RTG WG
Internet-Draft
Intended status: Informational
Expires: 2 September 2026

K. Kompella
P. Beeram
HPE
A. Mahale
Meta
R. Bhargava
Crusoe
N. Geyer
CoreWeave

1 March 2026

Scheduling Network Resources for Machine Learning Clusters
draft-kompella-rtgwg-mlnwsched-02

Abstract

Large Language Models (LLMs) are pushing the boundaries of technology. The scale that they have reached currently vastly exceeds the capacity of any single compute unit (XPU); this requires a distributed approach where multiple XPUs are connected via a "backend" network, sometimes in a single data center, but increasingly in multiple data centers connected by a "data center interconnect" (DCI). We are approaching the point where the scale exceeds that of a single data center, thus requiring multiple such data centers connected via a "data center interconnect" network. Training and inferencing are expensive and critical operations, thus they are typically scheduled, i.e., the (compute) resources they need are carefully estimated, allocated and deployed so that these resources are efficiently used. However, while compute investment in these LLM processing clusters dwarfs that of networks, it is becoming increasingly clear that the latter can greatly impact the former. This has been the focus of recent conferences, including the fantel Birds of a Feather meeting in IETF 123, @Scale: Networking 2025 and Open Compute Project 2025.

This memo proposes that the same care that is taken regarding allocation of compute resources to jobs be taken with networking resources: that they are estimated, allocated and deployed alongside compute resources; that they have contingency plans in case of network glitches; and that a holistic view be taken in order to optimize job completion times of training and inferencing jobs.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction

Large Language Models (LLMs) are pushing the industry to ever greater scale, both in training and in inference. This leads to more critical use of backend networks and a higher stake in producing timely results. A major learning from recent work is that the network cannot be taken for granted: a dropped or delayed packet can delay, stall or even abort a Machine Learning (ML) job, requiring more effort in checkpointing and managing job restarts, dealing with network congestion, and dealing with network failures. The problems get exacerbated in multi-tenant clusters where multiple jobs are run and job isolation becomes a key requirement. The fantel Birds of a Feather meeting (BoF) illustrated well the role the network plays in ML jobs, the potential for network events to disrupt jobs, and some early thoughts on how to handle these events. While the BoF was very successful in exposing these issues, we believe that adding a proactive approach would be beneficial; this can go hand in hand with the reactive approach of dealing effectively with network events.

This memo proposes that the network resources are reserved/scheduled in coordination with ML job scheduler, which is responsible for reserving compute resources (Central Processing Units [CPUs], Graphics Processing Units [GPUs], XPUs, memory, storage, ...). This is especially useful when multiple jobs are run in each cluster; an example is GPUaaS (GPU as a Service), or running several inference jobs simultaneously, or multi-tenancy. Reserving network resources reduces the probability of some disruptive network events and improves job isolation. This is the network analogy of reserving compute resources and ideally can be done at the same time. Essentially, when an ML job is scheduled, the "size" of the job (type of model, complexity of model, number of parameters, etc.) determines how many CPU/GPU/XPU cores are needed and how much memory and storage is needed; typically, the same parameters determine the amount of network resources needed during different collective (i.e., inter-XPU) communication stages (Broadcast, AllReduce, Reduce, etc.) Job placement (i.e., which XPUs to allocate for this job?) also determines the source(s) and destination(s) of the communication. If, at the time the job is scheduled, network resources are also reserved (and potentially, backup resources are put in place), the probability that network events can disrupt the job is reduced (although not eliminated). One can also set up the communication pathway and reserve resources when a collective communications API call ([MPI] or [NCCL] or the like) is made; this is especially relevant for long-running jobs where the time between communications phases can be long, and the phases vary from (say) Broadcast to AllReduce to quiescent. Finally, if backup pathways for a given communication are set up, traffic can quickly be protected when a failure happens, and in parallel, the sources can be notified of the failure and can reduce their traffic they send, build new end-to-end pathways or otherwise handle the failure.

The previous paragraph suggests a proactive methodology. Fast congestion notification and signaling constitutes a reactive methodology. These fit well together. One can couple network resource scheduling with fast event detection, signaling and mitigation for an overall much-reduced impact of network events on job progress.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.1.1. Definition of Commonly Used Terms

This section provides definitions for terms and abbreviations that are used in this memo.

