes. This parallelism mode can cope with large latencies (10s of ms) and requires exchanging less data than data parallelism.

c. **Tensor model-parallelism**, in which model layers are split into chunks that can be operated by different nodes. This parallelism mode needs to operate in low latency networks (10s of us) and requires exchanging a lot of data.

d. **Mixture of Expert parallelism**, in which nodes are holding smaller but specialized models trained over a smaller amount of data and holding fewer parameters. This parallelism mode can cope with latencies in the ms range.

- Machine learning applications rely most of the time on collective communication libraries [xCCL] that use patterns presented in details in Appendix A.4 such as All-to-All or All-reduce. [I-D.yao-tsvwg-cco-problem-statement-and-usecases] has already introduced some networking challenges associated with the use of collective communication in machine learning systems. From a networking perspective, those communication patterns translate in "on-off" communication patterns, with a few large flows starting simultaneously to allow nodes involved in a collective to exchange data and parameters. The volume, synchronism and imbalance of the traffic generated by those communication pattern is a burden to the distribution of machine learning workloads, and a challenge to be addressed by the networking community.

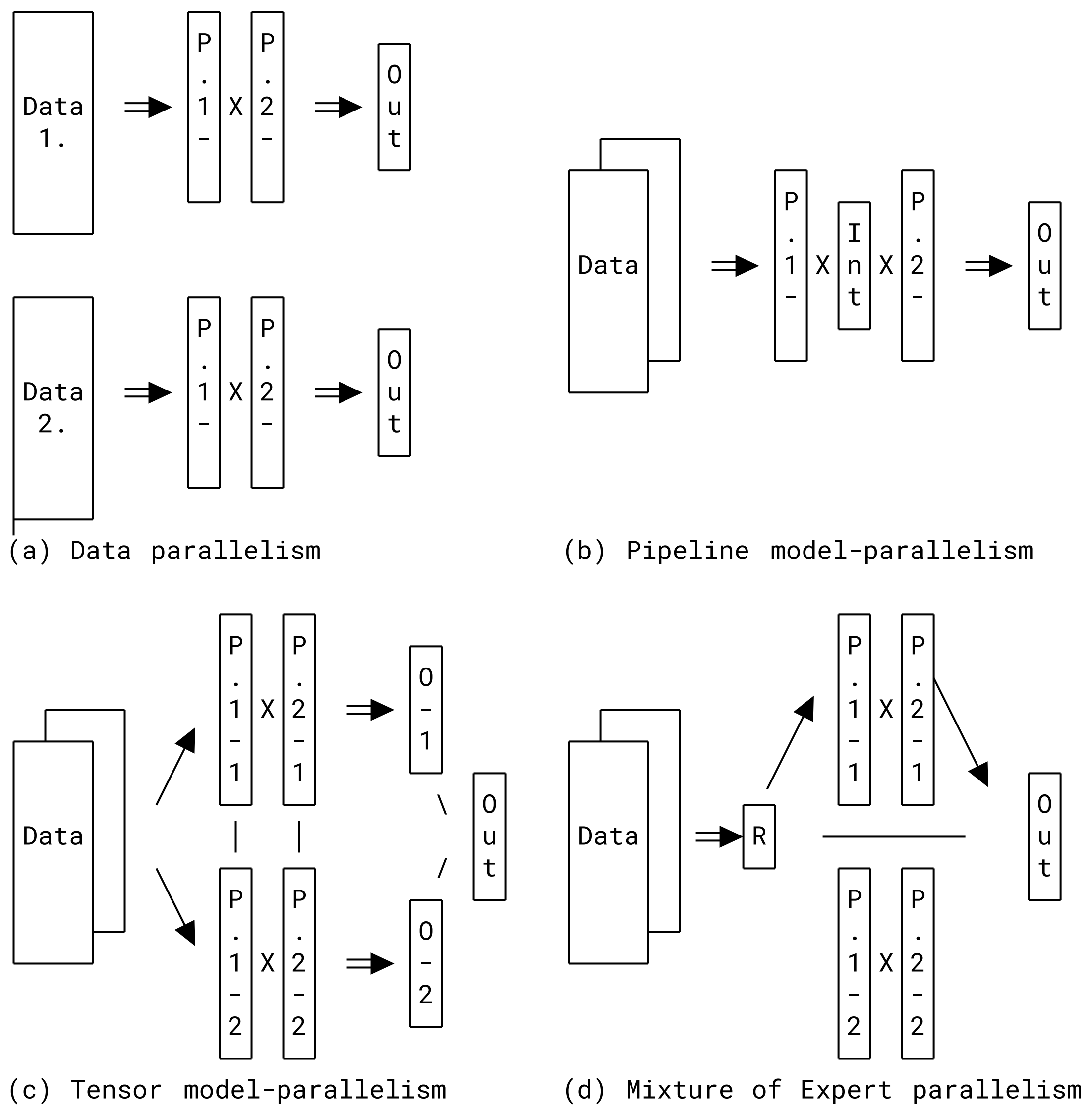

Figure 1: Parallelism models used in machine learning systems

3.3. The topology of distributed machine learning systems

To boost the workload distribution efficiency, nodes participating in the common execution of machine learning workloads are interconnected following specific topologies. Depending on the number of nodes involved, the nodes' capacities to connect directly with other nodes and the control of the machine learning application administrator over the physical connectivity substrate, difefrent topologies can be used. Those topologies can inherit from the work done either in the high performance computing community if the nodes are homogeneous, high capacity computers under the control of a single administrator or in the peer-to-peer or swarm computing community if the nodes are more diverse and numerous.

Topologies inspired from the HPC networking community are typically found in distributed systems in which all the nodes are located in the same datacenter. In specific cases, those topologies can be extended beyond a single datacenter. Then, the distribution of the workloads is done so as to avoid frequent and latency-bound communications over those links, and congestion control mechanisms are tuned to avoid deadlocks and bloats in network equipments manging the inter-datacenter link. Among those HPC-inspired topologies, we can find the n-dimension torus, the Dragonfly topology,the fat tree topology and the hypercube topology.

On the other end of the spectrum, topologies inspired from swarm computing and peer-to-peer networks have emerged as solutions connecting heteorgeneous nodes involved in distributed machine learning tasks outside datacenters, at the edge of the networks or among clusters of nodes hosted on public cloud platforms. Those topologies can be built either from dynamic wireless connections, which can be rearranged easily, or as overlay links on top of an immutable physical connectivity substrate. Among those swarm-inspired topologies presented in Figure 2, we can find the full mesh, the star topology, the Peer-to-Peer topology which presents a limited diameter without too many connections, the random topology and the clustered topology. Note that in the case of the clustered topology, nodes can be gathered based on their proximity to a proxy or an aggregator, or based on their interest or capabilities. Besides, the clustered topology can be structured according to a hierarchy.

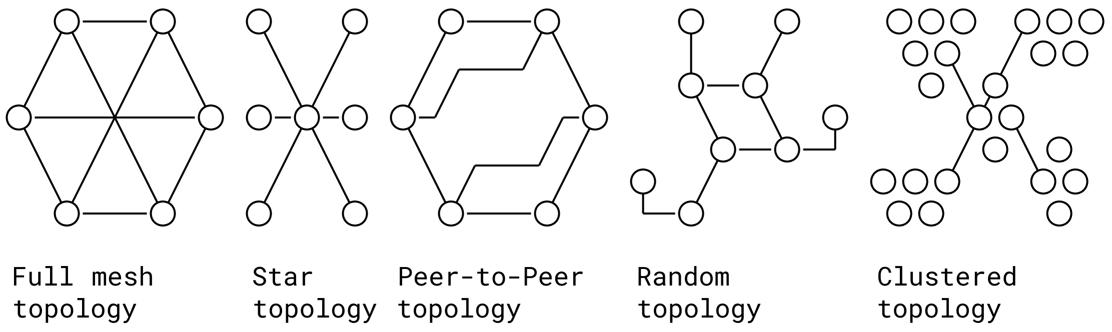

Figure 2: Decentralized topologies

The topology connecting the nodes together is not anecdotical. If it is under the control of the machine learning system administrator, the construction of a proper topology able to efficiently support the communication patterns generated by the parallelization model and the collective communication method used is a key performance factor. If the topology is inherited from the environment, machine learning system administrators will need to adapt the communication patterns they use to the characteirstics of the topology.

