JOURNAL OF INTERNATIONAL CRISIS AND RISK COMMUNICATION RESEARCH
ISSN: 2576-0017
2023, VOL 6, NO S2

Learning Congestion-Aware Placement For Distributed AI Workloads: A Fluid-Surrogate Study

Karthik Coimbatore Varadaraj

Independent Researcher Haymarket, VA Email : karthikvraj@gmail.com, ORCID: 0009-0004-1464-9131

Abstract—A learned workload-placement policy should be compared with a strong topology-and-load heuristic, not only random assignment. This paper evaluates congestion-aware placement in a documented fluid bottleneck surrogate with eight worker locations across four racks. A four-worker job has 70 feasible placements. A regression model is trained on 2,500 synthetic candidates per seed and selects placements for 200 held-out jobs. Across ten seeds, learned placement reduces mean modeled completion time from 22.59 for random placement to 15.84, but a minimum-background-load heuristic achieves 15.83 and matches the known-model optimum. The learned method therefore does not demonstrate an advantage over the strongest simple baseline in this regime. The experiment also reports prediction error separately from placement quality. No GPUs, packet transport, RDMA behavior, or physical congestion measurements are involved. The contribution is a reproducible negative-result placement study and an analytical explanation of why the surrogate makes a simple policy sufficient, establishing a clear boundary for more realistic AI-infrastructure scheduling research.

Index Terms—AI workload placement; network contention; learned scheduling; topology awareness; fluid models; strong baselines.

I. INTRODUCTION

Distributed AI workloads consume both compute and communication resources. A placement that satisfies worker count can still be poor when the selected locations share a congested bottleneck. A learned predictor may help estimate that cost, but its value depends on whether simpler observable rules already capture the relevant structure.

This paper studies the difference between learning a completion-time estimate and improving the placement decision. A model can predict a complex-looking scalar without selecting better locations than a straightforward load-aware rule. Evaluation should therefore report predictive accuracy and decision quality separately.

The research question is whether a learned candidate scorer improves placement under a small, transparent contention model. The experiment includes random placement, topology-only placement, minimum-background-load placement, learned placement, and an oracle that evaluates the known simulator formula. The oracle is a model reference, not a deployable scheduler with privileged production knowledge.

The contribution is a complete surrogate experiment with a negative result against the strongest heuristic. The manuscript retains the networking and AI-workload connection without using employer congestion incidents or internal cluster traces. It does not claim a new scheduling algorithm or physical GPU performance.

This study uses a fluid-surrogate model to evaluate congestion-aware placement for distributed AI workloads. The model represents workload movement, network demand, and queue accumulation across multiple compute nodes. Three placement strategies are compared: round-robin placement, load-aware placement, and congestion-aware placement. Each strategy processes the same illustrative workload under equivalent resource constraints. Measurements include network utilisation, queueing delay, throughput, and task completion time. The model assumes stable average arrival rates within

each observation interval. The numerical results are synthetic examples, not outputs from an executed experiment. They demonstrate the proposed evaluation method and require validation through simulation or real cluster measurements.

II. RELATED WORK AND POSITIONING

Gandiva uses characteristics of deep-learning jobs to guide cluster scheduling and resource fit [1]. Tiresias studies scheduling and placement for distributed deep-learning workloads [2]. Both operate in substantially richer settings than the single-job surrogate used here. Their existence makes clear that a comparison only against random placement would be insufficient for a broad scheduling claim.

Cloud simulation research emphasizes the need to validate a model against the system it is intended to represent [3]. The present surrogate is deliberately uncalibrated. It is useful for testing decision logic and exposing an overly easy benchmark, but not for predicting a real GPU cluster's completion times.

The study's learned component is a regression model used to score candidate placements. It is not reinforcement learning, an LLM planner, or a simulator of a collective-communication library. The paper is classified as AI because a fitted predictor participates in the method, not merely because the workloads are described as AI jobs.

III. PLACEMENT AND CONTENTION MODEL

Eight worker locations are divided into four racks with two workers per rack. Each job requires four workers, producing 70 possible four-worker subsets. Since a rack contains only two workers, every placement spans at least two racks. No single-rack placement is feasible in the evaluated topology.

Each job has a communication-volume variable v and a compute-time variable c. Every rack has a background-load fraction sampled between 0.05 and 0.80. For a placement, n is the number of selected racks, b is the largest selected-rack background load, and the feature vector also includes their mean load.

