Plebiscito: A Decentralized, Bandwidth-Aware Architecture for Distributed Learning Workloads

Andrea Pinto
Saint Louis University
St. Louis, MO, USA
andrea.pinto.1@slu.edu

Stefano Galantino
Politecnico di Torino
Torino, Italy
stefano.galantino@polito.it

Fulvio Risso
Politecnico di Torino
Torino, Italy
fulvio.risso@polito.it

Flavio Esposito
Saint Louis University
St. Louis, MO, USA
flavio.esposito@slu.edu

***Abstract*—As large-scale AI training increasingly relies on distributed GPU clusters, data-center network bandwidth has become a critical bottleneck. Existing systems often overlook real-time link utilization during job placement, leading to suboptimal scheduling decisions that exacerbate congestion and increase Job Completion Time (JCT). To address this gap, we introduce Plebiscito, a policy-based architecture that enables bandwidth-aware job placement in distributed AI training clusters. Using a distributed max-consensus auction protocol, nodes autonomously bid on incoming jobs based on local resource availability and network conditions. We formulate this as a network utility maximization problem and prove that our decentralized algorithm achieves a (1 - 1/e) optimality bound. Experiments on a Kubernetes-based prototype and through large-scale, trace-driven simulation show that Plebiscito reduces JCT, improves bandwidth utilization, and lowers allocation failure rates compared to bandwidth-agnostic baselines.**

***Index Terms*—Distributed Training, Resource Orchestration, Network Management, Data Center Systems, Bandwidth-Aware Allocation, Distributed Auctions.**

I. Introduction

ARTIFICIAL Intelligence (AI) models continue to grow in scale and complexity; training or retraining when data distribution drifts is unfeasible on a single machine, and GPU clusters in geolocated clouds have become a necessity, especially for applications utilizing locally collected data or when massive data transfers are inconvenient. This paradigm shift has inevitably transformed datacenter networks, which often become a significant performance bottleneck during training operations [1], [2]. Once the iterative training process begins, each processing unit (GPU or TPU) sends large amounts of data, mostly neural network parameters and gradients. Researchers have demonstrated that such intense, synchronized inter-process communication can quickly overwhelm datacenter fabrics, creating severe network congestion that throttles training throughput, inflates operational costs, and ultimately slows down training time [2], [3].

Modern data centers receive a continuous influx of training jobs, each differing in the amount of resources required and spanning various numbers of GPUs. This necessitates the efficient, coordinated allocation of both computational and networking resources. Existing cluster schedulers for machine learning (ML) workloads, however, focus predominantly on the fair and efficient allocation of *compute* resources, such as GPUs and CPUs [4]–[7]. Although sophisticated, these systems operate largely under a critical and simplifying assumption: network bandwidth is a static, unconstrained commodity. They make placement decisions by assigning the components of a distributed training job to physical nodes without being aware of the underlying network states. Such a network-agnostic approach frequently places workers on connected nodes that may not satisfy the network-intensive communication requirements, creating performance bottlenecks that stall distributed computation and significantly increase the job's end-to-end duration, or Job Completion Time (JCT).

Current solutions to address this challenge often introduce substantial overhead, rely on incomplete heuristics, or depend on hardware. Some efforts propose re-engineering the network data plane with in-network aggregation techniques [3], [8], leveraging expensive programmable switches to offload aggregation tasks. While valid, the practicality of these methods is constrained by the high cost of specialized hardware, limited on-chip memory, and reliance on complex custom protocols. Other cluster managers, such as Tiresias [4], claim placement awareness by consolidating jobs to minimize spatial fragmentation. However, they rely on static topological assumptions rather than real-time link utilization, frequently assigning workers to paths that are physically close but already saturated by cross-traffic. Alternatively, software-based communication schedulers like Cassini [9] and Crux [10] attempt to mitigate contention *after* placement by interleaving network transfers during execution. Yet, as Crux itself highlights, such interleaving methods struggle to coordinate at scale and rapidly lose effectiveness under heavy cluster loads. Despite significant progress, existing approaches still lack a proactive, software-driven mechanism to anticipate network contention. There remains a pressing need for a system that can intelligently plan workload placement *before* execution, leveraging real-time network insights to prevent congestion and achieve the scalability of distributed systems without the cost or rigidity of hardware-centric designs.

To this end, we present Plebiscito, a fully decentralized architecture for managing distributed training workloads. [1]

ISBN 978-3-903176-82-9 © 2026 IFIP

[1] "Plebiscito" (from Latin plebiscitum) means a decision or decree adopted by the common people (the plebs), historically, a vote passed by the Plebeian Council in ancient Rome.

Plebiscito frames the placement task as a Network Utility Maximization problem [11]. It employs a distributed, asynchronous max-consensus auction in which physical nodes collaboratively determine the optimal placement of incoming jobs. Each node autonomously bids based on a utility function that jointly considers its local compute capacity (CPU/GPU availability) and, crucially, the real-time bandwidth available on its network paths. Although both network latency and bandwidth are key performance metrics, their impact is workload-dependent. Inference is latency-bound, requiring millisecond response times. In contrast, large-scale, data-parallel training is fundamentally throughput-bound. The defining communication pattern is the periodic, synchronous exchange of large gradient tensors across all workers. This step often dominates total training time, scaling inversely with available bandwidth. Consequently, while we aim to find the optimal resource-aware placement, we focus on mitigating bandwidth contention, which is the most significant bottleneck affecting training throughput and JCT.

Our work makes the following contributions: (i) We design *Plebiscito*, a fully decentralized architecture that orchestrates ML workloads by integrating real-time network states directly into resource allocation decisions. Unlike centralized schedulers, Plebiscito eliminates single points of failure, enhancing resilience, adaptability to fluctuating network conditions, and fairness in resource sharing. (ii) At the core of Plebiscito, we propose a distributed max-consensus auction algorithm to solve the bandwidth-aware placement problem. We formally prove that this algorithm achieves a $(1 - 1/e)$ approximation of the optimal allocation, that this bound is optimal (unless P=NP), and that convergence holds even under the relaxed condition of pseudo-submodular utility functions. (iii) We implement and evaluate Plebiscito using a Kubernetes-based prototype and extensive, large-scale, trace-driven simulations. Our results demonstrate that our bandwidth-aware orchestration reduces the average JCT by up to $1.5\times$, lowers allocation failure rates, and improves overall bandwidth utilization compared to existing resource-agnostic baselines.

