Nona: A Stochastic Congestion-Aware Job Scheduler for Real-Time Inference Queries

Benoit Pit-Claudel
MIT
Cambridge, USA

Derya Malak
EURECOM
Sophia Antipolis, France

Alejandro Cohen
Technion
Haifa, Israel

Muriel Medard
MIT
Cambridge, USA

Manya Ghobadi
MIT
Cambridge, USA

Abstract— **This paper proposes a novel queueing-theoretic approach to enable stochastic congestion-aware scheduling for distributed machine learning inference queries. Our proposed framework, called Nona, combines a stochastic scheduler with an offline optimization formulation rooted in queueing-theoretic principles to minimize the average completion time of heterogeneous inference queries. At its core, Nona incorporates the fundamental tradeoffs between compute and network resources to make efficient scheduling decisions. Nona's formulation uses the Pollaczek–Khinchine formula to estimate queueing latency and to predict system congestion. Builind upon conventional Jackson networks, it captures the dependency between the computation and communication operations of interfering jobs. From this formulation, we derive an optimization problem and use its results as inputs for the scheduler. We introduce a novel graph contraction procedure to enable cloud providers to solve Nona's optimization formulation in practical settings. We evaluate Nona with real-world machine learning models (AlexNet, ResNet, DenseNet, VGG, and GPT2) and demonstrate that Nona outperforms state-of-the-art schedulers by up to 350×.**

I. Introduction

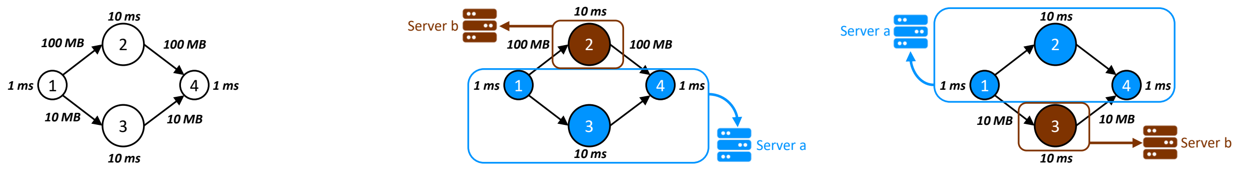
The compute and latency requirements of emerging online services, such as ChatGPT [1] inference requests, require dividing the workload across multiple datacenter servers [2]. As a result, job schedulers must efficiently distribute user-facing online Deep Neural Network (DNN) inference queries by meticulously considering computation resources, network capacity, and congestion while making fast real-time scheduling decisions.

To address this challenge practically, current job scheduling techniques rely on approximate representations of applications and deploy point solutions. They also depend on collecting real-time statistics about the datacenter, such as the status of currently running jobs or the instantaneous load of compute resources. However, *accurately measuring the queue occupancies of network switches in a cluster in real-time is nearly impossible* due to the bursty nature of datacenter traffic [3], [4]. Thus, the state-of-the-art job schedulers employ heuristic-based approaches to focus on the compute requirements of jobs, either ignoring the impact of network congestion completely [5]–[7], fitting a network cost model from past executions [8]–[11], or relying on the job to provide its networking demand [12].

On the other hand, queueing-theoretic approaches have the potential to capture network congestion, but extending today's solutions to structured DNN jobs breaks the independence assumptions about the arrival processes of data transfers in the network queues. Therefore, applying them to datacenter workloads presents two significant challenges. First, the Directed Acyclic Graph (DAG) structure of today's online queries induces several locality dependencies. DNN's DAGs are composed of tens to hundreds of compute operations (or *tasks*) that use the outputs of their parent tasks as inputs; distributing two consecutive tasks on different compute resources introduces communication costs and requires carefully considering the impact of data dependencies between the tasks on network resources and queues. Second, successive tasks in a DAG introduce stochastic dependency in the arrival processes of individual tasks: the arrival process of the last task of a DAG depends on the service distributions of all the other tasks. Queueing models usually assume Poisson arrivals of jobs automatically result in Poisson arrivals of every task in each job's DAG. Given the depth of DAGs of today's DNN inference models, this assumption is not accurate. Therefore, today's queueing-theoretic approaches cannot be used out of the box for jobs with deep, complex DAGs. Prior papers avoid these challenges either by considering individual, isolated jobs [13] or by excluding DAG parallelism [14]–[16] and ignoring network congestion. To highlight the prohibitive complexity of network-aware scheduling, the recent AlpaServe [17] paper proposed a simple queueing-theoretic model to capture the communication overhead of scheduling inference queries. However, due to the complexity of their model, they fell back to a heuristic-based solution [17, Section 4.2].

We propose Nona, a principled framework for distributing latency-sensitive inference jobs while considering *both compute and communication resources* of a cluster. Nona has three novel contributions. First, Nona uses an optimization formulation based on a flow network view [18, Chapter 3, 5.4] of the cluster that uses the well-established Pollaczek–Khinchine (PK) formula [19], [20] to predict network congestion and determine a series of stochastic decisions minimizing the average expected job completion time (JCT). Nona's optimization handles locality dependencies in the DAGs, building on prior work on networks of queues [21] to compute cluster settings.

Second, Nona's stochastic scheduler uses the solution found by the optimization to perform placement decisions in real time. This stochastic scheduler is lightweight, fast, and easy to implement (adding Nona's scheduler to the Apache Spark framework [22] requires $\approx$ 20 lines of Scala code): for each



(a) A Sample DAG (b) Allocation 1 (c) Allocation 2

Figure 1: (a) An example of a DAG with four tasks. (b, c) sample allocations. Tasks 2 and 3 are large and require 10 seconds to complete. Task 2 requires more data from the output of task 1 than task 3.

task in a job's DAG, Nona randomly samples a server from a pre-computed probability distribution obtained from solving the optimization.

Nona uses queueing theory to determine this series of stochastic decisions based on the steady-state properties of the workload. Consequently, Nona's optimization problem concentrates most of the complexity of our solution, leaving only lightweight operations to Nona's online scheduler. We argue that instead of developing complex online heuristic algorithms, latency-sensitive scheduling schemes should shift the complexity into an offline phase to enable making fast online decisions. This principle allows for solving the optimization and generating the corresponding probability distributions offline before running the system and adjusting them during operation without downtime.