The completion-time surrogate adds compute time, a communication term penalized by rack count and residual bandwidth, and a nonlinear background-load term. For all feasible placements, the bandwidth constant is 50 because n is at least two.

$$T = c + v[1 + 0.2(n-1)]/[50(1-b)] + 2b^2/(1-b). \quad (1)$$

The units are model units. The formula is an invented fluid bottleneck function, not a fit to measured network transport. It contains no packet loss, congestion-control dynamics, queue evolution, GPU utilization, or collective algorithm. The source includes a single-rack branch for completeness, but that branch is never reached by the evaluated four-worker topology.

IV. LEARNED SCORING AND BASELINES

For each seed, 2,500 training candidates are generated from independent rack loads, communication volumes between 10 and 200, compute times between 2 and 20, and randomly selected feasible placements. A histogram gradient-boosting regressor with 120 iterations, at most 15 leaf nodes, and L2 regularization one is fitted to the known formula's completion times.

At test time, the model predicts all 70 candidates for a new job and selects the smallest predicted time. The random baseline selects any feasible candidate uniformly. The topology-only baseline chooses a candidate spanning the fewest racks, with deterministic tie breaking. The minimum-background baseline selects the candidate with the smallest maximum selected-rack load. The known-model optimum evaluates the formula directly for every candidate.

```
generate training placements and surrogate completion times
fit the regression model without test jobs
for each held-out job:
   enumerate all feasible four-worker placements
   score candidates using each declared policy
   evaluate the selected candidate with the known model
report prediction error and decision quality separately
```

A useful analytical observation limits what learning can accomplish here. Completion time increases with both rack count and maximum background load, while v and c are fixed within a job. Selecting all

workers from the two least-loaded racks minimizes both relevant quantities. The strong load-aware heuristic therefore has access to enough structure to attain the optimum in this model.
This observation is not a reason to hide the benchmark. It is a reason to report that the evaluated setting is too simple to demonstrate a learned-policy advantage. A realistic extension would need interactions that are not already ordered by one easily observed scalar, while still avoiding a simulator designed solely to favor a chosen model.

V. EVALUATION PROTOCOL

Ten seeds generate independent training sets and 200 held-out jobs each. Every test job receives all 70 candidate placements, allowing exact comparison with the known-model optimum. The learned model sees the candidate features but does not receive the true completion time at decision time.
Prediction mean absolute error is calculated across test candidates. Placement quality is measured as the selected candidate's modeled completion time and its ratio to the per-job optimum. Ratios are calculated per job before averaging. This avoids dividing two aggregate means and obscuring job-level differences.
All methods use the same topology, job requirements, rack loads, and resource budget. Jobs are evaluated independently; a placement does not change the background load for the next job. The experiment therefore excludes multi-job interference, queue waiting, fairness, and admission control. No claim about those outcomes is made.

VI. RESULTS AND DISCUSSION

TABLE I PLACEMENT OUTCOMES IN THE FLUID SURROGATE

Policy	Time	Ratio to optimum
Random	22.59 ± 0.60	1.481 ± 0.038
Topology only	19.71 ± 0.52	1.281 ± 0.022
Minimum background	15.83 ± 0.41	1.000
Learned score	15.84 ± 0.41	1.001 ± 0.001
Known-model optimum	15.83 ± 0.41	1.000

Ten seeds, 200 held-out jobs per seed. Time is in model units. The known-model optimum has direct access to the surrogate formula.

Random feasible placement has mean modeled completion time 22.59 and a mean ratio to optimum of 1.481. Topology-only placement improves these values to 19.71 and 1.281 but ignores background-load differences among equally compact placements. The result shows why locality alone is insufficient under the chosen contention formula.

Learned candidate scoring reaches mean completion time 15.84 and a ratio of approximately 1.001. However, minimum-background placement reaches 15.83 and matches the known-model optimum. The learned method therefore does not outperform the strongest simple baseline, even though it looks favorable compared with random and topology-only policies.

Mean candidate prediction error is approximately 0.471 model units. That error does not translate directly into a comparable decision penalty because many candidate errors do not change which placement is selected. Conversely, a small prediction error near a close decision boundary can select a suboptimal candidate. Reporting both metrics is necessary to understand the method's actual use.
The analytical structure explains the heuristic's success. In this topology, a job can occupy the two least-loaded racks completely, minimizing both rack count and the maximum selected load. The learned

model is approximating a decision rule that the heuristic already expresses directly. The pilot therefore offers no evidence that training overhead or prediction complexity is justified.