XPU: one of several types of processing units: central processing unit (CPU), graphics processing unit (GPU), language processing unit (LPU), tensor processing unit (TPU) and the like. They fall under the category of "compute resources".

TE: traffic engineering, a technology that allows the specification of constraints (such as "admin groups" or colors) to guide the layout of

phop: previous hop (of N), a node and link that feeds in to junction N

nhop: next hop (of N): a node that is fed by N over a specified link.

MPTE: multipath TE, a technology that combines all the features of TE while offering multipathing with weighted load balancing for unicast traffic

MCTE: multicast TE, a technology that combines all the features of TE with load balancing for multicast traffic

ML: machine learning, a powerful technique to learn from data without explicit programming, used to solve problems of AI.

junction: a node in a DAG, with 0 or more phops, and 0 or more nhops. A junction with 0 phops is an ingress; a junction with 0 nhops is an egress. Other junctions are transit. A junction may be a unicast or a multicast junction. A DAG must have 1 or more ingresses, 1 or more egresses, and 0 or more transit junctions.

DSF: disaggregated scheduled fabric, a methodology for packet spraying in networks with multipathing.

DCI: data center interconnect

DAG: directed acyclic graph

2. Problem Statement

Consider the ML cluster Figure 1:

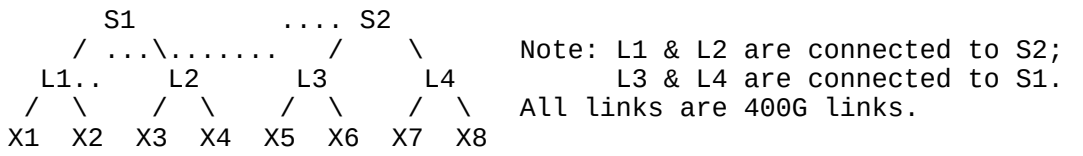

Figure 1: ML Cluster 1

The bottom layer consists of XPUs X1 through X8. The next layer up consists of "leaf" switches L1 through L4. The top layer consists of "spine" switches S1 and S2. All links between layers are 400Gbps; thus there is no oversubscription in the network, provided:

1. All XPUs are well-behaved.

2. All switches load balance fairly and perfectly.

However, "fair" load balancing is insufficient unless the load balancing is done on a per-packet (or better, per-cell) basis ("packet spraying") [DSF]. If load balancing is done on a per-flow basis ("flow level multipathing"), it is highly unlikely to be perfectly balanced across the next hops, in which case one next hop may see too much traffic, leading to congestion, packet delays or even packet drops. Disaggregated Scheduled Fabric (DSF) uses per-packet or per-cell load balancing, but it comes at a cost, and may not scale (and scale is a big consideration in these networks).

With flow level multipathing, say X1 and X2 are both sending 400G of traffic to L1. L1 tries to load balance X1’s traffic to S1 and S2 (in principle, 200G each). In practice, that may turn out to be 220G to S1 and 180G to S2. L1 does the same with X2’s traffic; let’s say this goes 190G to S1 and 210G to S2. The L1-S1 link will be congested, with 410G of traffic.

On the "downward" side (traffic going to the XPUs), there can be an "in-cast" problem: say both X1 and X3 are sending traffic to X6. In the worst case, each sends 400G for a total of 800G to X6, but the L3-X6 link can only transmit 400G. Thus, half the traffic will be dropped.

If the entire cluster (here, XPUs X1 through X8) is working on a single ML job, things are a bit simpler (but the issues remain). However, if this cluster is used for inferencing, or multi-tenant workloads, additional considerations arise. Tenant 1 (or inferencing job 1) (T1) may be using XPU X1 and part of X6; tenant 2 (or job 2) (T2) may be using XPU X3 and another part of X6.

If T1 and T2 simultaneously require communication to X6, there could be contention for the L3-X6 link. Again, this could lead to congestion, and hence delayed or dropped packets. But now, the issue is inter-tenant.

As stated in the Introduction Section 1, such delayed or dropped packets can have big consequences for the jobs that are running. Issues such as these are the motivation for DSF, packet spraying and fast congestion notification.