II. Related Work

Efficient orchestration of compute and network resources is essential for distributed ML, yet most prior systems lack dynamic bandwidth awareness.

Compute-Centric Schedulers. Standard production infrastructure, such as Kubernetes [12] and YARN [13], allocates jobs to dedicated GPUs without considering network topology, often leading to poor cluster utilization [14]. Optimizers such as Ray [15] and Clipper [16] accelerate task management but defer placement to these underlying, network-agnostic schedulers. Similarly, MLaaS schedulers [17], [18] and fairness-driven algorithms, including DRF [5] and THEMIS [19], focus strictly on maximizing compute shares or placement fairness. While cluster managers like Tiresias [4] attempt to reduce spatial fragmentation through profile-based consolidation, these systems ultimately neglect real-time network contention, creating bottlenecks that Plebiscito mitigates via proactive, bandwidth-aware placement.

Network Optimizations. Other approaches attempt to synchronize or structure network resource usage. Muri [20] and Cassini [9] exploit iterative computation patterns to interleave network transfers; however, as noted by Crux [10], these scheduling methods struggle to coordinate and scale under high cluster loads. Meanwhile, dynamic orchestrators like Optimus [21] and architectures like BytePS [22] optimize for specific topologies (e.g., Parameter Servers) but lack a generalized mechanism to avoid fabric congestion. Plebiscito is designed to complement these systems, providing a foundational, bandwidth-aware allocation layer that can operate efficiently beneath these dynamic topology policies.

Plebiscito's Advantage. Similarly to past distributed resource allocation protocols, we also use a max-consensus auction [11], [23], [24]. However, unlike prior work, Plebiscito embeds real-time link utilization directly into a decentralized, asynchronous max-consensus auction. This architectural shift allows it to proactively avoid network hotspots before they form, delivering a provable $(1 - \frac{1}{e})$ approximation and convergence guarantees that existing centralized or heuristic-based solutions cannot match.

III. Background and System Model

In modern distributed settings, ML training is fundamentally a high-performance networking problem. The predominant scaling method, data-parallelism, relies on iterative communication where workers exchange model updates via Parameter Servers (PS) [25] or All-Reduce [26]. While All-Reduce offers high bandwidth efficiency, PS architectures often prove more flexible in heterogeneous clusters [21], [22]. However, as GPU performance continues to outpace data-center network bandwidth [3], the synchronous communication phase often dominates the total iteration time. Consequently, network-agnostic schedulers that overlook topology frequently place components on congested links, leaving GPUs idle.

To address this, we formulate the placement task by modeling the total training time per iteration, γ, as the sum of computation (γ_{train}), communication (γ_{comm}), and update latency (γ_{update}):

$$\gamma \;=\; \gamma_{\text{train}} \;+\; \underbrace{\max_{e \in \mathcal{E}} \frac{2\,B_e}{C_e}}_{\gamma_{\text{comm}}} \;+\; \gamma_{\text{update}}. \tag{1}$$

Crucially, the communication term γ_{comm} is governed by the bottleneck link e with the lowest capacity C_e relative to the data volume B_e. The choice of topology dictates the nature of this bottleneck: a PS architecture concentrates traffic at server nodes, creating massive ingress/egress hotspots proportional to the number of workers, whereas Ring All-Reduce distributes the load across peers but remains gated by the slowest link in the ring [3]. To minimize γ and accelerate training, an orchestrator must therefore identify a placement that satisfies these distinct topological bandwidth requirements, motivating

our formulation of the placement task as a network utility maximization problem.

A. A Network Utility Maximization Problem

We formulate our resource allocation problem as a network utility maximization problem and model it as a mixed-integer linear program (MILP). While not all MILPs are NP-hard [27], our specific formulation is NP-hard due to its combinatorial structure and infrastructure constraints. A formal reduction can be built from the *Generalized Assignment Problem (GAP)*, which is known to be NP-hard [28], motivating the need for a distributed approximation algorithm.

Consider an overlay network of N nodes, indexed by $n \in \mathcal{N}$. Each node offers computing resources ρ_{rn}, where $r \in \mathcal{R}$ indexes specific capacities (e.g., GPU, CPU, memory). The network edges are represented by an adjacency matrix $E \in \{0, \text{bw}\}^{N \times N}$, weighted by the physical link capacity. The primary design objective of *Plebiscito* is to efficiently orchestrate job allocations while dynamically managing network link capacities to achieve near-optimal bandwidth utilization.

A job $j \in \mathcal{J}$ consists of a set of components $\lambda \in \Lambda_j$ (e.g., Parameter Servers and workers). Each component requires specific computing resources $\varrho_{rj\lambda}$ and network capacities $\Theta_{comm_{j\lambda}}$. To orchestrate this, we seek to maximize a utility function $U_{nj\lambda}$ that quantifies the value of assigning component λ of job j to node n. Let the binary allocation variable $x_{nj\lambda} \in \{0,1\}$ indicate the assignment of a component to a node, and $y_{ej\lambda} \in \{0,1\}$ indicate the usage of link e by that component. The optimization problem is defined as:

$$\max_{\boldsymbol{x},\boldsymbol{y}} \sum_{n\in\mathcal{N}} \sum_{j\in\mathcal{J}} \sum_{\lambda\in\Lambda_j} U_{nj\lambda}(\boldsymbol{x}_j, \boldsymbol{y}_j) \tag{2}$$

$$\text{s.t.} \sum_{j\in\mathcal{J}} \sum_{\lambda\in\Lambda_j} \varrho_{rj\lambda} x_{nj\lambda} \le \rho_{rn}, \quad \forall n \in \mathcal{N}, \forall r \in \mathcal{R} \tag{3}$$

$$\sum_{j\in\mathcal{J}} \sum_{\lambda\in\Lambda_j} \Theta_{comm_{j\lambda}} y_{ej\lambda} \le e, \quad \forall e \in E \tag{4}$$

$$\sum_{n\in\mathcal{N}} x_{nj\lambda} = \Lambda_j, \quad \forall j \in \mathcal{J}, \forall \lambda \in \Lambda_j \tag{5}$$

$$x_{nj\lambda}, y_{ej\lambda} \in \{0,1\}, \forall n \in \mathcal{N}, \forall e \in E, \forall j \in \mathcal{J}, \forall \lambda \in \Lambda_j \tag{6}$$

Constraint (3) ensures that cumulative compute demands do not exceed local node capacities. Constraint (4) prevents network congestion by ensuring data flows do not exceed the physical link bandwidth. Constraint (5) enforces a conflict-free assignment for all job components, while constraint (6) restricts the decision variables to binary values.