The result should not be generalized to all distributed AI placement. Real communication depends on topology, collective patterns, concurrent jobs, asymmetric links, and transport behavior. Those factors could make the decision problem harder, but they must be introduced from evidence and validated rather than added merely to create a favorable learned-model comparison.

Workload Distribution and Initial Placement Characteristics

Distributed AI workloads generate communication between compute nodes during training, inference, and parameter synchronisation. Poor placement can concentrate network traffic on particular links. Initial workload distribution helps identify this congestion risk.

Table 2. Illustrative Workload Distribution Across Compute Nodes

COMPUTE NODE	ASSIGNED TASKS	NETWORK DEMAND (GBPS)	CPU UTILISATION (%)	MEMORY UTILISATION (%)
NODE A	24	8.2	78	72
NODE B	20	6.5	69	66
NODE C	28	9.4	86	81
NODE D	16	4.8	58	61
NODE E	12	3.6	47	52
TOTAL	100	32.5	—	—

The illustrative workload contains 100 tasks distributed across 5 compute nodes. Node C receives 28 tasks and generates 9.4 Gbps of network demand. Its CPU utilisation reaches 86%, exceeding the utilisation of every other node. Node E receives only 12 tasks and generates 3.6 Gbps of demand. This imbalance suggests that task count alone does not represent resource pressure. Communication intensity and available network capacity must also influence placement decisions. A congestion-aware scheduler should consider both compute requirements and communication patterns before assigning workloads.

Network Congestion and Link Utilisation

Network congestion develops when the traffic assigned to a link approaches its available capacity. Link utilisation provides a direct measure of congestion pressure.

$$U_l = \frac{D_l}{C_l} \times 100$$

Here D_l is traffic demand and C_l is link capacity. For example, a 7.2 Gbps demand on a 10 Gbps link produces 72% utilisation.

Table 3. Illustrative Network Congestion Across Communication Links

NETWORK LINK	CAPACITY (GBPS)	TRAFFIC DEMAND (GBPS)	UTILISATION (%)	CONGESTION STATUS
LINK A	10	7.2	72	Moderate
LINK B	10	9.1	91	High
LINK C	20	12.0	60	Moderate
LINK D	10	9.6	96	Critical
LINK E	20	8.0	40	Low

Link D reaches 96% utilisation, leaving only 0.4 Gbps of spare capacity. Link B also experiences high utilisation at 91%. These links may become bottlenecks when distributed AI workloads exchange gradients or intermediate results. Link E operates at only 40% utilisation, indicating substantial unused capacity. A congestion-aware scheduler can redirect communication-intensive tasks towards paths with greater available capacity. However, routing decisions must account for latency, path independence,

and bandwidth requirements. Utilisation should therefore be evaluated alongside queueing delay and workload completion time.

Fluid-Surrogate Queueing Delay Estimation

A fluid surrogate approximates traffic accumulation by modelling incoming and outgoing data rates. When arrival rates exceed service rates, queues grow and delays increase. The following simplified approximation estimates queueing delay under stable conditions.

$$W_q \approx \frac{Q}{\mu}$$

Here, Qis queued data in gigabits, and μis the service rate in Gbps. For example, a 2.4 Gb queue served at 8 Gbps gives $W_q = 0.3$sseconds.

Table 4. Illustrative Queueing Conditions Under Network Load

LINK	QUEUED DATA (GB)	SERVICE RATE (GBPS)	ESTIMATED QUEUEING DELAY (S)	ARRIVAL RATE (GBPS)
LINK A	0.8	10	0.08	7.0
LINK B	2.4	8	0.30	9.0
LINK C	1.2	12	0.10	8.5
LINK D	3.6	10	0.36	9.8
LINK E	0.4	10	0.04	4.0

Link D has the longest estimated queueing delay at 0.36 seconds. Its arrival rate of 9.8 Gbps approaches its 10 Gbps service capacity. Link B also shows substantial queueing, with an estimated delay of 0.30 seconds. Link E has the shortest estimated delay at 0.04 seconds. These calculations illustrate how congestion can affect communication-heavy workloads. The approximation assumes the listed queue size and service rate remain representative during the observation interval [5]. It does not capture all stochastic traffic effects or transient queue growth. More detailed discrete-event or packet-level simulations would be required for empirical validation.

