capstract:2611.00052v1 [cs.DC] 20 Nov 2026

# Size-Biased Admission in Print-Gated GPU Clusters

Mara Venn
Calden Institute for Computing, Somerville
MA, USA

## Abstract

A print-gated scheduler issues a placement request only after one allocation sheet per requested GPU has reached the printer's output tray. If a jam invalidates the partly printed ticket, wide gangs need more attempts and rejoin the spool more often. We derive the offered load of this admission stage and evaluate it by discrete-event replay on a 64-GPU cluster model. At a 7% per-feed jam probability, eight-GPU tickets discard 2.47 ejected sheets per admission. Compared with an otherwise identical jam-free printer, jams reduce the scheduler's 95th-percentile placement wait from 110 to 93 s while increasing the client's 95th-percentile wait from 121 to 561 s. GPU utilization differs by 0.35 percentage points. A 15 min interruption leaves the placement queue empty and all GPUs idle while an average of 174 requests wait in the spool. Fitting two accelerator records per sheet retains the reprint rule and lowers the client tail to 135 s. The client clock includes the delay at the print spool.

## CCS Concepts

• Computer systems organization → Cloud computing; • Software and its engineering → Scheduling.

## Keywords

GPU clusters, gang scheduling, admission control, queueing delay, print spooling

**Reference Format:**
Mara Venn. 2026. Size-Biased Admission in Print-Gated GPU Clusters. Preprint, November 20, 2026.

## 1 Introduction

A GPU scheduler normally timestamps a job when it enters the placement queue. In an installation that requires a printed allocation ticket, this timestamp can be substantially later than submission. Consider a shared cluster that keeps one tear-off sheet per requested accelerator for its allocation records. The spooler issues the placement request after the last sheet reaches the output tray. A paper jam delays the request, but the delay does not appear in the placement queue because the request has yet to be issued.

The distinction matters for gang jobs. An eight-GPU job needs a node that can accommodate the whole gang [2, 3]. Under a one-sheet-per-accelerator rule it also needs eight consecutive successful feeds. A jam invalidates the partly printed ticket. When cleared, the printer moves the interrupted request to the back of the spool, where it can be overtaken by requests with fewer pages. Thus a printer can impose a size preference without reading a job's priority, runtime, or placement constraints.

We analyze this form of admission under three questions. How much of the preference comes from losing the prefix of a ticket, as opposed to the time needed to clear a jam? What does a queue monitor report when admission becomes the bottleneck? Can a change to the print layout relieve the bottleneck while preserving one physical record for each device? We answer these questions with a service-demand model and a discrete-event replay. An interrupted feed changes the order in which the scheduler sees jobs, and separating the spool from the scheduler changes the measured queueing tail even when the overall fraction of busy GPUs changes little.

## 2 Print-Gated Admission

### 2.1 Grant publication

A submission carries a request ID and a gang width $g$. The print ticket consists of $g$ sheets, each bearing the request ID, the print attempt number, and a distinct slot ordinal. The IDs name requested slots. Physical device IDs are assigned later, when the cluster has room for the gang. The subscriber at the scheduler receives a grant only after ejection of every sheet in the attempt. Until then, the spool retains the request and no device is held for it.

The Internet Printing Protocol defines per-job completion counters for media sheets [7]. Our gate assumes an ejection record with a request ID and sequence number. For admission, a last page counter by itself is insufficient. On restart, a complete set of slot ordinals from an active attempt is required. A partially ejected set is void, even if it remains in the tray. The spool retains the logical ID across retries. The scheduler consumes the grant at most once per logical ID, so repeating notification after ejection cannot admit a second gang. The evaluation treats spool state and admission publication as reliable and models neither crashes of the spool host nor duplicate messages.