Nona's optimization formulation leverages cloud providers' ability to continuously analyze traffic patterns *offline* and estimate each DAG's average arrival rate. In new deployments where the arrival rate is unknown or volatile, cloud providers may solve Nona's optimization with a set of predicted arrival rates and adjust the predictions over time. Our evaluation in section IV-D demonstrates that Nona is robust to changes in arrival rates of up to $\pm 40\%$. Moreover, cloud providers may precompute multiple scheduling strategies corresponding to multiple arrival rate scenarios so that Nona's scheduler uses a lookup table that best matches the current arrival rate conditions. Overall, Nona requires strictly less live state information than state-of-the-art heuristics [5]–[12], [17], [23].

Finally, Nona introduces a novel graph contraction procedure to reduce the search space and complexity of the optimization formulation without changing the optimal solution. This graph contraction procedure enables Cloud providers to execute Nona's offline optimization formulation within a few seconds.

To evaluate Nona, we implement our optimization problem and conduct extensive simulations comparing its performances to state-of-the-art schedulers such as Decima [7], and Spark [22], to an Expert scheduler, and to two custom lightweight congestion-oblivious schedulers (Random and Opportunistic). Using real-world DNN models (GPT-2 [24], DenseNet [25], VGG [26], AlexNet [27] and ResNet18 [28]), we demonstrate that Nona improves the average JCT by multiple orders of magnitude compared to Spark (56×), an Opportunistic scheme (180×), a Random scheme (202×), and Decima (350×), while being within 10% of an ideal Expert solution. Nona also improves 99%-*tile* tail JCT by 77×, 21×, 119×, and 81× respectively, compared to the same schemes. We then evaluate the impact of network bandwidth on Nona's performance and find that our results remain consistent for various network bandwidths. Finally, we study Nona's scalability and show that the execution time of solving our optimization problem is 30,000–245,000× faster than Decima's training time. Although both Decima and Nona have an offline phase, Nona is explainable, faster, and results in better performance.

II. Motivation

In this section, we explain how heuristic-based, congestion-oblivious schedulers can make poor scheduling decisions because of their inability to distinguish the network congestion implications of compute-equivalent distribution strategies. While we use a generic Congestion-Oblivious Schedulers in this section, our conclusions hold for other state-of-the-art scheduling systems, such as Decima [7], Optimus [8], Gandiva [9], Tiresias [10], Pollux [11], Themis [12], Spark [22], INFaas [29], AlpaServe [17] and DRM-DQL [30].

Sample job: Consider a job f summarized by the following operations:

$$f : (A, B) \mapsto \det(A + B) + \det(r(A + B)) \quad (1)$$

Where for any square matrix M, $r(M)$ is a sparse function setting all elements of M to zero except for a single random one on each row, and $\det(M)$ is the determinant of M. Figure 1a shows the DAG corresponding to this job with four tasks: (1) Summing A and B, (2) Taking the determinant of that sum, (3) Taking the determinant of the sparse version of the sum, and (4) Summing the outputs of (2) and (3). The top branch of the DAG has larger transfer sizes than the bottom branch (all of $A + B$ has to be transferred instead of just a sparse version), and the middle tasks require more compute time than the first and last tasks (determinants are more computationally expensive than sums). The duration of each task and the amount of data each task requires from the previous tasks are indicated next to the nodes and edges, respectively.

Single job scheduling: Consider a scenario with a cluster of two servers, a and b. We start by studying DAG distribution strategies for a single occurrence of f. Today's congestion-oblivious schedulers do not include any notion of topology or links between servers. When a job is distributed over multiple servers, it might pay a fixed penalty corresponding to various overheads, such as the startup time of a Java Virtual

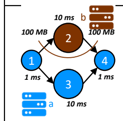
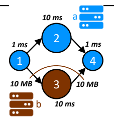
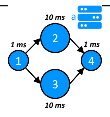
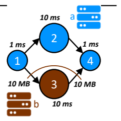
Cluster Load	Congestion-Oblivious Schedulers		Nona	
				
Low	50%	50%	20%	80%
Medium	50%	50%	50%	50%
High	50%	50%	80%	20%

Table I: Allocation statistics for different schedulers.

Machine (JVM) on that server. Moreover, congestion-oblivious schedulers tend not to employ cost functions that scale with the amount of data that needs to be transmitted between servers. Their outputs focus on the *number of servers* allocated to each task and the order in which different tasks should be executed. They are, therefore, unable to differentiate between two allocations with equivalent compute times but different network footprints.

As an example, Figures 1b and 1c show two possible task allocations by today's congestion-oblivious schedulers. Assuming servers a and b are equivalent, these two distribution strategies will have the same amount of compute on each server: place three tasks on one server (equivalent to $1 + 10 + 1 = 12$ ms compute time), and place the remaining task on the other server (10 ms compute time). However, allocation 1 in Figure 1b transfers 200 MB of data. On an 80 Gbps link, that represents 20 ms, or about 60% of the total JCT. In comparison, allocation 2 in Figure 1c requires transferring only 20 MB of data across the network corresponding to 2 ms on the same 80 Gbps link, or about 15% of the total JCT and $10\times$ less than allocation 1. As a result, even though the compute scheduling of both allocations is equivalent, their corresponding network traffic is vastly different. In practice, Today's congestion-oblivious schedulers end up randomly alternating between these strategies.

In contrast, Nona considers the size of data transfers on the DAG together with the available network bandwidth. As a result, Nona specifies which *groups* of tasks should run together on the same server instead of just the number of servers to allocate to each task. Consequently, Nona selects allocation 2 every time because it results in fewer bytes being transferred between the servers.

Scheduling streams of jobs: The previous example only considered a single job f. We now consider a more realistic case where a stream of users submits inference requests for f on different input data. State-of-the-art congestion-oblivious schedulers cannot distinguish between allocations 1 and 2. Thus, they end up load balancing between the two, as shown in Table I. The same strategy is applied regardless of the resulting load on the link and the cluster's total network load. Instead, Nona captures queueing delays in its optimization formulation and makes scheduling decisions based on the load on the cluster. For instance, Nona sometimes decides not to distribute the job if distributing results in worse performance. As shown in Table I, this decision depends on the expected average load of the cluster: when the network is lightly loaded, Nona chooses to distribute the computation and to use the network most of the time. When the network is busy, Nona mostly refrains from distributing f, to avoid adding to the network congestion and suffering from too much network delay.

This example highlights a significant potential source of gains over state-of-the-art congestion-oblivious schedulers. By incorporating network costs into the decision making process, Nona is able to reduce network congestion and JCT.

III. Nona's System Description

In this section, we first provide an overview of Nona's high-level design (§III-A) and relevant assumptions (§III-B). Then, we explain Nona's optimization formulation (§III-C). Finally, we describe two techniques to solve Nona's optimization for real-world jobs (§III-D).