3.4. Deployment considerations for AI systems

Given the computing and memory requirements of machine learning workloads, modern machine learning systems are fundamentally distributed. By combining functionnal roles together, distributing them using a combination of parallelization modes, and communicating following patterns, a variety of systems with different shades of decentralization can be built. Indeed, distributing workloads across several workers does not necessarily means that the control and orchestration of those workloads is distributed. In case a parameter server is used, the orchestrator role is played by a single node.

Yet, following the Federated learning [FederatedLearning] approach, machine learning systems can be decentralized as shown in Figure 3.

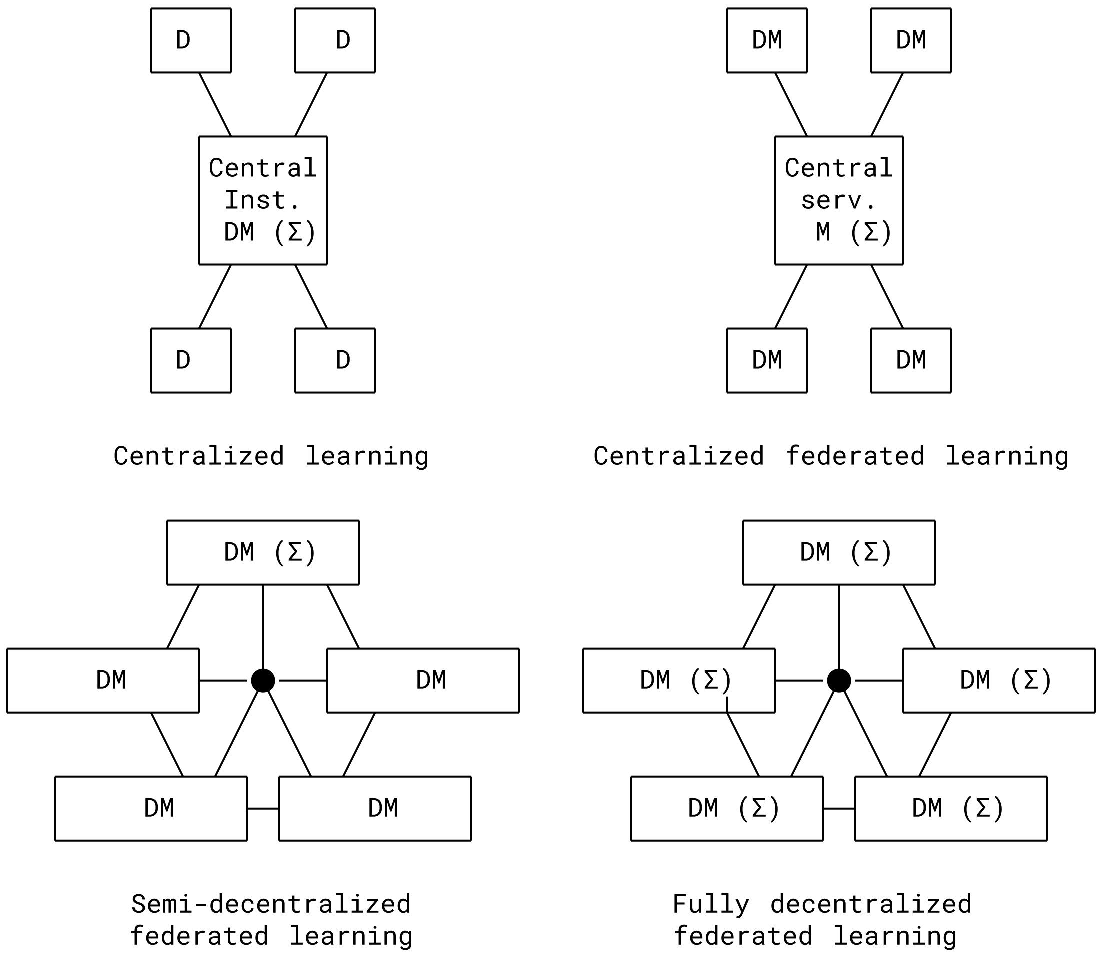

Figure 3: Different centralization models in federated learning

In federated learning, the coordination role of the orchestrator or the aggregation of parameters done by the aggregator can be done by a subset or all the worker nodes deployed in the system, using a distributed protocol to exchange information, synchronize parameters and agree on a reference. Of course, this comes at a communication cost, but decentralization also has benefits.

One of the first reason for decentralizing machine learning systems is to allow users to retain ownership of their data, and to control how their data is transferred or used. In a centralized setting, users need to trust the central entity managing the system for respecting the data management agreed upon with the user. In a decentralized setting, users can either infer from a copy of a generic model sent by a central entity (centralized federated learning), fine-tune the model and infer from it (semi-decentralized setting) or even cooperate with others to train together a model and use it afterwards (in a fully decentralized setting).

Besides, in a decentralized machine learning system, users and worker nodes can be co-located, or workers can be placed close to the user in order to reduce the overall communication latency. In such a decentralization scheme, if it is considered that edge nodes can benefit from less memory or computing power, a balance has to be found between the time spent in communicating between the user and worker nodes and the processing time to reach an optimal response time for user requests. Worker nodes located at the edge of the network may also collaborate with other, more capable nodes located in other locations. The tail latency of the flows associated with those tasks should be bounded to avoid degrading the user's experience.

4. Challenges (in Networking for AI)

4.1. Network resource management at regional scale

In (large) model-based machine learning, training, fine-tuning and infering from the model are workloads that involve the transfer of a very large volume of data. In [AITrainingTraffic], the authors estimate that the first iteration for training a GPT-3B model among 32 GPU nodes generate roughly 85 GB of tensor-parallel data traffic, 1 GB of pipeline-parallel data traffic and 741 MB of data-parallel data traffic. The tensor-parallel and pipeline-parallel data traffic consist in 125 MB messages that are periodically exchanged during communication phases, while very little traffic is exchanged during computing phases.

This traffic pattern is a good example of the characteristics of network traffic flows associated with machine learning workloads. Indeed, the network traffic generated by distributed machine learning systems consists in a relatively little number of large (elephant) flows, starting and stopping roughly simultaneously. Such synchronous traffic patterns are coming from the use of collective communication libraries such as NCCL and associated collective communication primitive such as All-Reduce or All-Gather, as mentionned in [Burstiness]. Dealing with this synchronized traffic in a stochastic network is challenging, because it generates periodic traffic bursts that bloat network equipment buffers. Besides, the network's capacity needs to meet peak requirements at the cost of an unefficient average utilization of the installed capacity. Even if the network's capacity is sufficient in theory to accomodate the peak requirements of machine learning systems, the transport mechanisms used on the links between the nodes make it difficult to immediately use the full deployed capacity, resulting in inefficiencies at the network level [NetworkBottleneck] At last, in such a synchronous communication pattern, the failure of a link, delaying data transmission between two nodes in the system might delay the whole system after a few iterations.

Mitigating the resource management challenges raised by machine learning traffic patterns is currently an open research area. At the application level, machine learning system designers work to develop hybrid parallellization schemes combining the patterns presented in Section 3.2 (detailed in Appendix A.3) and orchestration methods aiming at better utilizing the deployed network capacity and at avoiding "on-off" behaviors.

At the network level, the main challenge that the research community is trying to address is the proper balancing of the flows generated by machine learning workloads in the network. Indeed, to address the need for bandwidth and avoid bottlenecks, machine learning system designers are often deploying their system on top of networks whose topology presents a large number of equal cost links between two nodes. Yet, as mentionned earlier, machine learning traffic is constituted with a rather small number of large flows that have a very small entropy. This makes applying classic load balancing techniques challenging. To address this load balancing issue, most collective communication libraries use packet spraying strategies, which require specific tuning due to mentioned lack of entropy. Yet, some researchers are questionning the relevance of this approach [ChallengingSpraying].

4.2. Latency sensitivity of LLM training and inference

Training a (large) model in a distributed setting does not necessarily require data transfer to be operated at a controlled low latency, but large tail latencies related to packet losses or head of line blocking can delay the training of models considerably. Indeed, problems arising from packet losses, link failures or excessive buffering can have cascading consequences since in most parallellization methods, data and model parameters are exchanged following synchronized patterns (see Section 4.1).