If a feed jams, the engine is blocked during clearance and processes no other request. At the end of clearance, it appends the interrupted job at the tail with a fresh attempt number. Already admitted jobs continue to execute. When sheets are unavailable, the engine instead waits for stock and resumes the current attempt. The scheduler can therefore distinguish an empty placement queue from an empty system only if it can observe the spool.

### 2.2 Printer work per request

Let $m$ be the number of sheets required for one ticket, $p$ the independent probability that a feed jams, and $q = 1 - p$. For the default layout $m = g$. A complete first attempt has probability $q^m$. With eventual retry, the number of attempts is geometric with mean $q^{-m}$. This counts attempts per request regardless of intervening requests. The wall time between attempts depends on where the request rejoins the queue.

Printing a valid ticket requires a run of $m$ non-jammed feeds for that request. Let $N_m$ count the feeds expended across all its attempts, including the feeds ending in a jam. Conditioning on the next feed at each prefix length, with a jam resetting the prefix to zero, gives

$$\mathbb{E}[N_m] = \frac{q^{-m} - 1}{p}. \tag{1}$$

The limit at $p = 0$ is $m$. Each feed takes $s$ seconds, and each jam adds a clearance of $c$ seconds to the engine's occupied time. The expected occupied time per ticket is $(s + pc)\mathbb{E}[N_m]$. The expected number of ejected sheets discarded before a valid ticket is $q\mathbb{E}[N_m] - m$. A feed that jams does not count as an ejected sheet.

For job arrival rate $\lambda$ and width probabilities $\pi_g$, the offered printer load is

$$\rho_P = \lambda(s + pc) \sum_g \pi_g \mathbb{E}[N_{m(g)}]. \tag{2}$$

This expression accounts for retries even when the cluster has spare devices. If $\rho_P > 1$, a single printer cannot issue grants at the offered rate in the long run. The scheduler does not backpressure the printer, so the threshold is independent of the number of free devices. This calculation predicts the location of the printer bottleneck. It does not predict the distribution of wait times at either queue.

### 2.3 Recovery and layout controls

We compare the default policy, **Restart**, with five controls. **Digital** publishes immediately and incurs no print time. **Jam-free** runs the default one-sheet-per-GPU layout with $p = 0$. This controls for the time required to print a complete ticket. **Head restart** discards the same prefix as Restart, but holds the head of the spool until it completes a ticket. It has the same expected service demand as Restart and admits jobs in submission order.

**Resume** also holds the head of the spool, preserves each ejected sheet, and retries only the jammed feed, with the same clearance time. Its expected feed count is $m/q$. This policy requires a reliable record of which ordinals were ejected before clearance.

**Two-up** prints at most two adjacent slot records per page. The two records carry their own ordinals and can be separated after ejection. Here $m = \lceil g/2 \rceil$, and a jam still discards the whole prefix and moves the job to the tail. The engine's ejection time remains $s$ because both records are printed on the same face. The two-up control assumes that the custodian accepts cut sections as individual records. Sites requiring a full page for a slot would have to use a different policy.

## 3 Evaluation Method

### 3.1 Workload and replay

We model eight homogeneous servers with eight GPUs each. A job occupies all requested accelerators on one node for its entire runtime. At grant issuance and job completion, the scheduler scans the admitted queue in order. It places each job on the node with the fewest free GPUs that can still hold the gang. A job that cannot fit remains in the queue while subsequent jobs can start. The scheduler makes no reservations for skipped jobs and does not use predicted runtimes. This placement rule and all runtimes are held fixed across policies.

Large GPU traces document gang and locality constraints [3]. We use a synthetic workload rather than estimate print demand from those traces, which contain no such field. Widths are independent draws of 1, 2, 4, or 8 GPUs with probabilities 0.58, 0.22, 0.12, and 0.08. The corresponding mean runtimes are 90, 120, 160, and 240 s. Within each class we draw lognormal runtimes with a log-scale standard deviation of 0.65. The mean job uses 335.4 GPU-s. For nominal GPU load $\rho_G$, we set $\lambda = 64\rho_G/335.4$ jobs/s. Arrivals follow a Poisson process and continue during a jam, so the simulated client does not reduce load in response to delayed grants [6].