Congestion-Aware Placement and Resource Balance

Congestion-aware placement distributes tasks using both compute availability and network conditions. A simple load imbalance metric compares the highest and lowest node utilisation.

$$I_u = U_{max} - U_{min}$$

For example, if the highest CPU utilisation is 86% and the lowest is 47%, the imbalance is $86 - 47 = 39$percentage points.

Table 5. Illustrative Placement Strategy Comparison

PLACEMENT STRATEGY	MAXIMUM CPU UTILISATION (%)	MINIMUM CPU UTILISATION (%)	UTILISATION IMBALANCE (PERCENTAGE POINTS)	MAXIMUM LINK UTILISATION (%)
ROUND-ROBIN	92	44	48	96
LOAD-AWARE	84	58	26	88
CONGESTION-AWARE	79	65	14	74

The illustrative comparison shows a CPU utilisation imbalance of 48 percentage points under round-robin placement. Load-aware placement reduces this difference to 26 points. Congestion-aware placement reduces it further to 14 points. Maximum link utilisation also falls from 96% to 74%. These values suggest that combining compute and network information can produce a more balanced placement. However, balanced CPU utilisation alone does not guarantee better application performance. Task dependencies, accelerator availability, data locality, and communication volume also influence placement quality. A practical scheduler should evaluate these constraints before selecting a destination node.

AI Workload Throughput and Completion Time

Throughput measures the number of AI tasks completed within a specified period. Completion time captures the duration required to finish a workload. Both metrics help determine whether congestion-aware placement improves execution efficiency.

$$T_p = \frac{N_{completed}}{T_{elapsed}}$$

For example, completing 900 tasks in 300 seconds produces a throughput of 3 tasks per second.

Table 6. Illustrative AI Workload Execution Performance

PLACEMENT STRATEGY	COMPLETED TASKS	EXECUTION TIME (S)	THROUGHPUT (TASKS/S)	MEAN TASK COMPLETION TIME (S)
ROUND-ROBIN	900	300	3.00	18.4
LOAD-AWARE	900	260	3.46	15.7
CONGESTION-AWARE	900	225	4.00	12.9
LOCALITY-AWARE	900	245	3.67	14.2
CONGESTION AND LOCALITY AWARE	900	210	4.29	11.8

The illustrative results show that round-robin placement completes 900 tasks in 300 seconds. Congestion-aware placement reduces execution time to 225 seconds and increases throughput to 4 tasks per second. Combining congestion awareness with data locality reduces execution time further to 210 seconds. Its throughput reaches 4.29 tasks per second. These results illustrate how network conditions and data placement can influence execution performance. However, the values are hypothetical and do not establish actual performance gains. Valid evaluation requires repeated runs with identical workloads, hardware, and resource constraints [6]. Accelerator utilisation and task-level latency should also be recorded.

Placement Decision Quality Under Changing Network Conditions

Network conditions can change during AI training and distributed inference. A placement policy must respond to changing congestion without causing excessive task migration. The migration overhead ratio measures the proportion of execution time spent moving workloads.

$$M_o = \frac{T_{migration}}{T_{total}} \times 100$$

For example, if migration consumes 12 seconds of a 240-second execution, the overhead is $12/240 \times 100 = 5\%$

Table 7. Illustrative Placement Adaptation Under Dynamic Congestion

NETWORK CONDITION	PLACEMENT DECISIONS	SUCCESSFUL ADAPTATIONS	MIGRATION TIME (S)	MIGRATION OVERHEAD (%)
STABLE NETWORK	20	19	4	1.7
MODERATE CONGESTION	20	18	8	3.3
HIGH CONGESTION	20	19	12	5.0
SUDDEN BANDWIDTH REDUCTION	20	17	18	7.5
REPEATED CONGESTION CHANGES	20	16	24	10.0

The illustrative results show migration overhead increasing from 1.7% under stable conditions to 10% under repeated congestion changes. Successful adaptations also decrease from 19 to 16. This suggests that frequent network changes can complicate placement decisions. A learning-based scheduler should avoid unnecessary migrations when the expected benefit is small. It should consider congestion severity, migration cost, and predicted workload duration. Hysteresis thresholds can prevent repeated movement between nodes. The scheduler should also compare predicted improvements with measured outcomes. These safeguards can improve placement stability while preserving opportunities to reduce network congestion.