The extend of the amplification of the latency effects of a soft failure on a connection between two nodes depends on the topology of the network connecting the nodes. Besides, routing inefficiencies or failures to properly balance the load on some heavily-used links can also generate additional latencies. Thus, the topology of the network supporting machine learning workloads needs specific care.

In a large scale and decentralized AI context, heterogeneous links can be used between nodes participating in a decentralized machine learning system. The specificities of the links need to be taken into account when orchestrating or distributing AI-related tasks. As latency is affected by congestion, addressing the mismatch between links' bandwidth-delay products for efficient congestion management is an open challenge. In the research community, some projects have proposed to introduce proxies to investigate new control loops taking into account link segments characteristics [SiteToSite]. In the IETF, CSIG draft (now expired) presented a model to expose a rich congestion signal [I-D.ravi-ippm-csig].

4.3. Aligning communications patterns to the Internet's constraints

In the development and deployment of distributed machine learning systems within the real of an administrator's responsibility, it is feasible, and advised, to design the system to adapt the network's topology to the paralellization method used and to the collective communication patterns required to perform the training or inference task efficiently. As we have seen in Section 3, several network topologies can be adopted, depending on the model's architecture and on the design choice made while designing the training system.

In a decentralized or federated setting, such a co-design offers lmess freedom, as some adverserial choices can be made by people willing to cooperate on specific machine learning tasks. Besides, when the nodes involved are deployed at the edge of the network, in a majority of the cases, the topology is following the access network's specificities rather than adapting to the requirements of machine learning tasks.

As machine learning-related traffic is growing on the Internet, some challenges associated with the delivery of network flows related to the realization of decentralized training or inference tasks are appearing: How to inform a machine learning application about the topology of the network interconnecting involved nodes? How to adapt the parallelization model or collective communication pattern to maximize efficiency? In the IETF, the ALTO working group [ALTO] has investigated similar challenges related to the operation of peer-to-peer traffic. As machine learning workloads have a different set of requiremenst, it may be time to revisit this work.

4.4. Managing incast traffic related to AI inference

In a machine learning system, some specific nodes, such as the orchestrator of the aggregator, play a central role, and thus concentrate communications. This concentration might be reinforced by the use of specific collective communication primitives, and by the synchronicity of traffic patterns. The network traffic management challenges we previously mentionned (Packet losses, delays, congestion) can be amplified by this concentration on a few hotspots in the network.

Decentralized systems tend to mitigate this functionnal centrality by distributing the responsibility for fullfilling those roles accross several nodes. Yet, even in decentralized systems, it is difficult to completely avoid incast problems given the specificities of machine learning workloads. For instance, due to the characteristics of model inference, first prompt requests addressed to a given node are bound to generate a large incast traffic related to the personalization and fine tuning of the model instance run by the node, which can represent a large amount of data.

The research and operational communities dealing with scaling out machine learning systems are working on some solutions to address incast traffic management issues. For instance, in the same way as media codecs structured as layers of increasing resolution, models used in training and inference can be layered, with coarse grained models using less parameters than finer grained ones. Coarse grain models can be distributed more quickly, drafting a first answer to a request while finer grain models are retrieved and then used to build a more precise answer to a request. Besides, as inference requests towards large models are often done in the form of a converstaion or contextualized exchange between a user and the node running inference tasks addressing its requests, the use of service routing to consistently route requests to the same instance can be used.
In that extend, the work done in the CATS working group in the IETF is of particular relevance ([CATSWG]). Specific work on the proper metrics to be applied to CATS in the context of large

scale decentralized AI will need to be done, taking into consideration the large body of work on load balancing for machien learning workloads done in the research community. In particular, one problem of importance in the selection of the instance serving a specific inference request is the trade-off to find between the need to serve the requests with a bounded latency, requiring to use a node that is located as close to the requesting user as possible, the necessity to bootstrap this node with the user's contextual model parameters and the uncertainty about the length of the session between the user and the instance, which puts a stress on the memory needed to keep the session's context at hand.

4.5. Securing and attesting AI-related traffic

The distribution of machine learning workloads at a regional scale beyond the limits of a single datacenter and the decentralization of the control and management of those operations raises important security and privacy challenges. Indeed, when those workloads are operated in a single tenant datacenter, data can be secured by means of perimeter security measures, or adopting encryption at least for data stored in the datacenter's realm. In the same way, it is acceptable to exchange data between nodes in an unencrypted manner provided the network on which data is exchanged is isolated from the outside environment.

When data, models and their associated parameters are exchanged on the Internet, or on public connections, data managers need to make sure that the data is exchanged securely, and following policies complying both with users preferences and local regulations. As previously mentionned, data exchanges done during model training phases or directly before performing an inference task need to be done as quickly as possible, avoiding tail latencies as much as possible. Even if encryption of large data flow has improved considerably in the last years, it is adding a latency overhead, considering that the computational overhead related to cryptographic operations is handled beyond the chip (often a GPU or a tensor processor) performing the machine learning tasks. It would be interesting and benefitial to contribute to the efforts done in the IRTF and IETF to develop low latency, lightweight encryption schemes in order to reduce this overhead as much as possible.

Furthermore, as private data is involved in data exchanges related to training and inference, specific care must be taken to respect regulations steering the way those data can be processed and transfered. In particular, some data need to be processed in the same region or country they have been generated in. To make sure those regulations are respected, attestation mechanisms need to be put in place. By using those attestation mechanisms, data owners or managers can prove to local authorities that the data they are managing is being exchanged according to the policy they specified. In the IETF, the RATS working group [RATSWG], and the NASR initiative [NASR] are developping attestation mechanisms that could be adapted to address machine learning requirements, even if their work is still at the beginning.

5. Problem statement

In today's LLM-based machine learning landscape, we observe a strong concentration of training abilities in the hands of a few hyperscalers, while several companies tend to propose innovative approaches to improve inference by making it faster, and done by machines located closer to the users. Yet, there is a need to distribute both the training and inference workloads of AI.

In the same way as the Internet, there is, and will be several incentives to decentralize some or all aspects of AI's lifecycle to address scalability, resilience, personalization or privacy concerns. This decentralization trend is exemplified by Federated Learning [FederatedLearning], with different decentralization models, as presented in Section 3.4.

Given the challenges highlighted in Section 4, and the fact that multiple stakeholders are involved in properly adressing those challenges, AI-related network traffic will no longer be operated only on private infrastructure, but also on an ***open interconnected network***. Thus, it is desirable that the IETF community discuss networking challenges related to large scale decentralized AI to avoid the deployment of proprietary solutions, or of solutions putting the stability of the Internet at risk due to unfair resource mangement or competition between AI-related network traffic and other traffic in the Internet.

6. Next Steps

While some work addressing the challenges highligted in this document is done in working groups related to congestion management, deterministic networking or service routing, there might be interest in the IETF community at large to aggregate a group of contributors interested in elaborating specific requirements and corresponding solutions to the challenges listed in Section 4. In particular, the goal of such an initiative would be to:

- Formalize AI-related requirements for service routing: Indeed, it is needed to define the metrics to take into account for load balancing AI-related workloads, given the need for instance stickiness or the importance of latency aspects while addressing the specificities of AI-related traffic, consisting in elephant flows with little entropy.
- Formalize low latency / limited latency requirements associated with AI network traffic: As presented in this document, during the different phases of the AI lifecycle, it will be needed to enforce a different set of requirements in terms of latency, tolerance to jitter or packet loss for sustaining network traffic.
- Formalize congestion control aspects related to the operation of AI traffic at regional scale: The machine learning community is already engaged in improvements of the congestion aspects of AI workloads by working on application-level solutions taking for granted the underlying behavior of the network. It would be interesting to determine what are the possible improvements that the network could benefit from in order to better solve the congestion and incast management issues related to the way AI network traffic is managed.

- Formalize coordination aspects of AI distributed systems: This topic is very important to the realization of decentralized, large scale AI systems, and to the emergence of an inter-AI network of entities collaborating in the execution of end to end AI workloads for end users.

7. Security Considerations

Section 4.5 highlights privacy related challenges that AI operations at regional scale will have to address. Beyond this section, no additional security concern is raised by the elements presented in the document.

8. IANA Considerations

This document has no IANA actions.

9. Informative References

[AIBackbone] Sundaresan, J., Gopalan, A., and Meta, "AI impact on backbone", <https://atscaleconference.com/videos/ai-impact-on-backbone/>.