B. Utility Function Library for Orchestration

To solve this NP-hard placement task via a decentralized max-consensus auction, we define the utility function $U_{nj\lambda} \in \mathbb{R}_+^{|N|}$. In *Plebiscito*, this function acts as a configurable policy tailored to infrastructure objectives. We decompose the utility into two principal components: a *resources-fraction* capturing local computational availability, and a *network-fraction* reflecting residual path bandwidth. In Th. 1, we show that if this utility function maintains positivity, monotonicity, and submodularity, our approximation algorithm achieves a provable optimality bound.

Resources-Fraction. We first formally define the residual (unallocated) capacity of resource r at node n as the total capacity minus the currently allocated demands:

$$\Delta_{rn} = \rho_{rn} - \sum_{j'\in\mathcal{J}} \sum_{\lambda'\in\Lambda_{j'}} \varrho_{rj'\lambda'}\, x_{nj'\lambda'} \tag{7}$$

The local resources fraction is then calculated as a weighted sum reflecting the relative importance of CPU (C), GPU (G), or memory (M):

$$f_n^{\text{comp}} = \sum_{r\in\{C,G,M\}} w_r \frac{\Delta_{rn}}{\varrho_{rj\lambda}} \tag{8}$$

where w is a vector in $\mathbb{R}^{|r|}$ representing the relative importance of resource w_r. Equal values lead to a balanced allocation, whereas different values allow for resource prioritization.

Network-Fraction. To satisfy constraint (4) in a fully decentralized setting, each agent maintains a local view of the overlay graph. The residual bandwidth Δ_e on any link e is its total capacity C_e minus the bandwidth B_ϕ consumed by the set of active flows $\mathcal{F}$, expressed as $\Delta_e = C_e - \sum_{\phi\in\mathcal{F}:\, e\in\text{path}(\phi)} B_\phi$. When evaluating the placement of component λ of job j on node n, the agent identifies the unique sequence of links $\text{path}(n, d_{j,\lambda})$ to the destination. The bottleneck is the link with the smallest capacity-to-demand ratio:

$$f_n^{\text{net}} = \min_{e\in\text{path}(n,d_{j,\lambda})} \frac{\Delta_e}{B_{j,\lambda}} \tag{9}$$

This fraction, bounded in $[0,1]$, quantifies the network's ability to sustain the transfer without violating physical limits, directly determining the achievable communication efficiency for the given allocation.

Plebiscito Utility. By combining these metrics, the max-consensus auction naturally favors hosts provisioned with both local compute and network capacity. A tuning parameter $\alpha \in [0,1]$ smoothly trades off emphasis between the two:

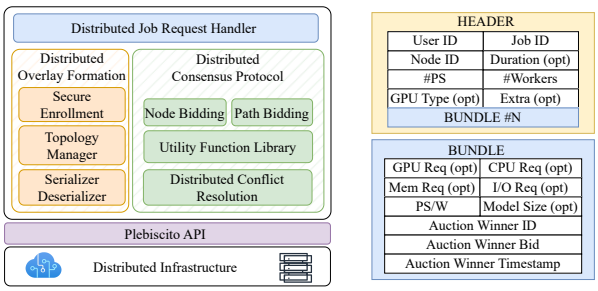
$$U_{nj\lambda} = \alpha \cdot f_n^{\text{comp}} + (1-\alpha) \cdot f_n^{\text{net}} \tag{10}$$

Remark: Because the formulation in Eq. 10 is monotone and each factor exhibits diminishing returns, our submodularity assumptions hold (Definition 2).

Utility Function Library. *Plebiscito* provides a configurable policy library; by substituting the f_n^{comp} component, it can subsume or reproduce various standard placement strategies. Table I details how *Plebiscito* supports distinct allocation behaviors, such as packing efficiency, fairness, or resource balancing. Plebiscito can then improve the overall allocation by balancing these different resource contributions.

TABLE I: The Utility function policy supported by Plebiscito subsumes related solutions.

Utility	Note
Smallest GPU First (SGF) [17]	$f_i^{\text{comp,SGF}} = -(\Delta_{\text{GPU},i} - \varrho_{\text{GPU},j})$ packs jobs, with $\Delta_{\text{GPU},i}$ the free GPU slots on node i and $\varrho_{\text{GPU},j}$ job j's GPU demand.
Largest GPU First (LGF)	$f_i^{\text{comp,LGF}} = \Delta_{\text{GPU},i} - \varrho_{\text{GPU},j}$ favors large GPU requests on well-provisioned nodes.
Apache YARN [13]	$f_i^{\text{comp,YARN}} = \min_{r\in\mathcal{R}}\big((\Delta_{r,i} - \varrho_{r,j})/\rho_{r,i}\big)$ maximizes the minimum normalized resource headroom after placement.
Tetris [29]	$f_i^{\text{comp,Tetris}} = \sum_{r\in\mathcal{R}}(\Delta_{r,i}/\rho_{r,i})\,\varrho_{r,j}$, where $\Delta_{r,i}$ is the residual amount of resource r on node i (i.e. total capacity $\rho_{r,i}$ minus currently allocated), and $\varrho_{r,j}$ the peak demand of job j.
Tiresias [4]	$f_j^{\text{comp,Tiresias}} = W_j\, t_j$, where W_j is a job-specific weight (e.g. priority) and t_j its expected runtime.
DRF [5]	$f_i^{\text{comp,DRF}} = -\max_{r\in\mathcal{R}}\big((u_{i,r} + \varrho_{r,j})/\rho_{r,n}\big)$, with $u_{i,r}$ the user's current resource allocation. This minimizes the user's dominant share.
Themis [19]	We capture co-location interference via $f_i^{\text{comp,Themis}}(\mathbf{G}_i) = T_i^{\text{sh}}(\mathbf{G}_i)/T_i^{\text{ind}}$, where T_i^{sh} is the runtime when job i shares hosts under allocation $\mathbf{G}_i$, and T_i^{ind} its isolated runtime.