The reference configuration uses 1.5 s per feed, 12 s of clearance per jam, $p = 0.07$, and $\rho_G = 0.82$. These are controlled settings, independent of any particular printer model. We sweep jam probability, clearance time, and nominal GPU load independently. An additional replay interrupts feed service for 15 min at hour 4. A feed that would overlap this interruption is deferred until service returns. Spool state and already ejected sheets survive this interruption. In each control and outage pair, we use the same workload and the same stream of jam draws by feed order.

### 3.2 Measurements

We run 32 independent traces per setting, with workload seeds 101 through 132 and a separate jam stream seeded with the workload seed plus 10,000. Each trace starts empty. We measure jobs submitted from hour 2 up to hour 8, keep generating arrivals for one further hour, and then drain both queues. All 108,104 submissions in this window at the reference load are included in each policy's latency estimates. Of these, 8,714 request eight GPUs. At settings for which printer demand exceeds capacity, this procedure yields a finite-replay delay, and comparisons at those settings depend on run duration.

The client clock starts at submission and the placement clock starts at grant publication. Both stop when the gang starts. We report the mean across traces of the within-trace $p_{95}$ computed with linear interpolation. A printed job's spool time is the difference between its two wait measurements. Quantiles of these components are computed separately. We compute paired 95% percentile bootstrap intervals from 5,000 resamples of trace-level differences. A submission counts as overtaken if at least one later submission receives its grant before it does. We determine this using grant order, regardless of the order in which the jobs start.

GPU utilization is busy GPU time divided by capacity during the six-hour measurement window, including work that started before the window. The outage replay additionally samples both queue lengths and busy GPU counts every 30 s. A ticket currently being printed or cleared counts toward the spool length.

# 4 Results

## 4.1 Paper loss scales with gang width

Table 1 compares the model and replay at the reference settings. The first-attempt success probability falls from 0.93 at width one to 0.56 at width eight. The observed eight-GPU request uses 1.79 attempts and discards 2.47 ejected sheets on average. The waste is concentrated in the largest gangs. They account for 8.1% of submissions and about 73% of all ejected sheets later voided. A failed one-sheet attempt has no prefix to discard.

**Table 1: Analytical and replayed ticket costs. $P_1$ is the analytical probability of completing the first attempt. Attempts and discarded sheets are replay means by width.**

| GPUs | $P_1$ | Attempts | Discarded sheets |
|---|---|---|---|
| 1 | .930 | 1.073 | 0.000 |
| 2 | .865 | 1.153 | 0.072 |
| 4 | .748 | 1.347 | 0.482 |
| 8 | .560 | 1.792 | 2.465 |

The model puts reference printer load at 94.9%. Setting clearance to zero keeps the width-dependent retry rule and leaves the spool charged for failed feeds. At this setting the overall client $p_{95}$ is 128.4 s. Increasing clearance to 4, 8, and 12 s raises it to 147.4, 208.0, and 560.6 s, respectively. Clearance time is felt by every job behind a jam, including jobs whose own ticket would print successfully on the first attempt.

Tail re-entry changes who pays this delay. Under Restart, a later submission receives its grant first for 6.8% of one-GPU jobs and 43.7% of eight-GPU jobs, averaging the proportions across traces. Head restart eliminates this overtaking while leaving the same sheets subject to loss. Returning the failed ticket to the tail adds 455.0 s to the wide-gang client $p_{95}$ (95% CI 290.3 to 650.6 s). The corresponding increase for all widths is 54.3 s (95% CI -6.7 to 125.5 s). Most submissions use one or two GPUs, so the aggregate delay obscures the penalty at width eight.