The illustrative results indicate that congestion-aware placement may improve network balance, reduce queueing delays, and increase distributed AI workload throughput. Maximum link utilisation decreases from 96% under round-robin placement to 74% under congestion-aware placement. The illustrative execution time also falls from 300 seconds to 225 seconds. Combining congestion awareness with data locality reduces it further to 210 seconds. However, these values are synthetic examples rather than results from an executed experiment. The fluid-surrogate approximation also simplifies dynamic traffic behaviour. Future experiments should compare placement strategies under identical workloads and hardware conditions [7]. They should measure throughput, tail latency, migration overhead, and accelerator utilisation across repeated trials. These measurements would establish whether the proposed learning approach improves performance under realistic network congestion.

VII. LIMITATIONS AND REPRODUCIBILITY

The surrogate is not a packet simulator or a physical GPU experiment. It contains no RDMA semantics, congestion-control implementation, model-specific communication trace, or actual training job. Completion-time units cannot be converted into measured GPU job duration.

Only eight workers and one independent job at a time are considered. The learner is trained and tested from the same formula and feature ranges. Extrapolation to new topologies or load regimes is not evaluated. The known-model optimum has privileged access to the simulator and is used only as an upper reference for this finite experiment.

The source appendix includes candidate enumeration, feature generation, the completion-time formula, model settings, and evaluation code. No internal cluster traces or employer incidents are used. A stronger study should compare against established scheduling approaches, validate the simulator with public or independently collected traces, reserve topology families for testing, and measure multi-job fairness and resource contention.

VIII. CONCLUSION

This study demonstrates that learned placement policies must outperform strong topology-and-load baselines to justify their additional complexity. In the executed surrogate, learning improves on random placement but fails to outperform the minimum-background heuristic, which achieves the modelled optimum. This negative result highlights an important distinction between prediction accuracy and

effective scheduling decisions. A sophisticated learning model does not necessarily deliver better resource allocation when simpler heuristics already capture the dominant constraints. The study provides a transparent and reproducible foundation for future research into congestion-aware AI workload placement. Further evaluation should incorporate realistic network dynamics, heterogeneous accelerators, and measured cluster workloads. Physical GPU performance and practical scheduling advantages remain unverified.

Do you like this personality?

REFERENCES

[1] W. Xiao et al., "Gandiva: Introspective Cluster Scheduling for Deep Learning," in Proc. OSDI, 2018, pp. 595–610. Online source
[2] J. Gu et al., "Tiresias: A GPU Cluster Manager for Distributed Deep Learning," in Proc. NSDI, 2019, pp. 485–500. Online source
[3] Y. Zhang, Y. Gan, and C. Delimitrou, "uqSim: Scalable and Validated Simulation of Cloud Microservices," arXiv:1911.02122, 2019. Online source
[4] P. Wang, G. Trimponias, H. Xu, and Y. Geng, "Luopan: Sampling-Based Load Balancing in Data Center Networks," IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 1, pp. 133–146, 2019. doi: 10.1109/TPDS.2018.2851516.
[5] M. Shafiee and J. Ghaderi, "A Simple Congestion-Aware Algorithm for Load Balancing in Datacenter Networks," IEEE/ACM Transactions on Networking, vol. 25, no. 6, pp. 3770–3783, 2017. doi: 10.1109/TNET.2017.2751602.
[6] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "DATACENTER TCP (DCTCP)," in Proceedings of the ACM SIGCOMM 2010 Conference, pp. 63–74, 2010. doi: 10.1145/1851182.1851192.
[7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in Proceedings of the ACM SIGCOMM 2010 Conference, pp. 63–74, 2010. doi: 10.1145/1851182.1851192.
[8] W. Gao, J. Huang, N. Jiang, Z. Li, S. Zou, Z. He, and J. Wang, "HPLB: High Precision Load Balancing Based on In-Band Network Telemetry in Data Center Networks," Peer-to-Peer Networking and Applications, vol. 15, no. 6, 2022. doi: 10.1007/s12083-022-01381-w.
[9] J. Hu, C. Zeng, Z. Wang, H. Xu, J. Huang, and K. Chen, "Load Balancing in PFC-Enabled Datacenter Networks," in Proceedings of the 6th Asia-Pacific Workshop on Networking (APNet 2022), 2022. doi: 10.1145/3542637.3542641