(a) *Plebiscito Architecture* (b) *Plebiscito Protocol*

Fig. 1: (a) Plebiscito API and mechanisms to orchestrate distributed training with guarantees. (b) Bundles carry the states needed for the distributed max-consensus auction.

IV. Plebiscito Design

To solve the NP-hard placement problem, we designed *Plebiscito*, a fully decentralized architecture driven by an asynchronous max-consensus auction. Nodes evaluate job requests, bid based on their local utility, and iteratively exchange messages with neighbors to reach a global agreement.

A. Architecture and Protocol State

To participate in the decentralized orchestration, each physical node in the cluster runs a dedicated Plebiscito agent. This agent operates through two cooperative mechanisms. First, a *Topology Manager* is responsible for seamlessly integrating the node into the network overlay, managing secure enrollment, and tracking peer discovery. Second, the *Max-Consensus Protocol* engine serves as the decision-making core, actively evaluating local resource availability and executing the distributed auction logic.

Algorithm 1 Plebiscito Event-Driven Max-Consensus

State: Local $bundle^*(j,\tau) \leftarrow (b,a,t)$ initialized to $\emptyset$

procedure On Local Bidding Event(job j)
  for each unassigned $\lambda \in \Lambda_j$ **do**
    if node has resources and bandwidth for λ **then**
      $b_\lambda \leftarrow$ Utility(λ) ▷ Evaluate Eq. 10
      Update $bundle^*_\lambda \leftarrow (b_\lambda, \text{NodeID}, \text{currentTime})$
    end if
  end for
  if $bundle^*(j)$ changed **then** Broadcast($bundle^*$ to $\mathcal{N}_n$)
  end if
end procedure

procedure On Receive Bundle(received $bundle^{rx}$ from n')
  $changed \leftarrow$ False
  for each $\lambda \in \Lambda_j$ **do**
    if $bundle^{rx}_\lambda.b > bundle^*_\lambda.b$ **or**
   ($bundle^{rx}_\lambda.b == bundle^*_\lambda.b$ **and** $bundle^{rx}_\lambda.t > bundle^*_\lambda.t$) **then**
      $bundle^*_\lambda \leftarrow bundle^{rx}_\lambda$ ▷ Outbid by neighbor
      Release locally held resources for λ
      $changed \leftarrow$ True
    end if
  end for
  if $changed$ **then** Broadcast($bundle^*$ to $\mathcal{N}_n$)
  end if
end procedure

During an auction phase, nodes exchange information using a specialized, lightweight protocol format, as depicted in Fig. 1b. The protocol header is designed to carry essential routing metadata—specifically the *UserID, JobID*, and the forwarding *NodeID*—alongside the structural requirements of the incoming job, such as the requested number of Parameter Servers and worker instances.

The cornerstone of the decentralized agreement is the local state maintained by each participating node. To achieve consensus without a central coordinator, every node n actively manages a local *Bundle* (Fig. 1b). This bundle acts as a snapshot of the auction state for a given job j at any discrete communication instance τ. Rather than storing an expensive global view of the network, the agent efficiently tracks only three critical state variables for each job component $\lambda \in \Lambda_j$. Specifically, it records the highest known bid value $b^n_\lambda(j,\tau) \in \mathbb{R}_+$, which quantifies the best utility offered by any node so far. It also logs $a^n_\lambda(j,\tau) \in \{0,1\}$, indicating the identifier of the node currently leading the auction for that specific component. Finally, to ensure strict causal ordering and safely resolve bidding conflicts in a fully asynchronous environment, the bundle includes $t^n_\lambda(j,\tau) \in \mathbb{N}_+$, which timestamps exactly when that leading bid was originally generated. By continuously exchanging and comparing these compact bundles, the cluster naturally converges toward a unified placement decision.

B. Distributed Max-Consensus Mechanism

The protocol operates asynchronously, alternating between a *Bidding Phase* (in which nodes assess their local capabilities against job requirements) and an *Agreement Phase* (in which conflicts are resolved). Algo. 1 details this event-driven logic.

Definition 1 (Max-Consensus)**.** *At each communication instance $\tau+1$, node n updates its local information for each*

component λ based on the information from its neighbors $\mathcal{N}_n$ (including itself). The node selects the state from the neighbor n' possessing the highest timestamped bid: $n' = \arg\max_{k \in \mathcal{N}_n} \{t^k_\lambda(j,\tau)\}$. Ties are broken using a predetermined rule (e.g., highest bid value, then lowest node identifier). Max-consensus is achieved with convergence time $\overline{\tau}$ if, for all $\tau \geq \overline{\tau}$ and for all $n, m \in \mathcal{N}$: $a^n_\lambda(j,\tau) = a^m_\lambda(j,\tau)$, and $b^n_\lambda(j,\tau) = b^m_\lambda(j,\tau), \quad \forall \lambda \in \Lambda_j$.

Implicit Gang Scheduling. Plebiscito naturally enforces gang scheduling without explicit hardcoding. Because network paths between collocated components possess higher residual bandwidth, a node that wins one component will generate highly competitive utility scores for subsequent components of the same job. This leads to organic co-location, reducing inter-job contention.

V. Plebiscito Performance Guarantees

We establish formal guarantees for Plebiscito's placement quality, algorithmic optimality, and convergence time.

A. Approximation Bound and Optimality

Our theoretical foundation rests on submodularity, which models the economic concept of diminishing returns.

Definition 2 (Submodular Function)**.** *Given a finite set Λ_j, a set function $U : 2^{\Lambda_j} \to \mathbb{R}$ is submodular if, for all subsets $\lambda' \subseteq \lambda \subseteq \Lambda_j$ and for every element $\lambda'' \in \Lambda_j \setminus \lambda$: $U(\lambda \cup \{\lambda''\}) - U(\lambda) \leq U(\lambda' \cup \{\lambda''\}) - U(\lambda')$.*

The residual capacity function defined in Eq. 7 is positive, monotone, and submodular, allowing us to formally bound the quality of our decentralized auction.