[AIConst] Baduge, S., Thilakarathna, S., Perera, J., Arashpour, M., Sharafi, P., Teodosio, B., Shringi, A., and P. Mendis, "Artificial intelligence and smart vision for building and construction 4.0: Machine and deep learning methods and applications", Elsevier BV, Automation in Construction vol. 141, pp. 104440, DOI 10.1016/j.autcon.2022.104440, September 2022, <https://doi.org/10.1016/j.autcon.2022.104440>.

[AITrainingTraffic] Li, W., Liu, X., Li, Y., Jin, Y., Tian, H., Zhong, Z., Liu, G., Zhang, Y., and K. Chen, "Understanding Communication Characteristics of Distributed Training", ACM, Proceedings of the 8th Asia-Pacific Workshop on Networking vol. 23, pp. 1-8, DOI 10.1145/3663408.3663409, August 2024, <https://doi.org/10.1145/3663408.3663409>.

[ALTO] "Application-Layer Traffic Optimization (alto)", n.d., <https://datatracker.ietf.org/wg/alto/about/>.

[Burstiness] Luangsomboon, N., Fazel, F., Liebeherr, J., Sobhani, A., Guan, S., and X. Chu, "On the Burstiness of Distributed Machine Learning Traffic", arXiv, DOI 10.48550/ARXIV.2401.00329, 2024, <https://doi.org/10.48550/ARXIV.2401.00329>.

[CATSWG] "Computing-Aware Traffic Steering (cats) Working Group", n.d., <https://datatracker.ietf.org/group/cats/about/>.

[ChallengingSpraying] Addanki, V., Goyal, P., and I. Marinos, "Challenging the Need for Packet Spraying in Large-Scale Distributed Training", arXiv, DOI 10.48550/ARXIV.2407.00550, 2024, <https://doi.org/10.48550/ARXIV.2407.00550>.

[DataParallelism] Valiant, L., "A bridging model for parallel computation", Association for Computing Machinery (ACM), Communications of the ACM vol. 33, no. 8, pp. 103-111, DOI 10.1145/79173.79181, August 1990, <https://doi.org/10.1145/79173.79181>.

[FederatedLearning] McMahan, H., Moore, E., Ramage, D., Hampson, S., and B. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data", arXiv, arXiv, DOI 10.48550/ARXIV.1602.05629, 2016, <https://doi.org/10.48550/ARXIV.1602.05629>.

[FEDI] Anaobi, I., Raman, A., Castro, I., Zia, H., Ibosiola, D., and G. Tyson, "Will Admins Cope? Decentralized Moderation in the Fediverse", ACM, Proceedings of the ACM Web Conference 2023, DOI 10.1145/3543507.3583487, April 2023, <https://doi.org/10.1145/3543507.3583487>.

[FLOWER] Flower Labs GmbH, "Flower - A Friendly Federated Learning Framework", n.d., <https://flower.ai/>.

[I-D.ravi-ippm-csig] Ravi, A., Dukkipati, N., Mehta, N., and J. Kumar, "Congestion Signaling (CSIG)", Work in Progress, Internet-Draft, draft-ravi-ippm-csig-01, 2 February 2024, <https://datatracker.ietf.org/doc/html/draft-ravi-ippm-csig-01>.

[I-D.yao-tsvwg-cco-problem-statement-and-usecases] Yao, K., Shiping, X., Li, Y., Huang, H., and D. KUTSCHER, "Collective Communication Optimization: Problem Statement and Use cases", Work in Progress, Internet-Draft, draft-yao-tsvwg-cco-problem-statement-and-usecases-00, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-yao-tsvwg-cco-problem-statement-and-usecases-00>.

[LLMSize] Zhang, B., Liu, Z., Cherry, C., and O. Firat, "When Scaling Meets LLM Finetuning: The Effect of Data, Model and Finetuning Method", arXiv, DOI 10.48550/ARXIV.2402.17193, 2024, <https://doi.org/10.48550/ARXIV.2402.17193>.

[MoEParallelism1] Jacobs, R., Jordan, M., Nowlan, S., and G. Hinton, "Adaptive Mixtures of Local Experts", MIT Press - Journals, Neural Computation vol. 3, no. 1, pp. 79-87, DOI 10.1162/neco.1991.3.1.79, February 1991, <https://doi.org/10.1162/neco.1991.3.1.79>.

[MoEParallelism2] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and J. Dean, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer", arXiv, DOI 10.48550/ARXIV.1701.06538, 2017, <https://doi.org/10.48550/ARXIV.1701.06538>.

[NASR] "Network Attestation for Secure Routing (NASR)", n.d., <https://datatracker.ietf.org/doc/bofreq-liu-nasr/>.

[NCCL] Nvidia, "NVIDIA Collective Communications Library (NCCL)", n.d., <https://developer.nvidia.com/nccl>.

[NetworkBottleneck] Zhang, Z., Chang, C., Lin, H., Wang, Y., Arora, R., and X. Jin, "Is Network the Bottleneck of Distributed Training?", ACM, Proceedings of the Workshop on Network Meets AI & ML, DOI 10.1145/3405671.3405810, August 2020, <https://doi.org/10.1145/3405671.3405810>.

[PipelineParallelism] Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N., Ganger, G., Gibbons, P., and M. Zaharia, "PipeDream: generalized pipeline parallelism for DNN training", ACM, Proceedings of the 27th ACM Symposium on Operating Systems Principles, DOI 10.1145/3341301.3359646, October 2019, <https://doi.org/10.1145/3341301.3359646>.

[RATSWG] "Remote ATtestation ProcedureS (rats) Working Group", n.d., <https://datatracker.ietf.org/wg/rats/about/>.

[RCCL] AMD, "AMD ROCm Software", n.d., <https://github.com/ROCm/ROCm>.

[SchedulingSharding] Chu, W., Choudhury, A., and Meta, "Scheduler and Sharding Considerations for Network Efficiency", <https://atscaleconference.com/videos/scheduler-and-sharding-considerations-for-network-efficiency/>.

[SiteToSite] Cangialosi, F., Narayan, A., Goyal, P., Mittal, R., Alizadeh, M., and H. Balakrishnan, "Site-to-site internet traffic control", ACM, Proceedings of the Sixteenth European Conference on Computer Systems, DOI 10.1145/3447786.3456260, April 2021, <https://doi.org/10.1145/3447786.3456260>.

[TensorParallelism] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", arXiv, DOI 10.48550/ARXIV.1603.04467, 2016, <https://doi.org/10.48550/ARXIV.1603.04467>.

[xCCL] Weingram, A., Li, Y., Qi, H., Ng, D., Dai, L., and X. Lu, "xCCL: A Survey of Industry-Led Collective Communication Libraries for Deep Learning", Springer Science and Business Media LLC, Journal of Computer Science and Technology vol. 38, no. 1, pp. 166-195, DOI 10.1007/s11390-023-2894-6, February 2023, <https://doi.org/10.1007/s11390-023-2894-6>.

Appendix A. A primer on Machine learning (extended version of *[Section 3]*)

Along its development, Machine Learning (ML) involves the use of an increasing amount of computing and storage resources in the operations of algorithms taking decisions or deriving insights from data. In recent generative AI algorithms, or in large language models, the amount of data used to train models of increasing size in terms of parameters has grown exponentially. Besides, the size of large models translates in an increasing memory and computing footprint. Given this evolution, ML algorithms can not be executed or trained on a single machine, and thus ML systems are "distributed" by nature, regardless of the architecture they adopt.

This appendix section introduces the lifecycle of ML systems (Appendix A.1), then explains how ML systems can be split between entities fulfilling different roles (Appendix A.2) and introduces the methods designed to parallelize ML jobs (Appendix A.3) and the communication methods used between instances to communicate data and parameters (Appendix A.4). It is an extended version of Section 3 which focuses on the consequences on network traffic patterns and challenges of the elements presented here.

A.1. Machine learning model lifecycle

In machine learning, two approaches can be adopted to algorithmically derive insights from a dataset: supervised learning and unsupervised learning. In unsupervised learning, algorithms are developped and used to find patterns, clusters or relationships among data without previous knowledge or reference. In supervised learning, algorithms use a reference data set (or training set), consisting in data labelled with answers or hints to the solution of the question being asked, to build (or train) a model to which data is compared later during the inference (or serving) phase.