A. High-level Design of Nona

Nona consists of two components: first, an offline optimization formulation that minimizes the JCT of a series of jobs distributed in a cluster with stochastic allocation strategies. Nona's formulation uses service provider's expected load of network and compute resources in the cluster to determine the expected queueing delays associated with different scheduling options. To do so, Nona uses the PK formula[1] [19], [20] to compute the expected JCT corresponding to each job's DAG. This allows the formulation to capture the tradeoff between gains stemming from running tasks in parallel and additional network delay due to ensuing network congestion. The solution resulting from the optimization is a probability associated to each scheduling decision.

The second component to Nona is its stochastic scheduler that uses the solution found by the optimization to perform placement decisions in real time. Consequently, Nona's offline optimization problem contains most of the complexity of our solution, leaving only lightweight operations to Nona's online scheduler. Tasks are then executed on their chosen server in the order defined by the DAG.

B. System Model and Relevant Assumptions

To build our optimization formulation, we consider a stream of jobs to be executed on a cluster. Servers and links in this cluster are represented as queues with deterministic service times. Each server has a compute rate which corresponds to the number of compute operations it performs per second. We assume each server works on a single compute task at any given instant, and has an infinite buffer to store pending tasks while it is busy. We divide the stream of jobs by class, where each class is defined by a DAG. In practice, for DNN inference jobs, each class represents a different DNN model.

We assume a large number of independent users submit inference requests to the cluster, thus following a Poisson arrival process [31, Proposition 11.2.VI]. Given that task

[1]The PK Formula is a queueing theory result giving the relationship between the queueing delay customers experience in an queue with Poisson arrival process, depending on the arrival and service rate distributions.

completion times are not exponentially distributed, and given that jobs in datacenter settings are not bound to a single server and can be split across different servers, we cannot directly use results from Jackson Networks [21]. However, we use a similar flow network [32] approach and take advantage of the PK Formula to estimate buffer delays.

Today's datacenters interconnect all servers through a hierarchical topology, such a Fat-Tree network [33]. Given that transport protocols adapt to the speed of the slowest link on the path regardless of the number of hops, we model the network topology with one bottleneck link. Our optimization problem can be extended to include network queueing delays on all links, but only yields marginal performance gains.

C. Optimization Problem

We start this section by outlining the high-level abstraction of our optimization problem, as follows:

Minimize:	Average job completion time (I)
subject to:	Communications, Computing, Flow,
	and Scheduling Constraints (II)-(XII) ,

(I) is the objective function to be minimized, (II), (III) and (VII) are the main constraints of this problem, and (IV)-(VI), (VIII)-(XII) define auxiliary variables, as we detail below[2].

Let $\mathcal{J}$ be the set of job classes, defined by their operation DAG $\mathcal{G}_j$ (tasks $\mathcal{T}^j$, data dependencies $\mathcal{D}^j$). For any DAG $j \in \mathcal{J}$, let β_j be the average arrival rate of jobs with DAG j, and τ^j be the average JCT of all requests corresponding to job j.

(I) We define our objective function to be $\sum_{j \in \mathcal{J}} \beta_j \tau^j$. This objective function captures the average completion time of jobs, weighted by the arrival rate of each job class.

Stochastic scheduling. A scheduling decision is a mapping of tasks onto servers, or equivalently a partition of the tasks in each DAG along with a mapping associating each part to a server. We frame our decision variables based on this graph partition view: we first divide the DAGs into subgraphs along bottleneck nodes (*i.e.*, nodes that are on every root-to-sink path) and then define probability distributions on partitions of these subgraphs.

More specifically, for a subgraph with source t_0, sink t_m, and intermediate tasks $T = \{t_1, \dots, t_{m-1}\}$, let π be a partition of T, and let π_k and π_l be parts of π. Let Ω_t be the set of all possible triplets (π, π_k, π_l) for the subgraph with root t. Ω_t represents the set of possible mappings for the source and the intermediate tasks of the subgraph. Let $\theta_t[(\pi, \pi_k, \pi_l)]$ be a probability distribution on Ω_t. $\theta_t[(\pi, \pi_k, \pi_l)]$ corresponds to the probability of choosing mapping (π, π_k, π_l) where the subgraph's source is placed on the same server as tasks in π_k, the intermediate tasks are grouped according to π, and the subgraph's sink is placed on the same server as tasks in π_l. To illustrate these notations, we show in Figure 2b a sample partition π, with three parts, π_0, π_1, and π_2. The contracted root task (see section III-D) corresponding to tasks $1+2+3$ is placed on the same server as tasks in π_2, *i.e.* tasks 12 to 15. Similarly, the sink task is placed on the same partition as tasks in π_1, *i.e.* tasks 7 and 11. This corresponds to choosing the mapping (π, π_2, π_1).

[2]More details on the formulation is available at https://nona.csail.mit.edu.

To completely characterize an allocation strategy, we must also determine the placement of the roots of the DAGs. We notice that to obtain a Pareto-efficient solution, the average load of every server should be equal. Therefore we set the assignment of the roots of the jobs to uniform distributions.

The assignment distributions $\theta_t[(\pi, \pi_k, \pi_l)]$ are our optimization variables. They are valid probability distributions, hence:

(II) $\sum_{\Omega_t} \theta_t[(\pi, \pi_k, \pi_l)] = 1$.
(III) $\theta_t[(\pi, \pi_k, \pi_l)] \geq 0$.

Communication cost. Let $b^{(t,t')}$ be the amount of information needed by each task t' from one of its predecessors t, and μ be the capacity of the bottleneck link. Let also $\mathcal{R}^j$ be the set of subgraph roots, $\mathscr{P}(t)$ and $\mathscr{S}(t)$ respectively be the set of predecessors and successors of t, and by extension $(t, t') \in \pi$ the event "t and t' belong to the same part of partition π". Each job arrival produces a set of communication arrivals, corresponding to edges in the DAG that are distributed over different servers. From job arrivals, we derive the average data transfer arrival and service rates (respectively $\lambda_{(t,t')}$ and $S_{(t,t')}$), and deduce a stability constraint on the link's load (ρ), the average queuing delay at the link (ϕ), and the average total communication delay ($c_{(t,t')}$):

(IV) Arrival Rates:

$$\lambda_{(t,t')} = \sum_{\Omega_t} \beta_j \theta_t[(\pi, \pi_k, \pi_l)] \cdot \begin{cases} \mathbb{1}(t' \notin \pi_k) & \text{if t}\in \mathcal{R}^j \\ \mathbb{1}(t \notin \pi_l) & \text{if t'}\in \mathcal{R}^j \\ \mathbb{1}((t,t') \notin \pi) & \text{o.w.} \end{cases} .$$

(V) Service time: $S_{(t,t')} = \frac{b^{(t,t')}}{\mu}$.