Theorem 1. *The Plebiscito consensus algorithm achieves a $(1 - \frac{1}{e})$-approximation to the optimal job assignment when the utility functions are positive, monotone, and submodular.*

Proof. Let $S \subseteq \mathcal{N} \times \Lambda_j$ be a partial assignment with total utility $F(S)$, and marginal gain $\Delta((n,\lambda) \mid S) = F(S \cup \{(n,\lambda)\}) - F(S)$. At each iteration, nodes bid their marginal gains. The max-consensus rule globally propagates the maximal bid, making the assignment $(n^\star, \lambda^\star) \in \arg\max_{(n,\lambda)} \Delta((n,\lambda) \mid S)$ identical to a centralized greedy algorithm. By the Nemhauser–Wolsey–Fisher theorem [30] for maximizing a monotone submodular function subject to a partition constraint, it achieves a $(1 - \frac{1}{e})$-approximation. □

Theorem 2. *The Plebiscito resource allocation problem cannot be approximated within a ratio better than $(1 - \frac{1}{e})$ in polynomial time, unless $P = NP$.*

Proof. We reduce from the Budgeted Maximum Coverage Problem (BMCP) [31]. Mapping components to universe elements, nodes to subsets, resource constraints to costs, and utility to coverage perfectly translates our formulation. Since BMCP cannot be approximated beyond $(1 - \frac{1}{e})$ unless $P = NP$, achieving a tighter bound for Plebiscito would contradict this known hardness. □

Handling Practical Deviations. If colocation causes utilities to increase (violating strict submodularity), we enforce convergence by applying a *warping function* to penalize repeated allocations on node n:

$$\mathcal{F}_{n\lambda j} = U_{n\lambda j} - \delta \cdot |\Lambda_j^{n,\text{assigned}}| \tag{11}$$

This penalty ($\delta > 0$) ensures the marginal utility of assigning new components strictly decreases, restoring pseudo-submodularity and preserving the approximation bound while avoiding auction oscillations.

B. Convergence Guarantee

Theorem 3. *For a network of N bidders with diameter D and a job of Λ components, Plebiscito achieves consensus within $O(\Lambda \cdot N \cdot D)$ iterations.*

Proof. By induction. For one component ($\Lambda = 1$), the winning bid requires at most D iterations to propagate. In an adversarial worst-case scenario, $N - 1$ nodes sequentially submit slightly higher bids, requiring $O(N \cdot D)$ iterations. Because the bidding process for $k + 1$ components is independent of the first k, converging Λ components takes $O(\Lambda \cdot N \cdot D)$ iterations. □

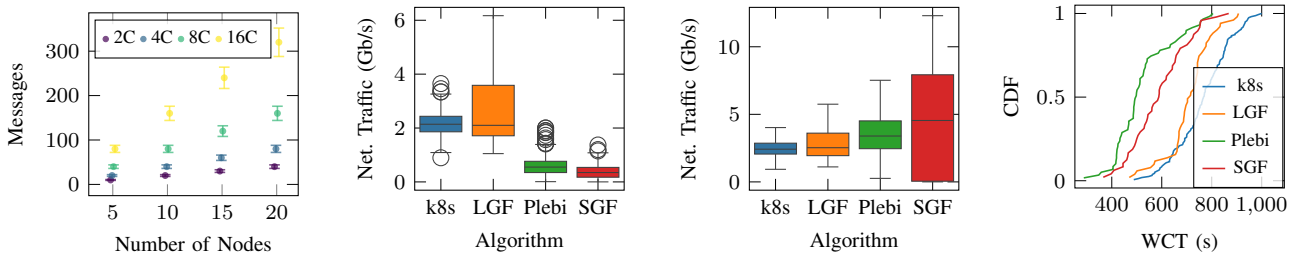
This linear scalability with respect to N validates the practicality of our decentralized approach. A comparable centralized scheduler would face an $O(N^2)$ communication and computation bottleneck [32] when simultaneously gathering state and resolving conflicts.

VI. Implementation and Prototype Evaluation

We implemented Plebiscito as a custom, out-of-process scheduler deployed as a Kubernetes (K8s) *DaemonSet*. Each agent operates as a lightweight HTTP server that maintains thread-safe views of local resources and cluster topology. Agents dynamically discover peers via the K8s API and execute the max-consensus auction over UDP, exchanging JSON-serialized bid bundles.

To handle distributed asynchrony, Plebiscito employs a two-tier timeout: a *quiescence* timeout (e.g., 500ms) detects stable consensus when message propagation stops, while a hard timeout (e.g., 1s) safely aborts and purges failed auctions during severe network partitions. Once consensus is reached, the initiating agent bypasses the default K8s scheduler by directly injecting the winning node's identifier into the target Pod's `spec.nodeName` field, seamlessly enforcing bandwidth-aware placement.

We validated the prototype on the FABRIC testbed [33]. The agent proved highly efficient for production nodes, consuming negligible CPU, only ≈ 75 MB of memory, and averaging tiny protocol messages of 500–600 bytes. We evaluated physical placement efficacy using PyTorch-based training [34] (ResNet-110 [35] on CIFAR-100 [36]) across a 6-node GPU cluster with throttled 5Gbps/10Gbps links. To scale our evaluation beyond physical GPU limits, we also built a trace-driven emulator over a 50-node Leaf-Spine topology using FRRouting [37], submitting 50 distributed jobs to analyze inter-node (NIC) and intra-node (localhost) network utilization.



(a) Messages Overhead (b) Physical interface transmit (c) Localhost interface transmit (d) Job training time

Fig. 2: *System Prototype*. The message overhead to reach consensus among Plebiscito agents grows with the number of nodes and workers to place (Fig. 2a). If possible, Plebiscito reduces inter-node bandwidth contention (Fig. 2b), while achieving higher local intra-node bandwidth utilization (Fig. 2c). Plebiscito strikes a better WCT compared to allocation policies more favorable to bin packing (SGF) or fairness (LGF) and the Kubernetes default allocator (Fig. 2d).

As shown in Fig. 2a, the protocol overhead remains consistently low, scaling linearly and reaching consensus within 300 messages even for 16-component jobs. Focusing on network utilization, Fig. 2b and Fig. 2c reveal a stark contrast in orchestrator behavior. The K8s default scheduler and LGF, being bandwidth-agnostic, severely over-utilize physical network interfaces, creating inter-node bottlenecks. Conversely, while SGF implicitly reduces inter-node traffic through aggressive bin-packing, it overloads the intra-node (localhost) interfaces.