2.1. Collective Operation

Collective operations [CO] are used in distributed computing for the participating compute entities to exchange information. One example is the Message Passing Interface [MPI]; others are the NVIDIA Collection Communications Library [NCCL] and the ROCm Communication Collectives Library [RCCL]. These are used by the compute entities in a deep learning cluster to send information to each other, or as a group.

Collective operations include both unicast and multicast communications. Thus, in scheduling network resources, both patterns should be covered.

2.2. Compute Scheduling

In shared compute environments, such as a compute cluster or a cloud, a scheduler is commonly used to orchestrate access to compute resources. SLURM [SLURM] is a commonly used scheduler in Linux clusters; its documentation says "First, [SLURM] allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work." Another is KAI [KAI] which says "KAI Scheduler is a robust, efficient, and scalable Kubernetes scheduler that optimizes GPU resource allocation for AI and machine learning workloads." There are several other schedulers in common use.

A scheduler offers several features. The following are taken from SLURM:

1. Accounting

2. Advanced reservation

3. Gang scheduling (time sharing for parallel jobs)

4. Backfill scheduling

5. Topology optimized resource selection

6. Resource limits by user or bank account

7. Sophisticated multifactor job prioritization algorithms

KAI offers the following:

1. Batch Scheduling

2. Bin Packing & Spread Scheduling

3. Workload Priority

4. Hierarchical Queues

5. Resource distribution

6. Fairness Policies

7. Workload Consolidation

8. Elastic Workloads

9. Dynamic Resource Allocation (DRA)

10. GPU Sharing

To summarize, a compute scheduler allows effective and optimal sharing of compute resources among multiple tenants and multiple jobs, while ensuring fairness, enforcing limits and enabling accounting. Without a scheduler, multitenancy and multiple jobs would be impractical and chaotic.

Note that multi-tenancy is implicit. There may be ways to reserve resources for a particular tenant or group of tenants with allocating them, but the documentation doesn’t say how.

2.3. Network Scheduling

In shared network environments (which almost all networks are), a scheduler can be used to orchestrate access to network resources -- primarily bandwidth, but also highly prized links(*), QoS, etc.

The primary task of network resource scheduling is to reserve resource along a pathway (tunnel) from one or more XPUs (ingresses) to another set of XPUs (egresses). Note that the paradigm here is of uni-directional reservations; this is more general than bidirectional reservations, as the traffic requirements may not be symmetric.

Given that X1 wants to send 20Gbps to {X2, X3, X4}, one would create a tunnel from X1 to {X2, X3, X4} with 20Gbps capacity. Note that this traffic might be unicast (distributing different parts of a matrix to the recipients) or broadcast (distributing the same information to all). If further, one wanted to use certain links exclusively, one can color links in the network and state that this tunnel must/must not use links of a certain color. Thus, link coloring is a tool that network administrators can use to hold back links for a subset of job types. The compute analogy would be to hold back some XPUs, mark them "blue" and allow only a subset of jobs to use those XPUs.

Link coloring allows a provider to partition their network to optimally serve their customers. While links in a Clos network (as most ML clusters are) are perfectly symmetrical, once one gets into "distributed clusters" that are connected via DCI links, link coloring and other link attributes will find greater use.

Reserving bandwidth means that a particular job J1 (probably) won’t step on another job J2’s traffic. Say J1 is using a tunnel T1 with a reservation of 20G, and J2 is using a tunnel T2 with a reservation of 50G. The reservation procedure ensures any links T1 and T2 traverse in common have sufficient bandwidth for both T1 and T2 (and any other tunnels with reservations). Of course, J1 may use more than its allocated bandwidth; this can negatively impact J2. To reduce/ prevent this, one can apply a policer at the ingress of J1’s tunnels to ensure that J1 sends no more than its allocated share over each tunnel. This policer can drop traffic over the limit, or simply mark it as such, so that if the other jobs on a common link are not using their full quota, J1’s traffic can go through.

This last point is crucial for multi-tenancy. A provider who cannot provide hard (or at least soft) guarantees to their customers that they will in fact get the resources they asked (and paid) for will soon be out of business.

Elastic bandwidth is a very useful feature that goes along with elastic compute. If a job's requirements are: start me off with 5 XPUs, but expand that to 8 as the need arises, and shrink it back down to 5 when no longer needed, then the job's bandwidth requirements are likely to grow and shrink in tandem. Thus, in addition to making binding reservations, one must be able to adjust those reservations as needs change.