(VI) Link load: $\rho = \sum\limits_{j \in \mathcal{J}} \sum\limits_{t \in \mathcal{D}^j} \sum\limits_{t' \in \mathscr{S}(t)} \lambda_{(t,t')} S_{(t,t')}$.

(VII) Stability condition: $\rho < 1$.

(VIII) PK formula: $\phi = \frac{\sum\limits_{j \in \mathcal{J}} \sum\limits_{t \in \mathcal{D}^j} \sum\limits_{t' \in \mathscr{S}(t)} \lambda_{(t,t')} S^2_{(t,t')}}{2(1-\rho)}$.

(IX) Communication delay: $c_{(t,t')} = \phi + S_{(t,t')}$.

Cost propagation. Let p^t be the number of operations required by each task $t \in \mathcal{T}^j$ and ν the compute power of servers. Let $\mathcal{N}(t)$ the root node coming after t in $\mathcal{R}^j$, sorted in topological order. We independently compute the average completion times for each subgraph. First, we compute $\chi_{t,k,l}[(\pi, \pi_i)]$, the time required for all branches in part π_i of π to complete given that (π, π_k, π_l) was chosen, for every partition. Then we average those times to κ^t, the average time between the completion of the source and sink of a subgraph. Finally, we propagate the completion times of individual subgraphs to obtain the average JCT:

(X) Conditional branch completion time:

$$\chi_{t,k,l}[(\pi, \pi_i)] = \sum_{t_2 \in \pi_i} \sum_{t_1 \in \mathscr{P}(t_1)} \Bigg[\frac{p^{t'}}{\nu} + \mathbb{1}(t_2 \notin \pi_i) c_{(t_1,t_2)} + \mathbb{1}(t_1 = t \wedge i \neq k) c_{(t_1,t_2)}\Bigg] + \mathbb{1}(i \neq l) \sum_{t_1 \in \pi_i | (t_1, \mathcal{N}(t)) \in \mathcal{D}^j} c_{(t_1, \mathcal{N}(t))}.$$

(XI) Subgraph completion time:

$$\kappa^t = \sum_{(\pi,\pi_k)} \theta_t[(\pi, \pi_k, \pi_l)] \max_{\pi_i \in \pi} \{\chi_{t,k,l}[(\pi, \pi_i)]\} + \frac{p^{\mathcal{N}(t)}}{\nu}.$$

(XII) JCT: $\tau^j = \frac{p^{t_0}}{\nu} + \sum_{t \in \mathcal{R}^j} \kappa^t$.

Due to equations (IV, VIII, XI), the objective can at most be reduced to a geometric fractional function in simple cases, hence Nona's optimization formulation is not convex. We discuss practical considerations to approximate the optimal solution in the following section.

D. Practical Solving Considerations

We introduce two novel techniques to reduce the search space of the optimization formulation. Note that the last technique is already reflected in the formulation presented in section III-C through the choice of variables.

Technique 1: DAG Contraction. First, we notice that the size of the search space scales with the number of tasks in the DAGs. Depending on the DAG, some tasks need to be executed sequentially. Consequently, executing sequential tasks on different servers brings no benefit since they cannot be run in parallel. We use this observation to *contract* edges with sequential endpoints into a single node.

However, contraction needs to maintain correct task dependency. For instance, only isolated edges—subsets of the graph limited to a single input and output—may be contracted. Otherwise the resulting graph would not faithfully represent dependencies between tasks and would present unnecessary idle periods.

The following contraction procedure leverages this insight to keep only edges whose endpoint might gain from parallel execution: as long as the DAG can be further contracted, contract all edges $a \to b$ where all input nodes of b are also input nodes of a, and all output nodes of a are also output nodes of b. More rigorously, given $\mathcal{G}(\mathcal{T}, \mathcal{D})$ a graph to be contracted, with nodes $\mathcal{T}$ and edges $\mathcal{D}$, the set of edges to be contracted in a contraction step is given by:

$$\mathcal{C} = \Big\{(a, b) \in \mathcal{D} \mid \forall n \neq a, (n, b) \in \mathcal{D} \implies (n, a) \in \mathcal{D}, \forall n \neq b, (a, n) \in \mathcal{D} \implies (b, n) \in \mathcal{D}\Big\}$$

Figure 2a illustrates this contraction procedure. Edges to be contracted in the next step are highlighted in green. Some edges seem contractable but are not; we highlight them in yellow and show the edges preventing them from being contracted in red. For example, contracting the edge between nodes 4 and 5 would lead to node 8 waiting for node 5 to complete before starting, a dependency not present in the initial graph. As shown by the bottom branches (nodes 12 to 15), multiple rounds can be necessary to ensure the graph is fully contracted.

Technique 2: Relative Assignments. The second technique we use takes advantage of the relative assignments between tasks and servers. In particular, we argue that the critical question the optimization problem should answer is not which specific task should be placed on which specific server. Rather, the assignment decision can be reduced to a set of *relative* decisions: which tasks of a job should be grouped together on the same server.

Therefore, Nona's optimization uses $\theta_t[(\pi, \pi_k, \pi_l)]$ to group tasks together on an arbitrary server, instead of looking at which specific server should host a given task. Note that relative assignment makes the optimization formulation independent of the number of servers, thereby enabling Nona to scale to large clusters. Our evaluations demonstrate this highly desirable feature in Section IV.

IV. Evaluations

A. Methodology

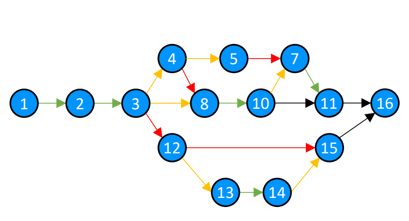
To evaluate Nona, we augment the event-based simulator in Decima [7] to become network-aware. The simulator takes the following input parameters: (1) a series of DAGs, each corresponding to a class of jobs, (2) an average arrival rate for each job, and (3) the cluster properties such as the number of servers, network capacity, and compute capacity per server. Upon every task arrival or departure, the simulator places the subsequent available tasks on servers according to scheduling policy of the evaluated scheme. Finally, the simulator reports statistics about job and task completion times.