## 4.2 Jams improve the reported placement tail

At the reference point, Restart lowers the placement $p_{95}$ by 17.0 s against Jam-free (95% CI 2.0 to 33.4 s lower). The client $p_{95}$ rises by 439.4 s (95% CI 332.9 to 565.5 s higher). For eight-GPU gangs, the two effects are larger: the placement tail falls by 69.8 s (95% CI 22.4 to 120.3 s lower) and the client tail rises by 674.8 s (95% CI 480.9 to 903.2 s higher). Table 2 gives the absolute waits and utilization. The utilization difference between Restart and Jam-free is -0.35 percentage points, with a paired 95% interval of -0.78 to 0.02 points.

**Table 2: Mean per-trace 95th-percentile waits, in seconds. P measures from placement-queue entry, C from client submission. Util. is cluster-wide GPU utilization (%).**

| Policy | All P | All C | 8-GPU P | 8-GPU C | Util. |
|---|---|---|---|---|---|
| Digital | 109.9 | 109.9 | 427.5 | 427.5 | 82.4 |
| Jam-free | 110.3 | 121.2 | 418.1 | 434.5 | 82.4 |
| Restart | 93.3 | 560.6 | 348.3 | 1,109.3 | 82.0 |
| Head restart | 102.8 | 506.3 | 327.8 | 654.2 | 82.3 |
| Resume | 107.4 | 202.2 | 388.3 | 466.1 | 82.4 |
| Two-up | 116.2 | 134.7 | 417.3 | 438.8 | 82.4 |

Figure 1 traces the effect as faults become more frequent. At $p = 0.08$, modeled printer load exceeds 100% and the overall client $p_{95}$ reaches 1,758.5 s in the finite replay. The placement tail is 69.8 s, the lowest in the sweep. At $p = 0.07$, the same split appears at high GPU demand. Moving from $\rho_G = 0.62$ to 0.90 raises Restart's client tail from 107.4 to 2,073.5 s and crosses the printer's capacity threshold. At load 0.90, bypassing the printer yields 296.5 s on both clocks, more than twice the placement wait recorded under Restart (146.5 s).

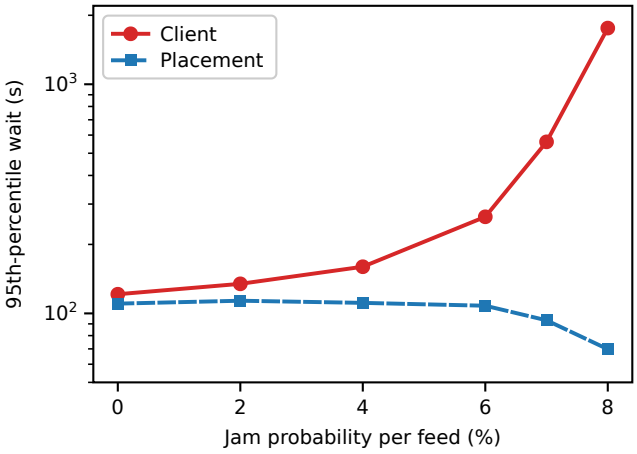



**Figure 1: Jam-probability sweep under Restart at 82% nominal GPU load. Each point averages 32 within-trace 95th percentiles. At 8% jams the spool is overloaded and the client estimate corresponds to a six-hour submission window.**

## 4.3 Changing the page layout

Resume lowers the reference client tail from 560.6 to 202.2 s. The reduction is 358.3 s (95% CI 258.5 to 480.6 s). By retaining every ejected sheet, the policy reduces offered printer load to 84.3%. The client tail under Head restart is 506.3 s with the same first-in, first-out grant order as Resume. Holding the ticket at the head is insufficient on its own when a jam still wastes the whole prefix.