Finally, not all jobs (and all customers) are created equal. Priority and preemption are powerful tools in schedulers to give preference to certain jobs over others. Without these tools, a provider would be helpless if their cluster were overrun with low priority jobs. In addition, it would be nice to have a graceful way of managing preemption.

2.3.1. Traffic Engineering

All the features mentioned in the last section are available today, in bandwidth-aware traffic engineering (TE).

TE constraints allow a user to specify constraints on the path a tunnel will take. These can include administrative groups (colors), shared risk link groups (SRLGs), TE metric, other metrics such as delay, bandwidth reservations, and many others.

Bandwidth reservation allows the allocation of bandwidth resources to a tunnel. Policers are a useful adjunct to enforce limits.

Elastic bandwidth (aka "auto-bandwidth") allows a tunnel to dynamically adjust its reservations (within limits).

Priority and preemption are implemented by all vendors. Graceful preemption is possible using "soft preemption".

New traffic engineering parameters such as available buffer space, available queue-pairs for communication, etc. will be introduced and discussed in a future version of this memo, as well as in companion documents.

2.3.2. Multipathing

There is one missing piece with "regular" TE: ML clusters (and Clos networks or fat trees in general) make heavy use of multipathing, and often have multiple ingresses and egresses for their communications. Current traffic engineering techniques focus on a single path from one ingress to one egress. However, a new technique for multipath TE that allows for multiple ingresses and egresses and multiple paths between them is being developed that has relevance here

[I-D.kompella-teas-mpte].

2.4. Comparing Compute and Network Scheduling Features

In this section, we look at compute scheduling features, and ask whether the corresponding feature exists in network scheduling.

SLURM - Compute Scheduling Features	Network Scheduling (Feature Availability)
Accounting	Yes
Advanced reservation	Yes (bandwidth calendaring)
Gang scheduling	Yes (primary effort is on compute)
Backfill scheduling	N/A
Topology optimized resource selection	Yes
Resource limits by user or bank account	Yes (via controller policy) (enforcement via policers)
Sophisticated multifactor job prioritization algorithms	No (maybe N/A)

Table 1: Comparing SLURM and Network Scheduling

KAI features	Network Scheduling (Feature Availability)
Batch Scheduling	Yes (via multi-ingress/multi-egress tunnels)
Bin Packing & Spread Scheduling	Yes ("least-fill", "max-fill")
Workload Priority	Yes
Hierarchical Queues	Yes (via QoS in the data plane)
Resource distribution	Yes (via tunnel priority)
Fairness Policies	Yes
Workload Consolidation	N/A
Elastic Workloads	Yes ("auto-bandwidth")
Dynamic Resource Allocation (DRA)	N/A (multivendor is a given)
GPU Sharing	Yes (link sharing)

Table 2: Comparing KAI and Network Scheduling

As can be seen, almost all features are supported; some other features are supported in network scheduling that may not have analogies in compute scheduling.

2.5. Back to the Problem

Back to Figure 1.

With flow level multipathing, say X1 and X2 both send 400G of traffic to L1. L1 tries to load balance X1’s traffic to S1 and S2 (in principle, 200G each). In practice, that may turn out to be 220G to S1 and 180G to S2. However, L1 knows that it’s only supposed to send 200G to S1 from X1. S1 adjusts its load balancing weights ("adaptive load balancing") until the traffic sent to each of S1 and S2 is 200G. L1 does the same with X2’s traffic; if all works well, L1 will send a total of 400G to each of S1 and S2.

On the "downward" side (traffic going to the XPUs), there can be an "in-cast" problem: say both X1 and X3 are sending traffic to X6. Now, X1 has a TE tunnel to X6 with only 200G; similarly for X3. So, in principle, the L3-X6 link should only carry 400G.

Reservations can be temporarily exceeded; that is equally true with compute reservations. Depending on the enforcement policies, an oversubscription situation should be temporary and is clearly visible (since accounting is easy), allowing more severe enforcement should it be persistent.