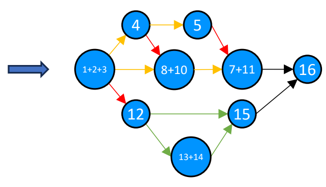
Workloads. We consider five classes of inference jobs constructed from GPT2 [24], Densenet121 [25], VGG16 [26], AlexNet [27], and ResNet18 [28], for which we retrieve jobs' DAGs using PipeDream's [34] profiling tool. We then contract each graph as described in section III-D. We report model characteristics in Table II, including the number of tasks in each DAG before and after contraction. Our contraction procedure reduces the number of tasks by up to a factor 107, reducing the search space by similar factors. Some models, such as Alexnet and VGG are reduced to a single node by our contraction procedure. This indicates that their operations graphs are sequential with no model parallelism opportunities.

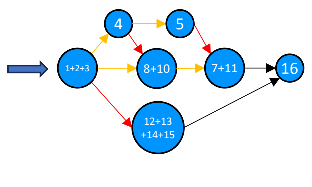
Model	Size (MB)	Layers	# of tasks in the DAG	
			Pre-Contraction	Post-Contraction
GPT2	487	164	178	**40**
Densenet	31	369	429	**4**
VGG16	528	40	41	**1**
Alexnet	233	22	23	**1**
Resnet18	45	61	71	**10**

Table II: Properties of DNNs used in evaluating Nona.

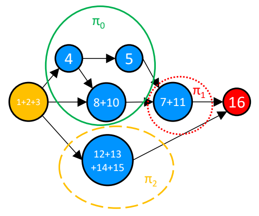
To simulate a realistic multi-tenant cluster, we generate a series of background compute and communication jobs. Each background compute job consists of a single task in





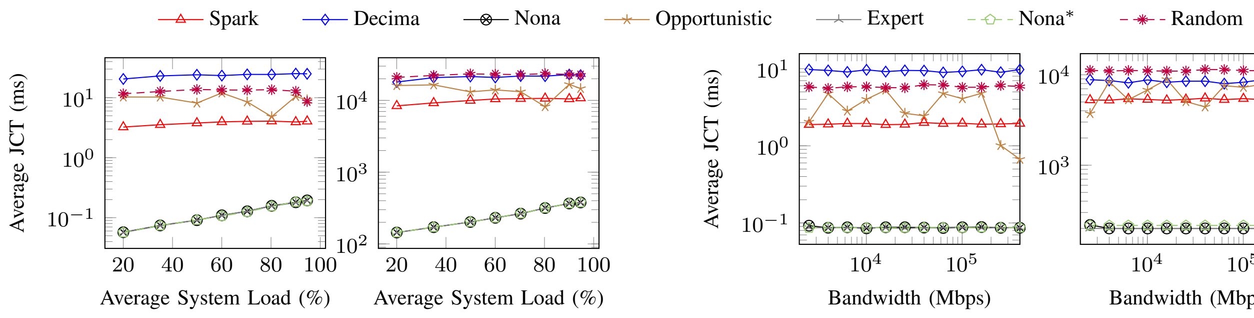


(a) Contraction procedure.



(b) Graph partitioning.

Figure 2: (a) Contraction procedure on a sample graph. (b) Summary of graph partition notations.



(a) All jobs (b) Inference jobs

Figure 3: Average JCT depending on system load.

(a) All jobs (b) Inference jobs

Figure 4: Average JCT depending on link bandwidth.

its DAG, with zero communication demand. Each background communication job has zero compute demand. Empirically, we choose the ratio of inference-to-background arrival rates to be 100:1.

Compared Schemes. We simulate the following schemes:

Decima [7]: Decima is a reinforcement-learning scheduler to make scheduling decisions based on instantaneous information about the state of the cluster. The model is congestion-oblivious: its training environment does not consider network delays. We use a DNN model trained on SQL queries provided by the authors.

Spark's Fair Scheduler: this scheme shares the compute resources of the cluster fairly between all active jobs. It also requires instantaneous information about the state of the cluster. By default, Spark's Fair Scheduler prioritizes large jobs and starves smaller jobs by blocking all of the servers allocated to a job until it completes. We modify it such that tasks release their resources when they are done.

Random: Tasks allocations are chosen randomly. While this scheme has almost no overhead, it is both compute-oblivious and congestion-oblivious.

Nona: We run Nona's optimization offline separately from its simulator. The performances of our implementation of Nona's optimization are discussed in section IV-F. The output of the optimization is saved to a file. The simulator loads Nona's probability distributions from the file at startup, and uses them to make congestion-aware scheduling decisions.

Opportunistic: Whenever the contracted graph presents parallel branches, distribute all these branches on different servers. The specific servers are chosen randomly such that one of the branches is placed on the same servers as the root of the subgraph, making this scheme also congestion-oblivious.

*Nona**: Nona's optimization problem takes the average arrival rate of jobs as input. In some cases where that information is not available initially, or if the arrival patterns change during operations, Nona would run with a non-optimal assignment strategy. To test Nona's robustness to varying system conditions, we run simulations using a single set of allocation probabilities, obtained from solving the optimization a single time, and use this same probability distribution for all the loads (or arrival rates).

Expert: this scheme uses a manually derived probability distribution. For every data point, we reason about the job's DAG and the system's expected load and determine the optimal distribution. This process is tedious and does not scale in the size of the cluster, the number of DAGs, or the complexity of the DAGs.

Among these schemes, the Expert, Opportunistic, Random, as well as Nona and Nona* do not use the current state of the cluster queues when making scheduling decisions. Decima and Spark's Fair Scheduler use live state information about the state of the cluster and the completion progress of jobs to make decisions.

B. Nona's Overall Gains

Figure 3 compares different scheduling techniques' average JCT on a cluster with 80 servers and 10 Gbps link bandwidth. We vary the interarrival rates λ_j of jobs j and derive the corresponding compute load as $\sum_j \lambda_j S_j$ where S_j is the sum of the service times of all tasks of j.