Plebiscito optimally balances intra-node and inter-node communication. By explicitly factoring both network layers into its utility function, Plebiscito successfully avoids physical congestion without creating localhost bottlenecks. Consequently, Plebiscito achieves the lowest overall training time (Fig. 2d), significantly outperforming SGF, which suffers an approximate 20% increase in training time due to localhost saturation.

VII. Evaluation Results

After presenting our testbed solution, we evaluate higher-scale results using a trace-driven discrete-event simulator, highlighting the critical importance of bandwidth-aware placement. We compare Plebiscito's results with different utility functions (Tab. I), against baseline orchestrators, demonstrating that our decentralized, bandwidth-aware placement reduces total training time by up to $1.5\times$ in large-scale data center scenarios. We assess system efficiency via Job Completion Time (JCT) and throughput via Wall Clock Time (WCT). We measure robustness under resource pressure using the Allocation Failure Rate (AFR), i.e., the fraction of jobs failing first-attempt placement.

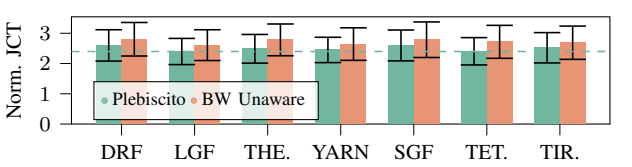
Our trace-driven simulator models a 100-node heterogeneous data center (each with up to 8 GPUs and 96 CPU cores) arranged in a leaf-spine topology (5 leaf, 2 spine switches) with a 3:1 oversubscription ratio. We evaluate 25Gbps and 100Gbps bottleneck scenarios. Workloads are derived from public Alibaba cluster traces [38], which exhibit a Pareto-like bandwidth demand distribution [17] (Fig. 4).

For each run, we sample 250 jobs, each composed of $\Lambda \in [2, 32]$ distributed components (supporting both Parameter Server and ring-all-reduce architectures). To evaluate system behavior under contention and recovery, jobs arrive via a Poisson process (mean inter-arrival time of 15 seconds) to induce peak resource saturation. We run 30 workloads per configuration for statistical significance. We assume a heterogeneous cluster in which weaker nodes are penalized by reduced utility values, thereby naturally maintaining efficiency. Unless otherwise specified, Plebiscito equally balances compute and network preferences ($\alpha = 0.5$ in Eq. 10).

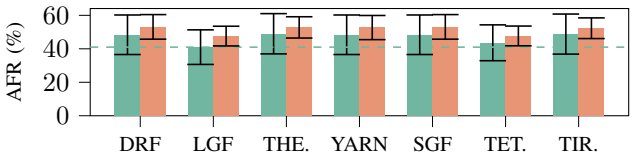
A. Performance Gains from Bandwidth-Aware Orchestration

Here, we test the workload across the settings described earlier to compare the JCT obtained by varying the different utility functions (Tab. I), highlighting the importance of being bandwidth-aware. We found that, regardless of the allocation policy, Plebiscito achieves JCT up to $1.5\times$ lower than that of bandwidth-unaware allocators.

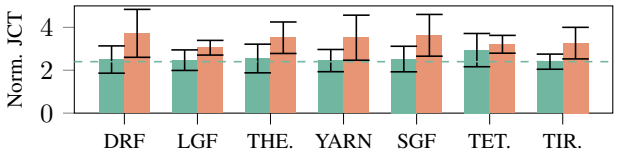
Comparing Fig. 3a (100Gbps network links on the servers) and Fig. 3c (25Gbps), which report the normalized average JCT, it is evident that higher bandwidth leads to shorter JCT. Nevertheless, Plebiscito consistently improves performance across all tested utility functions Eq. 10. In both scenarios, as the allocation across data center nodes and the network becomes more congested, relying solely on local resource availability becomes insufficient. The worst bandwidth-unaware performances are achieved by YARN, DRF, and Tetris. Instead, SGF employs a packing strategy that can increase local leaf switch congestion, whereas LGF attempts to evenly distribute jobs across nodes, potentially leading to uniform bandwidth saturation and, consequently, higher congestion. In contrast, Themis and Tiresias introduce more placement-aware strategies. Themis balances fairness and performance using a finish-time fairness metric, while Tiresias relies on profile-based consolidation. However, these methods trade off placement quality for long-term performance and may be too slow to adapt to dynamic conditions, whereas Plebiscito captures trade-offs more effectively in real time. The same pattern holds for the AFR metric in Fig. 3b, where Plebiscito is more successful in meeting bandwidth requirements, resulting in fewer failed allocations. A job fails on the first attempt when



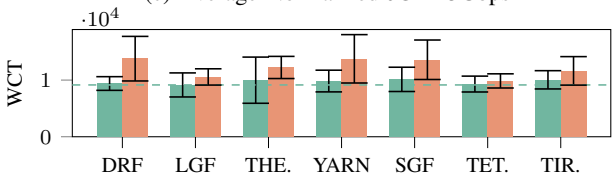
(a) Average Normalized JCT 100Gbps



(b) Allocation Failure Rate 100Gbps

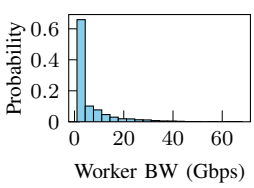


(c) Average Normalized JCT 25Gbps

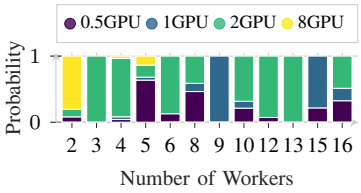


(d) Wall Clock Time (sec) 25Gbps

Fig. 3: *Plebiscito mitigates network bottlenecks, reducing JCT, WCT, and AFR.* Fig. 3a and Fig. 3b show improvements in Average Normalized JCT and AFR under 100Gbps node bottleneck, while Fig. 3c and Fig. 3d highlight gains in JCT and WCT under 25Gbps node bottleneck. Plebiscito Network-aware placements lead up to 1.5× JCT, WCT, and AFR improvements.



(a) Bandwidth requirements.



(b) GPU requirements.

Fig. 4: *Workload characterization*: (a) Bandwidth follows a Pareto distribution. (b) Workers range from 2 to 16, with varying numbers of GPUs required per worker.

its resource requirements, especially bandwidth for distributed training jobs, cannot be met. In this case, bandwidth-aware scheduling leads to better bandwidth utilization, lower AFR, higher resource utilization, and reduced idle time. Finally, the WCT under 25 Gbps in Fig. 3d shows that bandwidth-unaware allocations result in heavier bandwidth contention and poorer allocations, leading to higher WCT.