3. Proposal

Multipath TE (MPTE) [I-D.kompella-teas-mpte] has all the features of Traffic Engineering, including the above-mentioned TE constraints. However, whereas "regular" TE [RFC2702] considers a TE path with one ingress, one egress and a single path between them, MPTE allows multiple ingresses and egresses, and considers all paths between ingresses and egressses that meet the TE constraints. Thus, MPTE build a directed acyclic graph (DAG) between ingresses and egresses. This allows traffic flowing over the MPTE DAG to be load balanced across these paths. Moreover, MPTE computes near optimal load balancing factors at each node; it does not simply use an equally weighted scheme.

This memo proposes the use of MPTE to compute, set up and allocate bandwidth for unicast collection communication among compute nodes in a deep learning cluster.

Multicast TE (MCTE) uses similar constructs as MPTE (namely, DAGs and junctions) to set up point-to-multipoint and multipoint-to-multipoint tunnels among compute nodes. MCTE also obeys TE constraints and allocates bandwidth resources. Thus, whatever the type of communication is required at various phases of a deep learning job, there is a TE construct to allocate network resources and instantiate the communication pattern.

Both MPTE and MCTE can preprogram "backup" paths in case of a link or node failure.

We believe the use of MPTE and MCTE will reduce the incidence of congestion in a deep learning cluster. Of course, congestion can happen for a number of reasons, including network failures. Thus congestion notification will be needed; however, with the state installed in the network for the TE tunnels, a node X that detects a (link or node) failure knows exactly what tunnels are affected by a given failure and which ingress nodes to notify. Furthermore, X can quickly put in place a backup path to protect against that failure until the ingresses can either reduce the traffic they send, or compute alternate end-to-end tunnels.

4. Conclusion

As mentioned in the Introduction, to make optimal use of deep learning clusters, especially when multiple jobs (e.g., inferencing or multi-tenancy) are run, and multi-tenancy is in play, network scheduling takes on increasing importance as a proactive measure to prevent network events such as congestion. (This works orthogonally to packet spraying.) One can add fast network event notification as a reactive measure. Together, these techniques present a more holistic approach and should allow much better utilization of ML resources.

5. IANA Considerations

None, for now.

6. Security Considerations

TBD

7. References

7.1. Normative References

[I-D.kompella-teas-mpte]
Kompella, K., Jalil, L., Khaddam, M., and A. Smith, "Multipath Traffic Engineering", Work in Progress, Internet-Draft, draft-kompella-teas-mpte-01, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-kompella-teas-mpte-01>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

7.2. Informative References

[CO] "Collective operation", November 2025, <https://en.wikipedia.org/wiki/Collective_operation>.

[DSF] "Disaggregated Scheduled Fabric", October 2024, <https://engineering.fb.com/2024/10/15/data-infrastructure/open-future-networking-hardware-ai-ocp-2024-meta>.

[KAI] "KAI Scheduler", n.d., <https://github.com/NVIDIA/KAI-Scheduler>.

[MPI] "MPI: A Message-Passing Interface Standard, version 5.0", 5 June 2025, <https://www.mpi-forum.org/docs/mpi-5.0/mpi50-report.pdf>.

[NCCL] "Collective Operations", 2020, <https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html>.

[RCCL] "ROCm Communication Collectives Library", 31 October 2025, <https://rocm.docs.amd.com/projects/rccl/en/latest/>.

[RFC2702] Awduche, D., Malcolm, J., Agogbua, J., O'Dell, M., and J. McManus, "Requirements for Traffic Engineering Over MPLS", RFC 2702, DOI 10.17487/RFC2702, September 1999, <https://www.rfc-editor.org/rfc/rfc2702>.

[SLURM] "SLURM Workload Manager", n.d., <https://slurm.schedmd.com/overview.html>.

Authors' Addresses

Kireeti Kompella
HPE
Sunnyvale, California 94089
United States of America
Email: kireeti.ietf@gmail.com

Vishnu Pavan Beeram
HPE
Sunnyvale, California 94089
United States of America
Email: vishnu-pavan-kumar.beeram@hpe.com

Aditya Mahale
Meta
Sunnyvale, California 94085
United States of America
Email: aditya.ietf@gmail.com

Raghav Bhargava
Crusoe
Sunnyvale, California 94085
United States of America
Email: rbhargava@crusoe.ai

Nikolas Geyer
CoreWeave
Canberra ACT
Australia
Email: ngeyer@coreweave.com