The model is the cornerstone of supervised machine learning. It is a representation of a system that is observed and measured with data. When the model is trained, the model is parametrized or modified so it is able to model the behaviour of the system from the dataset used in the training phase. Then, during the inference phase, data is presented to the trained model in order to derive information or make predictions about the system.

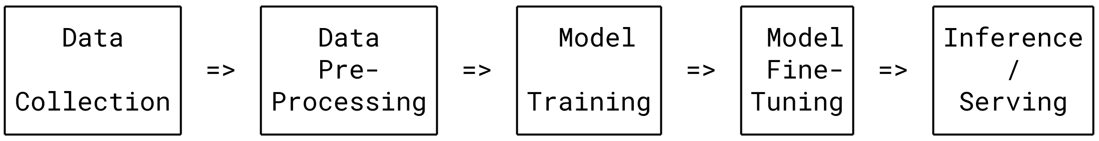

Figure 4: Supervised machine learning lifecycle

In the renewed interest in deep neural networks, and the strong development of generative AI, a supervised learning approach has been favoured, using reference models. Supervised machine learning follows a cycle presented in Figure 4.

To obtain a reference model, first, data is collected from a variety of sources. This data is then gathered and pre-processed, in order to clean it (align formatting, erase obvious measurement errors, *etc.*) and potentially to label it. In particular, large data sets are divided into chunks, or tokens, which are the basic data units used as input or outputs of models. For instance, if the data set consists in a corpus of texts, tokens are words, subwords or short sequences of words.

The pre-processed data is used to train the reference model, *i.e.* convert the knowledge to extract from the pre-processed data into a set of parameters including weights ruling how the model will answer future requests. Parameters are variables adjusting the structure underlying the model that are set during the training phase. Weights are a subste of the parameters, and determine how strong the ties are between some other parameters in the model.

Once the model has been trained, it can be used to infer from a request, or, in other words, to serve an insight from a request's data. Inference operations are not necessarily done by the same nodes that have trained the model. Models can be transferred to other worker nodes to perfom inference tasks, sometimes placed close to the users making requests. Besides, those transferred models can be re-trained or fine-tuned in order to better serve requests in the context they are done. This can be done for instance with private or locally relevant data.

A.2. System model

The machine learning systems built to meet the requirements of use cases presented in Section 2 are fundamentally distributed in order to meet scaling and time to answer requirements. Indeed, if we consider modern lareg language models, they use from 125 million to 175 billion parameters, where each parameters can be encoded with 4 bytes to 4 bits depending on the precision required in the answer provided by the model. From a memory perspective, this means that storing a model and its parameters will cost from 62,5 MB (for a model using 125 million parameters with 4-bits parameters) to 700 GB (for a model using 175 billion parameters with 4 bytes parameters). The memory footprint of modern large language models makes those models both difficult to manipulate on single nodes and to exchange data between nodes participating in a distributed task.

In machine learning systems' distribution, nodes can play specific roles, depending on their (hardware) capabilities or their location in the network.

Those roles, presented in Figure 5 are:

- **Data producer / Owner:**

  This entity is producing data from which users are willing to retrieve insights. It can be a sensor in a network, a server producing logs and interacting with a set of users, etc.

- **Data manager:**

  This entity is in charge of managing the data produced by the data producer. In that extend, it can pre-process, filter, add context (or metadata), store or transfer data to other entities in the system. In its data management operations, the data manager has to take specific care of security and privacy aspects of data management. In particular, it must make sure that data is stored securely, transferred to authorized entities and that user's consent and privacy preferences are respected. To do so, it may use specialized encryption methods and protocols, meeting challenges presented in Section 4.5.

- **Worker:**

This entity is in charge of processing data retrieved from the data manager in order to gain insights from it. To do so, it can training a (potentially large) model during the model training phase; and personalize this model to better serve specific needs or requests from users during the inference phase. In some specific cases, for instance, if the model from which the inference is done is small, the worker might be alone, but given the operational requirements of new approaches in machine learning, workers need to collaborate with other workers to complete a task together.

- **(Model) Aggregator:**

  In case the training, personalization or fine tuning of a model is done by several collaborating entities, the aggregator is responsible for gathering the results of the individual tasks performed by the workers and synthetize them in a general model. This model can be sent back to the workers for subsequent processing.

- **Inference engine / Server:**

  This entity is in charge of producing inferences, *i.e.* deriving insights from a request done against a trained model. Those requests might be contextualized from a conversation, *i.e.* a sequence of requests and their answers.

- **Coordinator / Orchestrator:**

  This entity is in charge of orchestrating the execution of distributed tasks among the workers in charge. This coordination role might be done by means of a centralized node (*e.g.* a parameter server), a federation of locally responsible nodes or in a coordinated fashion using a consensus protocol.

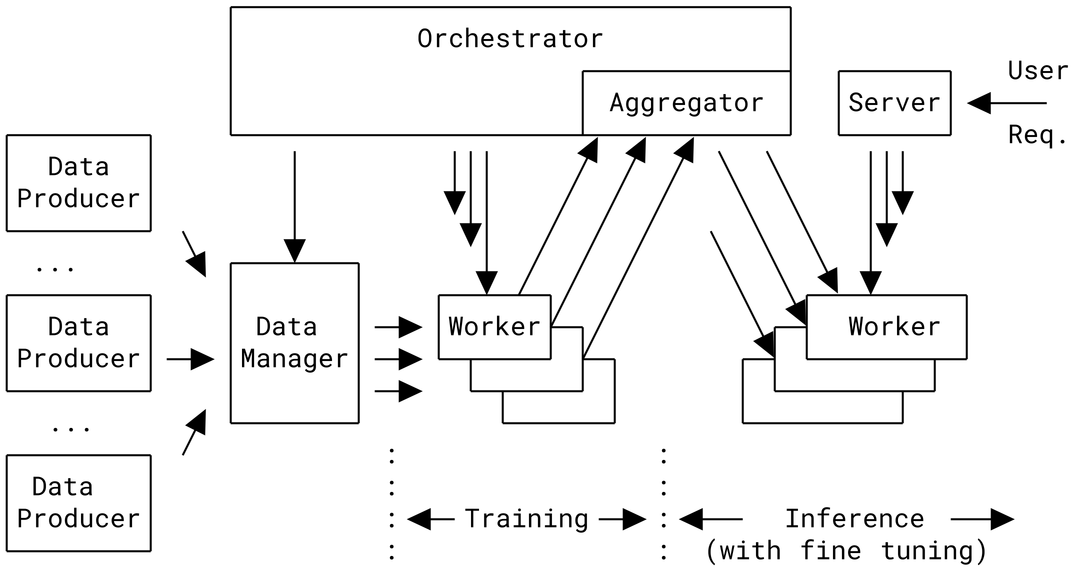

Figure 5: Machine Learning syste model

Those functional roles can be arranged, managed and executed in many different way in a ML system. This arrangement determines how centralized and / or distributed a ML system is.

A.3. Parallelization modes

In model-based machine learning, training and inference phases follow a pipeline similar to the pipeline presented in Figure 6. As mentionned in previous sections, given the size of currently used models and the amount of data required to either train or infer from a model, machine learning workloads need to be distributed. This distribution can follow different patterns (or a mix of those patterns): data parallelism, tensor parallelism, pipeline parallelism or mixture-of-expert parallelism. Those patterns are presented in the following subsections.

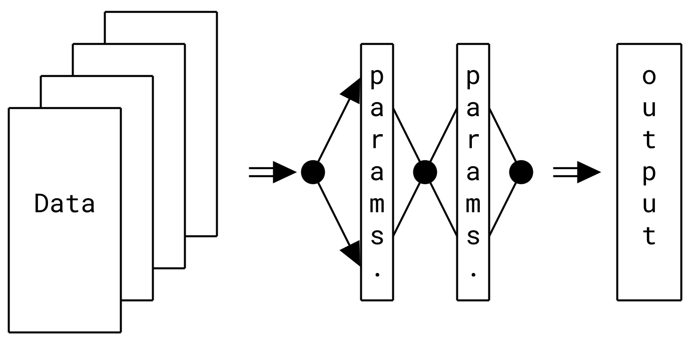

Figure 6: Model-based AI pipeline

A.3.1. Data parallelism

Data parallelism is the oldest among the parallelization patterns we highlighted. It has been introduced in [DataParallelism]. As presented in Figure 7, data parallelism consists in the partitioning of the data used in a given machine learning task in a set of batches that are used to train or tune the model. In data parallelism, a node manipulates a complete model and change its parameters using the data partition it has been allocated. The result of the tasks performed in parallel by the workers are aggregated by an aggregator node at the end of the pipeline. The model parameters resulting from this aggregation are transmitted back to the workers for future use of the model in order that each worker benefits from the work done by others in parallel.