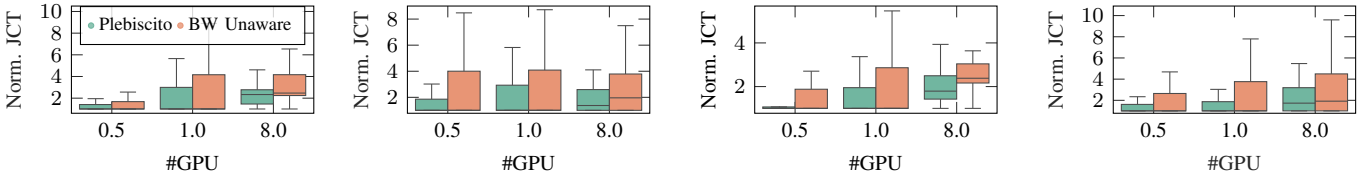
Plebiscito bandwidth awareness achieves up to 1.5× improvement, enabling jobs to run on the cluster with reduced JCT, fewer bandwidth contentions, and optimal performance.

B. JCT Sensitivity to Resource Footprint

To better understand how resource demands and placement strategies impact the JCT, we highlight the behavior of jobs whose tail latency exceeds their nominal execution time, which can be estimated for each job based on its given duration. A longer duration is a direct consequence of bandwidth contention, which typically occurs at peak system utilization. Specifically, we analyze JCT as a function of two key dimensions: the number of GPUs per worker and the total number of nodes provisioned. We compare Plebiscito's JCT under the suite of utility functions introduced earlier. This breakdown reveals clear patterns: jobs with high GPU requirements per worker suffer more when the allocation policy is not bandwidth-aware. Similarly, jobs distributed across many nodes experience increased JCT, depending on how sensitive the utility function is to network versus compute headroom. This analysis highlights how distinct resource profiles map to JCT improvements under each placement strategy.

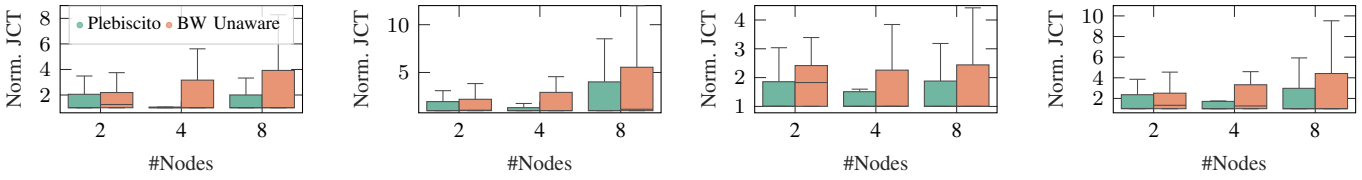
GPU Sensitivity. Fig. 5 shows job performance grouped by GPU requirements per worker, focusing on jobs requesting 0.5, 1, or 8 GPUs. For each category, we include only jobs whose JCT exceeds their expected duration due to bandwidth bottlenecks. A general trend emerges: as GPU requirements per worker increase, baseline allocation mechanisms show greater slowdowns, especially when compared to the bandwidth-aware Plebiscito allocation. The long upper whiskers in Fig. 5a, Fig. 5b, and Fig. 5c suggest that a small fraction of jobs endure severe contention; likely those whose workers end up straddling the data center spine. Plebiscito's bandwidth-aware bids reduce these extreme tails. The less severe JCT performance is observed in Fig. 5c, where the Themis policy results in slowdowns up to 5× the target JCT. In contrast, Plebiscito mitigates these slowdowns through its bandwidth-aware allocation strategy and Gang Scheduling policy, which places workers of the same job in close proximity to reduce communication overhead, as described next.

We then evaluate Plebiscito's Gang Scheduling performance by enforcing bandwidth-aware allocation and analyzing its impact under varying network capacities. Specifically, Fig. 6 shows the JCT as a function of the number of nodes used per job (2, 4, and 8). We observe how different allocation strategies affect performance under these conditions. We further examine the spatial distribution of job components across nodes to assess whether Gang Scheduling is effectively enforced. This analysis focuses exclusively on jobs requiring one GPU per worker. As shown, a consistent trend emerges: the more nodes used per job, the higher the JCT. Notably, Themis exhibits strong performance in these cases (Fig. 6c). However, it may still benefit from Plebiscito's bandwidth-aware placement, which attempts to colocate workers to minimize path length and avoid bottlenecks.



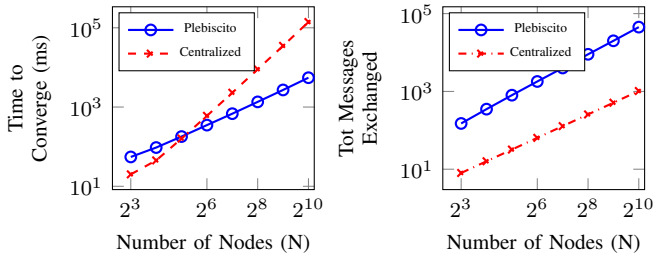
(a) DRF (b) YARN (c) THEMIS (d) LGF

Fig. 5: *JCT vs. GPUs number.* Tail JCT jobs distribution shows the impact of different allocations depending on the number of required GPUs. The higher the number of GPUs, the higher the JCT, which is mitigated with Plebiscito BW aware formulation.



(a) DRF (b) YARN (c) THEMIS (d) LGF

Fig. 6: *JCT vs. allocated Nodes number.* Tail JCT jobs distribution shows the impact of different allocations depending on the number of nodes used to allocate the job. The higher the number of used nodes, the higher the JCT, which is mitigated with Plebiscito BW aware formulation.



(a) Wall-clock time to placement. (b) Communication overhead.

Fig. 7: Empirical scalability for placing a 16-component job (log-log scale).

Conversely, when a job spans fewer nodes, bandwidth consumption is more localized and results in lower JCT across all utility functions (Fig. 6). Again, Plebiscito consistently prioritizes bandwidth feasibility, reinforcing Gang Scheduling by keeping job components closer together on the node overlay.