As shown in Figure 3a, when considering all jobs in the cluster, Nona performs similarly to the Expert case, and

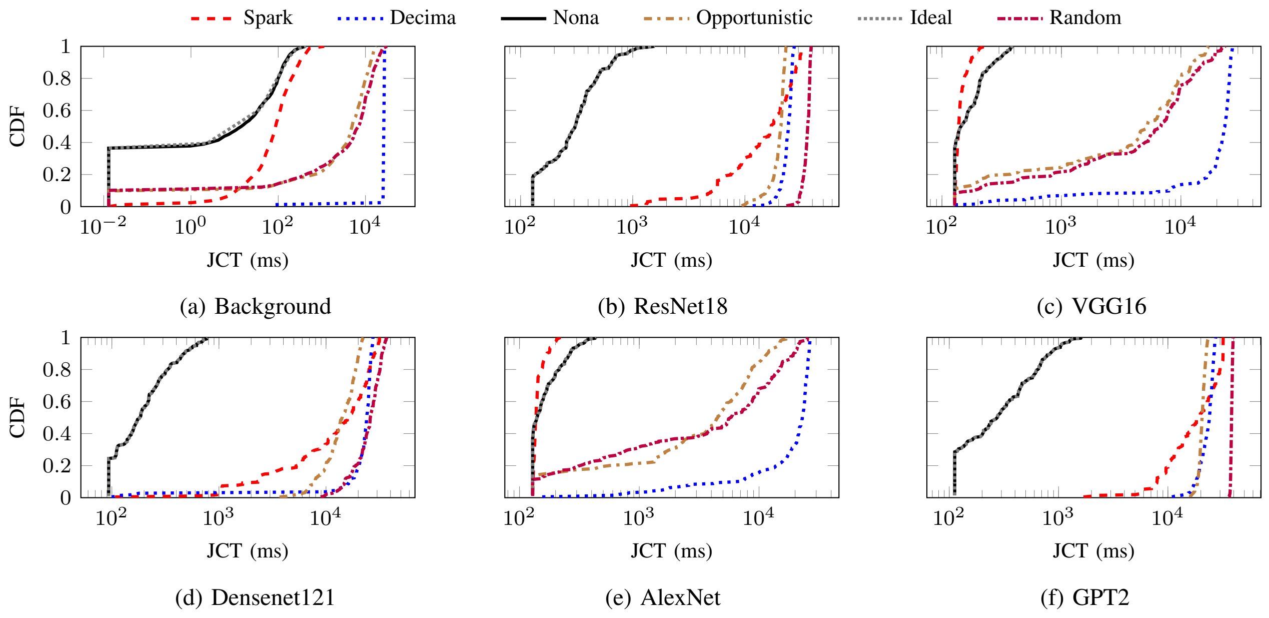


Figure 5: CDF of the JCT for each job, on a 80-server cluster, with 70% communication and compute load.

outperforms Spark by a factor of 32 to 56×, the Opportunistic scheme by a factor 30 to 180×, the Random scheme by a factor 70 to 202× and Decima by a factor of 139 to 350× on average JCT. As the system load increases, the likelihood of having more active jobs than servers increases. Spark attempts to fairly allocate as many servers to every job in the system. This means giving each job a single server. Thus, the average JCT for Spark plateaus as the scheduler stops performing any parallelism to maintain fairness. Nona still achieves lower JCT both average and 99%-tile (by a factor 77×), since even at high load, some amount of parallelism can be beneficial when low amounts of data are transmitted on the parallel branches. The JCT for the Opportunistic and Random schemes is dominated by network congestion even at low load, and therefore their performance is not affected significantly by load variations when compared with Nona's performance.

Similarly, Figure 3b shows that when the average is taken only over inference jobs, Nona also yields average JCTs similar to the Expert case and also outperforms Spark, the Opportunistic and Random schemes, and Decima by respective factors of 28 to 56×, 25 to 111×, 59 to 145, and 59 to 125×, respectively. Decima performs better when the average JCT is taken over inference jobs only since it prioritizes completing existing jobs over running the Shortest Remaining Tasks First (SRTF), and therefore starves the small background jobs.

The main reason for these improvements is network congestion: while Nona is offline and does not have an instantaneous view of the network, neither of Spark, Decima, or the Opportunistic or Random schemes take the network into account when making scheduling decisions. Therefore, all these schemes overload the network link by eagerly distributing jobs as much as possible, while Nona successfully prevents the system from jamming.

C. *Impact of the Network Capacity*

To isolate the impact of network resources on average JCT, we fix the arrival rate and vary both the capacity of the network and the network demand of each job, while keeping the compute load constant. In Figure 4, Nona still performs as well as the Expert solution, and yields consistent average JCT gains over a wide range of bandwidths. More specifically, Nona outperforms Spark's Fair Scheduler, the Opportunistic and Random schemes, and Decima, respectively by up to 22× (27×), 60× (44×), 70× (57×), 111× (44×) when the average is taken over all jobs (over only inference jobs). Here again, Decima suffers particularly from starving background jobs.

D. *Nona's Robustness to Arrival Rate Uncertainty*

To show the robustness of Nona's optimization to uncertain arrival rates, we run the optimization a single time, for expected loads and bandwidths of 50% and 10Gbps respectively. Then, we run the same experiments as in sections IV-B and IV-C for Nona using only this single solution. The resulting JCT, labeled Nona* in Figures 3 and 4, show Nona does not require a precise knowledge of the effective arrival rates. Nona*'s performance is within 15% of Expert and Nona, and therefore outperforms all the other schemes by factors similar to Nona's.

E. *Impact of the Job Structure*

Figure 5 breaks down Figure 3 and shows the Cumulative Distribution Function (CDF) of JCTs for each job class, at a load of 70%. For inference jobs, Nona performs similarly to the Expert allocation and outperforms every other scheduling approach when the DAG presents parallelism options (Figures 5b,d,f), and outperforms all other schemes except for Spark for jobs with a single task (Figures 5c,e). Nona gives slight priority to background jobs given that its objective function is an average over the JCT of all jobs, including background. Indeed, while we modified Spark to prevent head-of-line blocking for these small background jobs, Figure 5a shows that Nona serves almost 40% of background jobs with no queueing delay, compared to 1% for Spark. Overall, for inference jobs, Nona accelerates the 99%-tile tail JCTs over Spark, the Opportunistic and the Random schemes, and Decima by $57\times$, $15\times$, $89\times$ and $58\times$ respectively. Figure 5a confirms Decima suffers from starving the short and frequent background jobs, with 99%-tile tail JCT $343\times$ higher than Nona. A similar behavior is observed for single-task inference jobs as demonstrated in Figures 5c and e: Nona achieves a 99%-tile tail JCT $343\times$ lower than Decima.