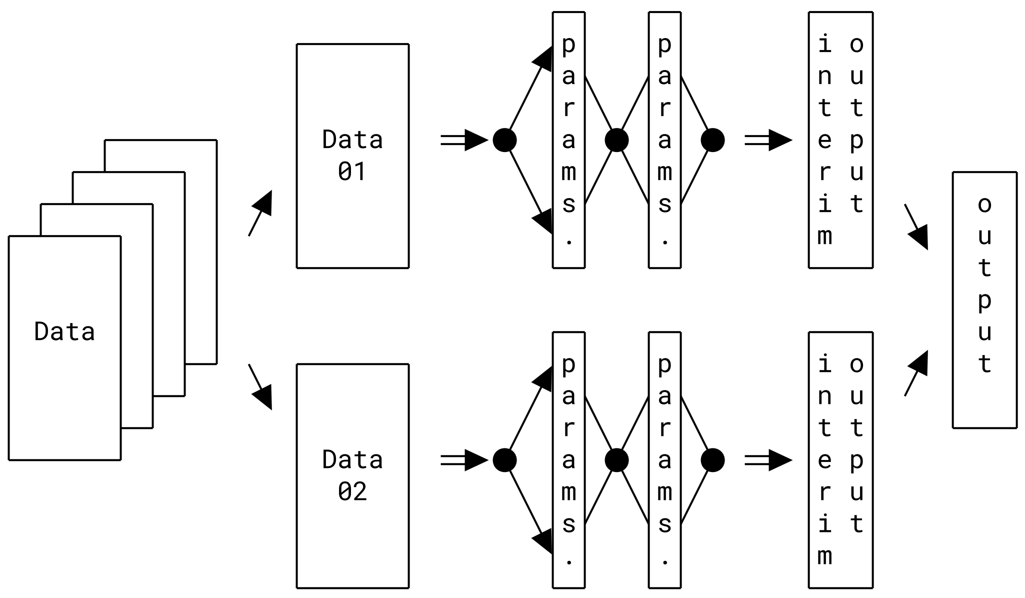

Figure 7: Data-parallel AI pipeline

In this pattern, the workers are only weakly synchronized, once the model has been aggregated and its parameters sent back. In that extend, data parallelism can sustain up to seconds latency in the synchronization traffic. Yet, as parameters are sent back by the aggregator to all the ndes for the entire model, the volume of data exchanged between training iterations of inference tasks can be quite large. Besides, the aggregator is a focal point in the traffic between nodes, which can raise traffic management challenges.

A.3.2. Model parallelism

For the last few years, models involved in machine learning tasks have increased in size, making their use by single nodes complex. To allow the training and inference of larger models, some parallelization patterns have been designed to spilt models among several worker nodes: **pipeline parallelism** and **tensor parallelism**.

A.3.2.1. Pipeline parallelism

Pipeline parallelism, described in [PipelineParallelism], takes advantage of the fact that models used in deep neural networks are structured in layers. In pipeline parallelism, as shown in Figure 8, a model is separated into stages consisting in a few consecutive layers of the entire model. Each stage is allocated to a separate worker node. Each stage is executed using the intermediate results from the previous stage, and after each iteration, the parameters from adjacent stages are used to refine the model's stage held by the worker node.

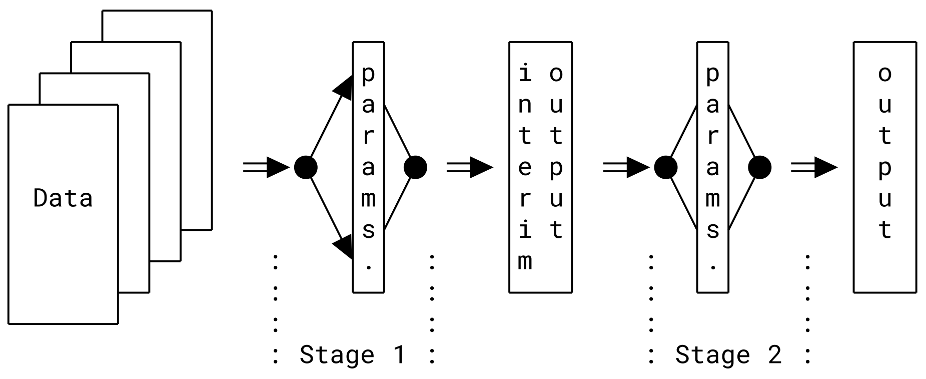

Figure 8: Pipeline model-parallel AI pipeline

In this pattern, the communication volume between node can be less important than in data parallellism, as each node only communicates with nodes holding adjacent layers. Yet, as stage iteration can be faster to execute, communications can be more frequent.

A.3.2.2. Tensor parallelism

Tensor parallelism, stemming from the work presented in [TensorParallelism], is taking advantage of the fact that training and inference operations in model-based machine learning are using operations on matrixes that can be split according to several dimensions. In comparison with pipeline parallellism, tensor parallelism is a pattern in which layers are split into chunks that can be operated by different nodes, as shown on Figure 9. Tensor paralellism can be naively presented as a split in the model parameter space rather than along the layer / stage dimension.

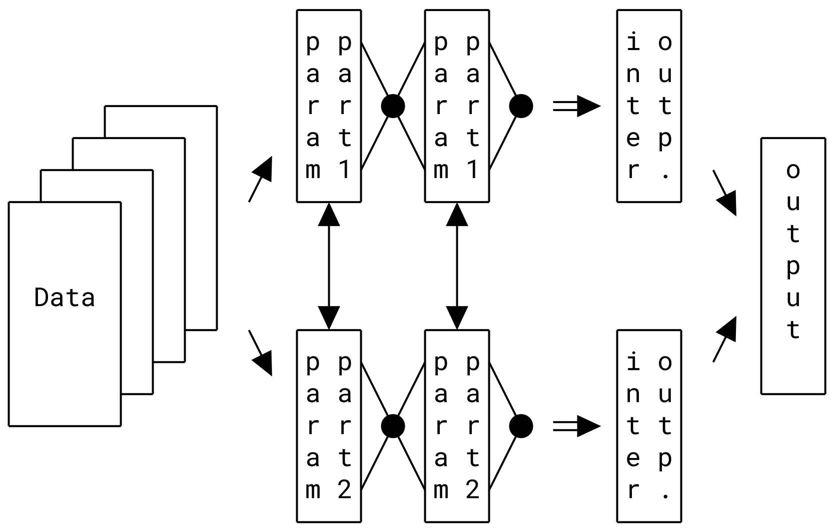

Figure 9: Tensor model-parallel AI pipeline

In tensor parallellism, nodes holding chunks of the same model (or of the same stage) need to communicate during the execution of a model (or stage) iteration. This involves even tighter latency requirements compared with pipeline parallelism for inter-node communications.

A.3.3. Mixture of Expert parallelism

Mixture of experts (MoE) parallelism stems from an idea introduced in [MoEParallelism1] and adapted to deep neural networks in [MoEParallelism2]. In MoE parallellism, nodes are holding smaller but specialized models trained over a smaller amount of data and holding fewer parameters. When a request or specific token is submitted to a MoE model, a gateway node, the router, takes a decision about which model instance to use against the specific token or request that has been submitted. After the selected worker executed the task against the token or request, the result is given, and considered as the result given by the whole model.