C. Protocol Performance and Scalability

While theoretical bounds are essential, a practical orchestrator must balance decision quality with latency at scale. To evaluate Plebiscito's real-world viability, we measure the end-to-end wall-clock convergence time and communication overhead for a 16-component job as the cluster scales up to 1024 nodes. We compare against a modeled centralized scheduler using our event-driven simulator, scaling up baseline metrics obtained from our FABRIC deployment.

The results (Fig. 7) confirm the scalability of our decentralized approach. As shown in Fig. 7a, centralized decision latency scales quadratically, becoming impractical beyond 64 nodes. This latency stems from a severe serial compute bottleneck: to match Plebiscito's placement quality, the central orchestrator must evaluate paths across an $N \times N$ adjacency matrix. In contrast, Plebiscito's convergence time scales near-linearly, resolving well under a second.

Fig. 7b explains this divergence. A centralized approach minimizes network traffic to $O(2N)$ messages (gathering telemetry and dispatching decisions) but suffers from serial processing. Plebiscito generates more total messages, but by fully parallelizing state evaluation and conflict resolution across all nodes, it completely bypasses the computational bottleneck. Consequently, it achieves vastly superior wall-clock times that remain well below our theoretical worst-case bounds, validating its readiness for real-time data center orchestration.

VIII. Conclusion

As distributed ML models continue to scale, the datacenter network has transitioned from a transparent conduit to a primary performance bottleneck. In this work, we demonstrated that resolving this bottleneck does not strictly require expensive, specialized network hardware. Instead, we introduced *Plebiscito*, a fully decentralized, software-driven orchestrator that elevates network bandwidth to a first-class scheduling constraint. By combining a practical Kubernetes-based architecture with the rigorous theoretical guarantees of an asynchronous max-consensus auction, Plebiscito empowers physical nodes to collaboratively bypass congested links and achieve a provable $(1 - 1/e)$ optimal placement. Our evaluations, highlighted by a $1.5\times$ reduction in Job Completion Times, elimination of serial compute bottlenecks, and seamless prototype integration, prove that migrating from centralized, compute-centric schedulers to a decentralized, topology-aware model is a highly effective and scalable strategy for maximizing throughput in modern AI clusters.

Acknowledgment

This work has been supported by NSF awards #2201536 and #2133407.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[2] Meta, "Watch: Meta's engineers on building network infrastructure for ai," November 2023, accessed: 2025-05-15.

[3] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik, "Scaling distributed machine learning with In-Network aggregation," in *NSDI 21*. USENIX, Apr. 2021.

[4] 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 *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 485–500.

[5] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*. Boston, MA: USENIX Association, Mar. 2011.

[6] 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 Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 595–610.

[7] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia, "AntMan: Dynamic scaling on GPU clusters for deep learning," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 533–548.

[8] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. Swift, "ATP: In-network aggregation for multi-tenant learning," in *NSDI 21*. USENIX, Apr. 2021, pp. 741–761.

[9] S. Rajasekaran, M. Ghobadi, and A. Akella, "Cassini: Network-aware job scheduling in machine learning clusters," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024, pp. 1403–1420.

[10] J. Cao, Y. Guan, K. Qian, J. Gao, W. Xiao, J. Dong, B. Fu, D. Cai, and E. Zhai, "Crux: Gpu-efficient communication scheduling for deep learning training," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1–15.

[11] F. Esposito, D. D. Paola, and I. Matta, "On distributed virtual network embedding with guarantees," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, p. 569–582, Feb. 2016. [Online]. Available: https://doi.org/10.1109/TNET.2014.2375826

[12] E. A. Brewer, "Kubernetes and the path to cloud native," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 167.

[13] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: yet another resource negotiator," in *Proc. of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. ACM, 2013.

[14] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of Large-Scale Multi-Tenant GPU clusters for DNN training workloads," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX, Jul. 2019.

[15] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging AI applications," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 561–577.

[16] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A Low-Latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 613–627.

[17] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 945–960.

[18] Q. Weng, L. Yang, Y. Yu, W. Wang, X. Tang, G. Yang, and L. Zhang, "Beware of fragmentation: Scheduling gpu sharing workloads with fragmentation gradient descent," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 995–1008.

[19] 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 Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 289–304.

[20] Y. Zhao, Y. Liu, Y. Peng, Y. Zhu, X. Liu, and X. Jin, "Multi-resource interleaving for deep learning training," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 428–440.

[21] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. ACM, 2018.

[22] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters." USENIX Association, Nov. 2020, pp. 463–479.

[23] F. Esposito, I. Matta, and Y. Wang, " VINEA: An Architecture for Virtual Network Embedding Policy Programmability ," *IEEE Transactions on Parallel & Distributed Systems*, vol. 27, no. 11, Nov. 2016.

[24] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 912–926, 2009.

[25] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *OSDI 14*. USENIX, Oct. 2014.

[26] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 117–124, 2009.

[27] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge: Cambridge University Press, 2004.

[28] M. L. Fisher, R. Jaikumar, and L. N. Van Wassenhove, "A multiplier adjustment method for the generalized assignment problem," *Management Science*, vol. 32, no. 9, pp. 1095–1103, 1986.

[29] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, p. 455–466, Aug. 2014.

[30] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions–i," *Mathematical Programming*, vol. 14, no. 1, pp. 265–294, 1978.

[31] S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Information Processing Letters*, vol. 70, no. 1, pp. 39–45, 1999.

[32] M. H. J. Saldanha and P. S. L. de Souza, "High performance algorithms for counting collisions and pairwise interactions," in *Computational Science – ICCS 2019*, J. M. F. Rodrigues, P. J. S. Cardoso, J. Monteiro, R. Lam, V. V. Krzhizhanovskaya, M. H. Lees, J. J. Dongarra, and P. M. Sloot, Eds. Cham: Springer International Publishing, 2019, pp. 182–196.

[33] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, "FABRIC: A national-scale programmable experimental network infrastructure," *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47.

[34] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[36] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[37] FRRouting Project, *FRRouting User Manual*, FRRouting Project, 2025.

[38] Alibaba, "Alibaba public traces," accessed: 2025-05-15, URL: https://github.com/alibaba/clusterdata/tree/master/cluster-trace-gpu-v2020.