F. Nona's Optimization Implementation

Some linearization techniques have been proposed to solve problems with geometric fractional objective functions [35], [36] like Nona's. However, these methods consider problems with no equality constraints, as opposed to Nona (equation II). Therefore, we implement the objective function of the Lagrangian relaxation of Nona's problem, and attempt to minimize it. The Lagrangian parameters are set empirically to multiple orders of magnitude above the expected JCTs, to sufficiently penalize points outside of the feasibility region. After contracting jobs, the size of the search space for the five jobs chosen is of 46, spread across 9 probability distributions; in general, the problem scales linearly with $\sum_j |\mathcal{R}^j|$ and is independent of the number of servers in the cluster.

Table III reports the runtime of our single-threaded implementation of Nona's optimization's solver, running on one AMD EPYC 7502P CPU, and compare it for various cluster sizes to the time required to train Decima's RL model on an Nvidia A100 using the same input parameters. We scale the training set proportionally to the size of the cluster to ensure a constant average number of jobs per server, explaining the linear scaling in training time. Nona, on the other hand, uses a queueing theoretic model independent of the cluster size to capture the essence of the problem's properties, and solves its optimization formulation $30{,}000\times$–$245{,}000\times$ faster[3].

Scheme	Number of servers							
	16	32	48	64	80	96	112	128
Decima	108h	215h	321h	438h	555h	651h	765h	889h
Nona	13s	13s	13s	13s	13s	13s	13s	13s

Table III: Training/optimization time comparison.

V. Related Work

This section categorizes prior approaches into five classes of techniques: (i) Inference Schedulers, (ii) Heuristic-Based Schedulers, (iii) DNN Schedulers, (iv) Queueing-Theoric Techniques, and (v) Optimal Resource Allocation.

[3]Our simple approach for solving Nona's optimization problem is fully parallelizable, and could be made multi-threaded. Smarter approaches to solving the optimization problem could further reduce the time required by the optimization. We leave these improvements to future work.

Inference Specific Schedulers: Datacenter schedulers for inference tasks have gathered interest from the community in the past couple of years. However, as shown in the survey from Ye et al. [37], the focus of previous approches has been on clever Machine Learning or GPU architecture based approaches to make the compute operations themselves more efficient. For example, in [38], the authors explored the benefits of caching data in GPU memory; in [23], the authors expanded on this idea by arguing for a system that only reloads the difference between two variants of the same model. INFaas [29] proposed a system to automatically choose model varients depending on current cluster state. Finally, AlpaServe [17] demonstrated benefits of model parallelism for statistical multiplexing of compute tasks in jobs. Our approach is based on the insight that the network must be taken into account for latency-sensitive online queries. Some of the techniques exposed in these papers (like caching) are compatible with Nona but would require some modifications of our optimization formulation; we leave this as future work.

Heuristic-Based Schedulers: Heuristic-based schedulers attempt to distribute tasks on a cluster by observing its current status and using a heuristic to make fast placement decisions. For instance, Gandiva [9], Tiresias [10], Themis [10], Pollux [11], and Pipedream [34], proposed several algorithms to distribute machine learning training jobs while optimizing compute utilization, compute throughput, or fairness metrics. However, these approaches did not consider *latency-sensitive* user-facing inference jobs. Sparrow [39] proposed an online randomized sampling mechanism to determine the status of compute queues before allocating tasks to servers. Yet, Sparrow's approach is not extendable to network queues because of the bursty nature of network flows [3]. In addition, the key difference between online schedulers and our approach is that online schedulers rely on the short-term behavior of queues while our queueing theory-based approach draws on the long-term stochastic system characteristics.

DNN Schedulers: Recent DNN models have become sufficiently large to warrant dedicating entire clusters or isolated subsets of clusters to single jobs [40]. DNN-specific approaches thus work offline to derive an optimal allocation of resources specific to one cluster and one job. In [13], the authors derived an optimal DNN job DAG split minimizing the training time. In [6], the authors explored parallelization dimensions beyond model and data parallelism and find strategies combining them. PipeDream [34] and Gpipe [41] improved pipelining for DAG parallel training of a single job. Gpipe assumed that the entire cluster is available for a single job; consequently, Gpipe is not tailored to consider interactions between different jobs. Therefore, extending these solutions to handle *streams* of inference jobs is non-trivial: it requires considering much larger search spaces that consider *multiple DAGs* and their interactions.

Queueing-Theoretic Techniques: Unlike heuristic-based schedulers, queueing-theoretic scheduling techniques leverage the steady-state properties of the cluster to distribute a series of tasks on a network of queues [15]. For instance, Jackson networks [21] consider networks of queues with Poisson arrival processes and exponential service time distributions to summarize the flows of jobs by a Markov model [16], and to derive a product-form expression of the JCTs. However, Jackson networks consider independent customers, whereas datacenter jobs DAGs introduce *dependency* between tasks. On top of this, DNN inference workload runtimes do not match exponential distributions, and only approximations are available for Jackson networks with non-exponentially distributed service times.

Optimal Resource Allocation: Several papers have proposed scheduling methods for non-DNN jobs. For example, [42] derived optimal query execution plans for geo-distributed data. Similarly, [43], [44] studied latency-optimal scheduling schemes and proposed methods to optimize resource utilization. Finally, [45] explored dependency-aware scheduling. Yet, these works do not consider DAG parallelism for user-facing inference jobs with a complex DAG structure, often because they are geared toward smaller jobs with DAGs reduced to one task. On top of this, these solutions do not take into account *network congestion*.

VI. Acknowledgements

The MIT-affiliated authors are supported by NSF SHF-2107244, NSF CAREER-2144766, NSF PPoSS-2217099, NSF CNS-2211382, NSF CNS-2008624, NSF FuSe-TG-2235466, and the Center for Ubiquitous Connectivity (CUbiC), sponsored by Semiconductor Research Corporation (SRC) and Defense Advanced Research Projects Agency (DARPA) under the JUMP 2.0 program.

This research was partially supported by the program "PEPR Networks of the Future" of France 2030.

Co-funded by the European Union (ERC, SENSIBILITÉ, 101077361). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

VII. Conclusion

In this paper, we present Nona, a stochastic, queueing-theory-based scheduler for DAG parallelism in datacenter clusters. Nona uses an optimization formulation to derive placement probability distributions minimizing average job completion time. Our approach takes into account both network and compute service and queueing times, and can easily be extended to consider other constraints (*e.g.*, memory). We show that Nona outperforms state-of-the-art heuristic-based solutions by up to $350\times$.

References

[1] OpenAI, "Gpt-4 technical report," 2023.

[2] L. Weng and G. Brockman, "Techniques for training large neural networks," 2022.