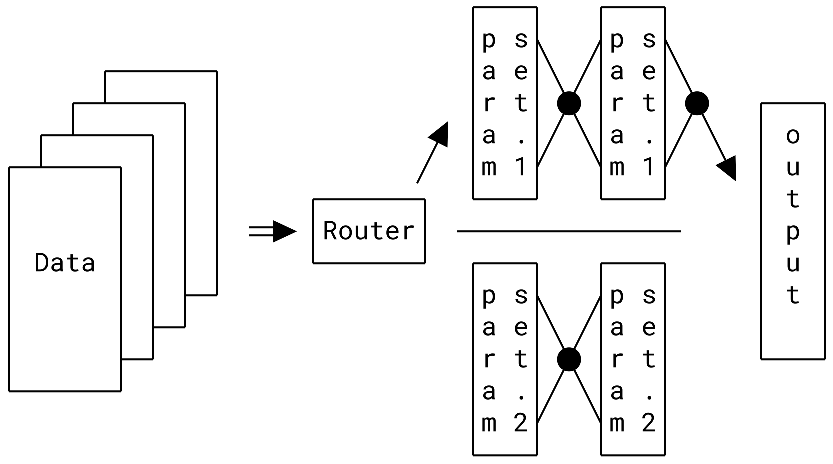

Figure 10: Mixture of Expert parallel AI pipeline

In MoE parallellism, the router plays a major role, and is a communication focal point.

A.4. Collective communication methods

In their development, distributed machine learning systems benefit from collective communication libraries such as NCCL ([NCCL]) or RCCL ([RCCL]) to abstract the setup and management of the connections used to exchange data between worker nodes involved in distributed training or inference tasks. As presented in [xCCL] or in [I-D.yao-tsvwg-cco-problem-statement-and-usecases], those libraries introduce collective communication methodss, used in accordance with parallelization modes presented in Appendix A.3.

To better explain what are the main collective communication methods, we consider a collective consisting in four nodes. Those nodes exchange pieces of data in the frame of the execution of a distributed machine learning task. The topology underlying the connections between those nodes is not detailed at this stage. The major collective communication methods are:

A.4.1. Broadcast

This collective communication method is very intuitive to understand for networking people, as its behaviour is the same as when a host broadcasts a packet in an IP-based network. In the broadcast collective communication, method, a node (N.1) is sending the same information to all the nodes participating in the collective. No data transformation is performed during this operation.

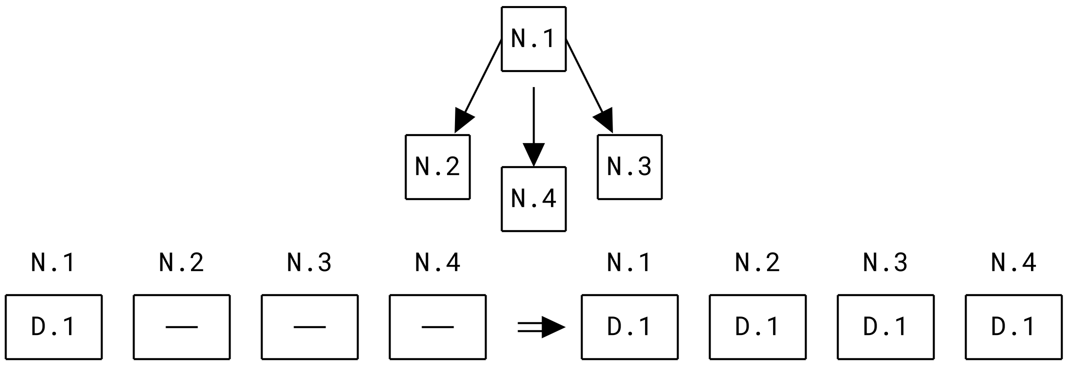

Figure 11: Broadcast method

A.4.2. Reduce

In this collective communication method, the data exchange is combined with an operation on the received data to produce an output. During a Reduce operation, the nodes involved in the collective send their data to an aggregator processing the received inputs *I.1* ... *I.4* using an operator *f* to obtain *Out*. *Out* is then provided to one of the nodes in the collective (N.1). Note that most of the time, the aggregation is done by one node in the collective, carrying the aggregator function.

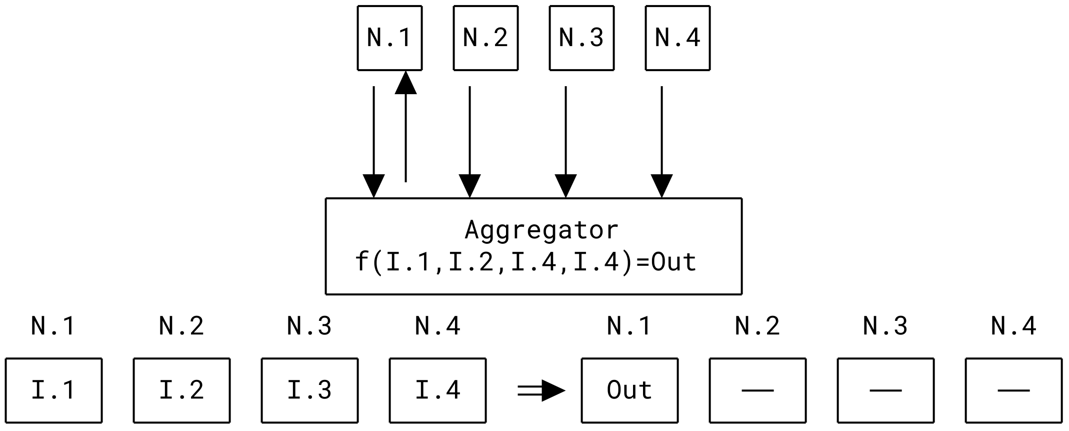

Figure 12: Reduce method

A.4.3. Scatter

In this collective communication method, a node in the collective splits the data it has into equal size chunks, and distribute a different chunk to every single other node in the collective, in order to distribute the data evenly.

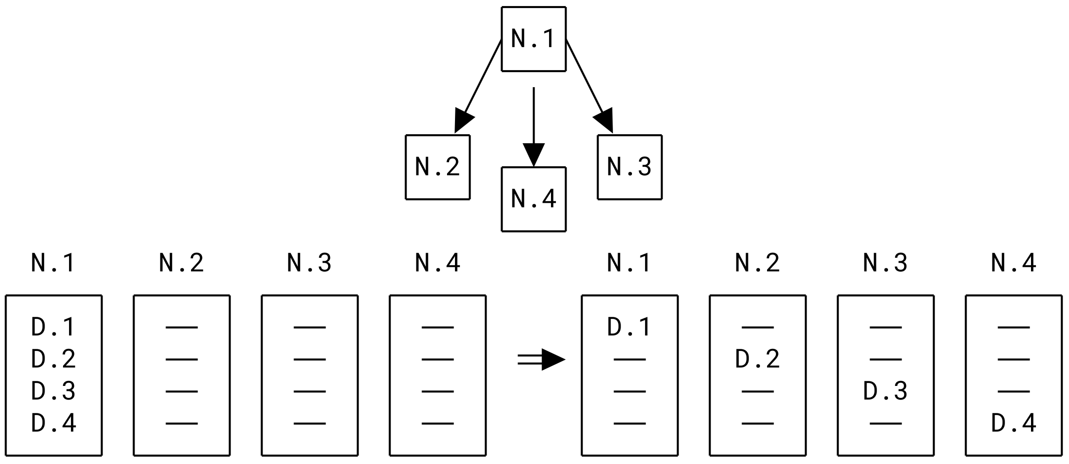

Figure 13: Scatter method

A.4.4. All-Gather

This collective communication method can be seen as a simultaneous broadcast done by every node in the collective. Indeed, in an All-gather operation, every node sends the data it has to every other nodes in the collective, so that in the end, every node has a copy of every piece of data held by the nodes in the collective before the operation.

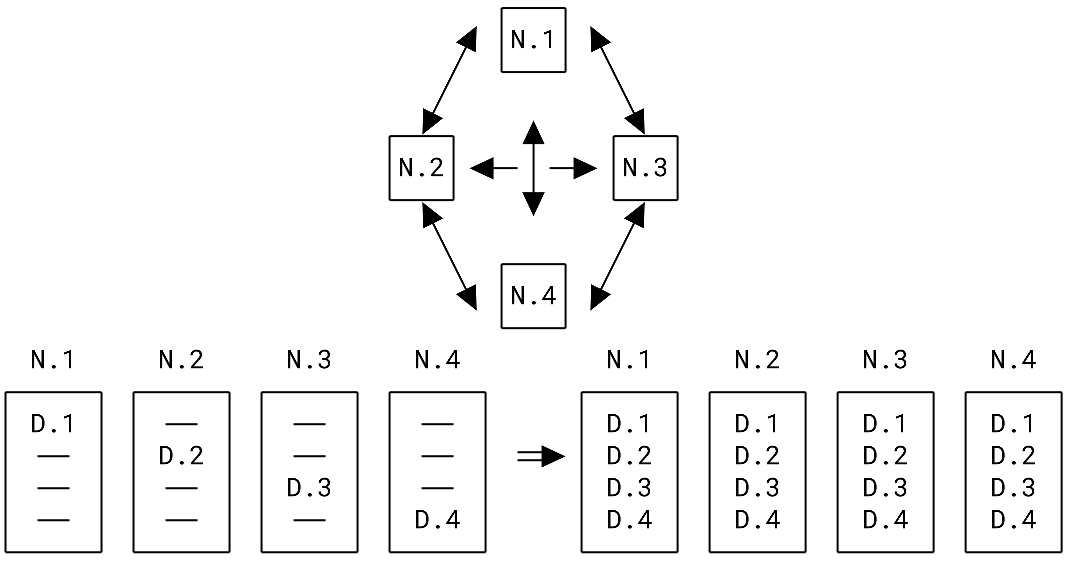

Figure 14: All-gather method

A.4.5. All-to-All

This collective communication method can be seen as a simultaneous scatter operation done by every node in the collective. Indeed, in an All-to-All operation, every node splits the data it holds in equal size chunks, and sends one chunk to every other node in the collective. As a result, the data held by each node after the All-to-All operation is different, but every node holds roughly the same amount of data, even if the data volume was not balanced before the operation.

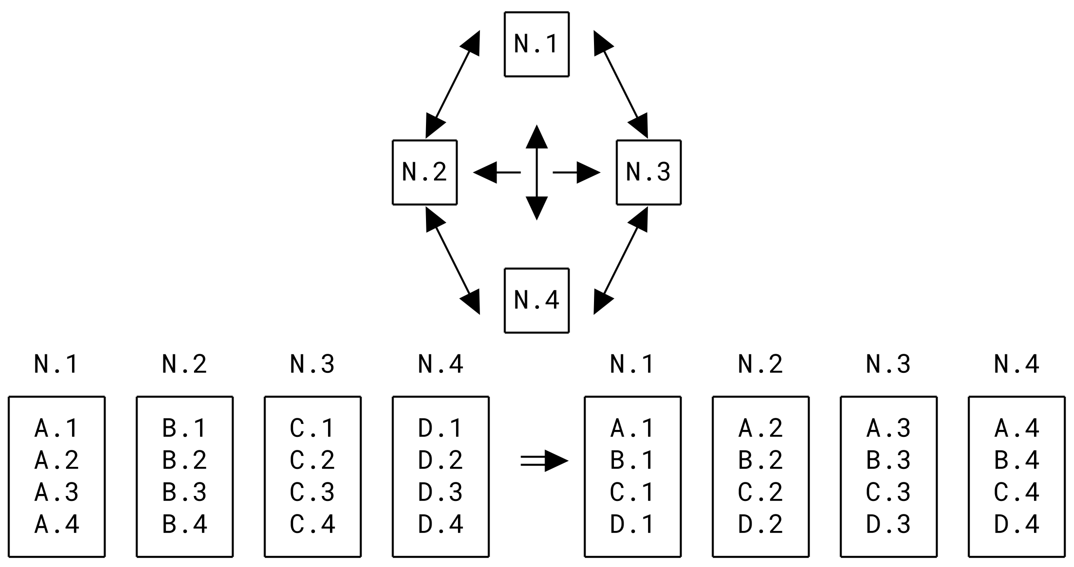

Figure 15: All-to-All method

A.4.6. All-Reduce

This collective communication method is very similar to the Reduce method. During an All-Reduce operation, the nodes involved in the collective send their data to an aggregator processing the received inputs *I.1 ... I.4* using an operator *f* to obtain *Out*. To the difference of the Reduce operation, *Out* is sent back to every node in the collective. In the implementation of this method, the aggregator function can be done by one node in the collective, or every single node applies the operator *f* on the inputs received after an All-to-All operation.

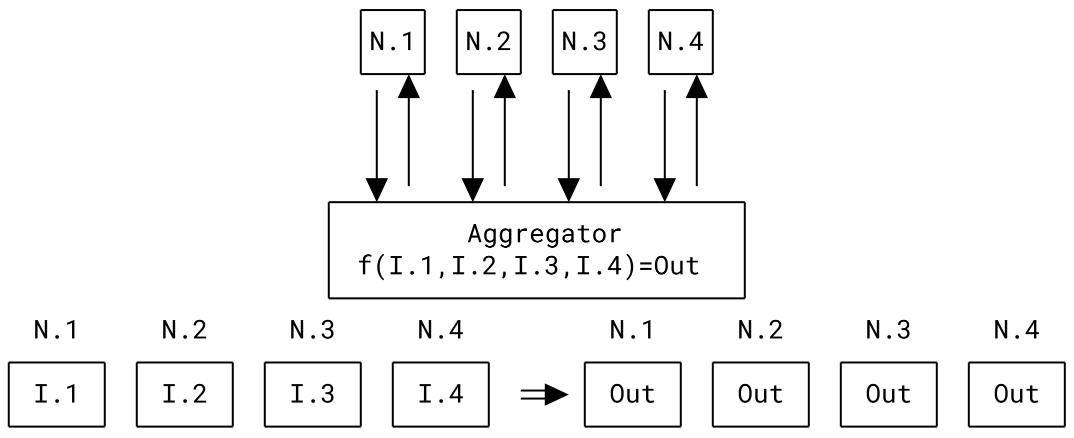

Figure 16: All-Reduce method

A.4.7. Reduce-Scatter

This collective communication method is a combination of the All-Reduce method with the Scatter method. During a Reduce-Scatter operation, the nodes involved in the collective send their data to an aggregator processing the received inputs *I.1* ... *I.4* using an operator *f* to obtain *Out*. To the difference of the All-Reduce operation, *Out* is split into equal size chunks *O.1* ... *O.4*, and each chunk is sent to a different node in the collective.

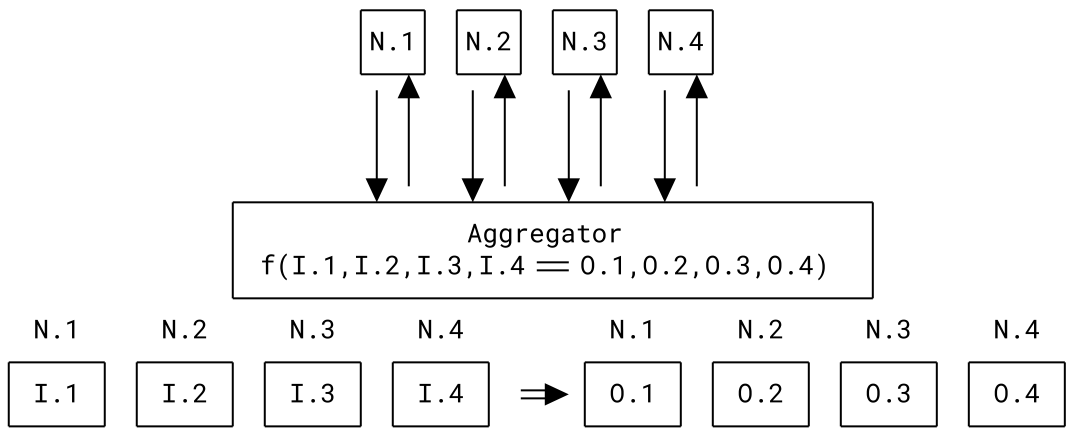

Figure 17: Reduce-Scatter method

Authors' Addresses

Antoine Fressancourt
Huawei Technologies France S.A.S.U.
18, Quai du Point du Jour
92100 Boulogne-Billancourt
France
Email: antoine.fressancourt@huawei.com

Luigi Iannone
Huawei Technologies France S.A.S.U.
18, Quai du Point du Jour
92100 Boulogne-Billancourt
France
Email: luigi.iannone@huawei.com

David Lou
Huawei Technologies Duesseldorf GmbH
Riesstrasse 25
80992 Munich
Germany
Email: zhe.lou@huawei.com

Dirk Trossen
Huawei Technologies Duesseldorf GmbH
Riesstrasse 25
80992 Munich
Germany
Email: dirk.trossen@huawei.com