[3] S. Gheissi, S. Mahmood, and S. Ghorbani, "Burstiness in data center topologies," in *Proc. of the 3rd International CoNEXT Student Workshop*, ser. CoNEXT-SW '22. NY, USA: ACM, 2022, p. 29–31.

[4] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo, "Micro-burst in data centers: Observations, analysis, and mitigations," in *2018 IEEE 26th International Conference on Network Protocols*, 2018, pp. 88–98.

[5] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proc., ACM SIGCOMM Computer Communication Review*, vol. 44, 2014, pp. 455–466.

[6] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," *Proc. of Machine Learning and Systems*, vol. 1, pp. 1–13, 2019.

[7] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. of the ACM Special Interest Group on Data Communication*, New York, NY, USA, 2019, p. 270–288.

[8] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. of the Thirteenth EuroSys Conference*. NY, USA: ACM, 2018.

[9] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective cluster scheduling for deep learning," in *13th USENIX Symp. on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 595–610.

[10] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A gpu cluster manager for distributed deep learning," in *Proc. of the 16th USENIX Conference on Networked Systems Design and Implementation*, USA, 2019, p. 485–500.

[11] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing, "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," in *15th USENIX Symp. on Operating Systems Design and Implementation*, Jul. 2021, pp. 1–18.

[12] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla, "Themis: Fair and efficient GPU cluster scheduling," in *17th USENIX Symp. on Networked Systems Design and Implementation*, Santa Clara, CA, Feb. 2020, pp. 289–304.

[13] J. M. Tarnawski, A. Phanishayee, N. Devanur, D. Mahajan, and F. Nina Paravecino, "Efficient algorithms for device placement of dnn graph operators," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 15 451–15 463.

[14] I. Grosof, Z. Scully, M. Harchol-Balter, and A. Scheller-Wolf, "Optimal scheduling in the multiserver-job model under heavy traffic," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 3, dec 2022.

[15] R. Nelson, *Probability, stochastic processes, and queueing theory: the mathematics of computer performance modeling*. Springer Science & Business Media, 2013.

[16] S. Feizi, M. Médard, and M. Effros, "Compressive sensing over networks," in *Proc., Annual Allerton Conference on Communication, Control, and Computing*, Sep. 2010, pp. 1129–1136.

[17] Z. Li, L. Zheng, Y. Zhong, V. Liu, Y. Sheng, X. Jin, Y. Huang, Z. Chen, H. Zhang, J. E. Gonzalez, and I. Stoica, "AlpaServe: Statistical multiplexing with model parallelism for deep learning serving," in *17th USENIX Symp. on Operating Systems Design and Implementation (OSDI 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 663–679.

[18] D. Bertsekas and R. Gallager, *Data networks (2nd ed.)*. USA: Prentice-Hall, Inc., 1992.

[19] A. Y. Khintchine, "Mathematical theory of a stationary queue," *Matematicheskii Sbornik*, vol. 39, no. 4, p. 73–84, 1932.

[20] F. Pollaczek, "Über eine aufgabe der wahrscheinlichkeitstheorie. i," *Math Z*, vol. 32, p. 64–100, 1930.

[21] J. R. Jackson, "Jobshop-like queueing systems," *Management Science*, vol. 10, no. 1, pp. 131–142, 1963.

[22] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, p. 56–65, oct 2016.

[23] X. Yao and A. Klimovic, "Deltazip: Multi-tenant language model serving via delta compression," 2023.

[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[25] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[26] A. Z. Karen Simonyan, "Very deep convolutional networks for large-scale image recognition." in *Proc. of the 3rd International Conference on Learning Representations (ICLR)*, May 2015.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Red Hook, NY, USA, 2012, p. 1097–1105.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[29] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Automated model-less inference serving," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, Jul. 2021, pp. 397–411.

[30] H. Geng, D. Zeng, and Y. Li, "Performance efficient layer-aware dnn inference task scheduling in gpu cluster," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 2242–2247.

[31] D. J. Daley and D. Vere-Jones, *An introduction to the theory of point processes: volume II: general theory and structure*. Springer, 2008.

[32] J. R. Jackson, "Networks of waiting lines," *Operations Research*, vol. 5, no. 4, pp. 518–521, 1957.

[33] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, 1985.

[34] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," in *Proc. of the 27th ACM Symp. on Operating Systems Principles*, NY, USA, 2019, p. 1–15.

[35] W. Chun-Feng, L. San-Yang, and S. Pei-Ping, "Global optimization for sum of geometric fractional functions," *Applied Mathematics and Computation*, vol. 216, no. 8, pp. 2263–2270, 2010.

[36] F. Bazikar and M. Saraj, "Solving fractional geometric programming problems via relaxation approach," *MatLAB J*, vol. 1, no. 3, pp. 1–14, 2018.

[37] Z. Ye, W. Gao, Q. Hu, P. Sun, X. Wang, Y. Luo, T. Zhang, and Y. Wen, "Deep learning workload scheduling in gpu datacenters: A survey," *ACM Comput. Surv.*, vol. 56, no. 6, jan 2024.

[38] G. R. Gilman, S. S. Ogden, R. J. Walls, and T. Guo, "Challenges and opportunities of dnn model execution caching," in *Proc. of the Workshop on Distributed Infrastructures for Deep Learning*, ser. DIDL '19. New York, NY, USA: ACM, 2019, p. 7–12.

[39] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proc. of the Twenty-Fourth ACM Symp. on Operating Systems Principles*, ser. SOSP '13. New York, NY, USA: ACM, 2013, p. 69–84.

[40] W. Wang, M. Khazraee, Z. Zhong, M. Ghobadi, Z. Jia, D. Mudigere, Y. Zhang, and A. Kewitsch, "TopoOpt: Co-optimizing network topology and parallelization strategy for distributed training jobs," in *20th USENIX Symp. on Networked Systems Design and Implementation (NSDI 23)*, Boston, MA, Apr. 2023, pp. 739–767.

[41] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019.

[42] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "CLARINET: WAN-aware optimization for analytics queries," in *Proc., USENIX Symp. on Operating Systems Design and Implementation*, 2016, pp. 435–450.

[43] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *Proc. of the 11th USENIX Conference on Operating Systems Design and Implementation*, USA, 2014, p. 285–300.

[44] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *SIGOPS European Conference on Computer Systems (EuroSys)*, Prague, Czech Republic, 2013, pp. 351–364.

[45] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "Graphene: Packing and dependency-aware scheduling for data-parallel clusters," in *Proc., USENIX Symp. on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 81–97.