Two-up retains tail re-entry, reduces eight-GPU tickets to four sheets, and leaves the one-GPU ticket unchanged. At $p = 0.07$, it raises predicted first-attempt success at width eight from 0.560 to 0.748; the replayed mean number of attempts falls from 1.79 to

1.33. Offered printer load drops to 55.4%, and the overall client tail falls to 134.7 s. Placement wait rises to 116.2 s. The cluster receives more of the waiting work directly. A dashboard that only shows placement delay would favor Restart over both recovery options.

### 4.4 A paper-out incident

Figure 2 interrupts printing with both queues already active. Five minutes after interruption, the placement backlog has emptied in every treated trace. The spool has grown from a mean of 31.7 to 80.2 requests, while the mean busy GPU count has fallen from 53.1 to 11.8. By minute 15, all 64 GPUs are idle in every treated trace and the spool holds a mean of 174.2 requests. The control has 54.0 busy GPUs and a mean spool length of 31.2 at the same time.

Twenty minutes after replenishment the treated cluster is using 54.2 GPUs on average, against 52.4 without an interruption, but the mean spool still holds 150.3 requests against 36.7. At the reference workload the long-run printer service budget is 9.89 tickets/min against 9.39 incoming requests/min. That margin is too narrow for the return of busy GPUs to imply that the spool has recovered. A single status panel tracking only placement and utilization shows the beginning and end of a capacity incident on a different schedule from the one experienced by clients.

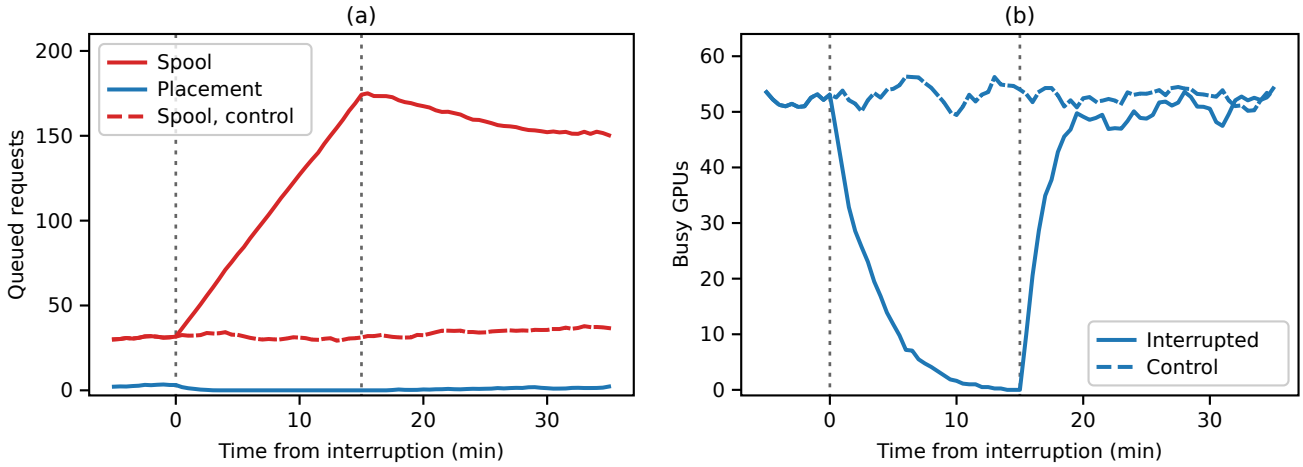



**Figure 2: Response to a 15 min interruption in feed service. Samples at 30 s intervals are means across 32 paired replays. Panel (a) counts jobs waiting at the printer, including the active ticket, and jobs waiting for placement. Dashed lines show uninterrupted controls. Vertical rules delimit the interruption.**

## 5 Related Work

Tiresias uses attained service and placement decisions to reduce job completion time in GPU clusters [2]. Gandiva time-slices and migrates jobs to improve assignment fit [8], and Pollux changes both job configurations and cluster allocations to improve goodput [5]. Themis explicitly considers how shared cluster service affects finish times [4]. In all these cases, the policy is evaluated over jobs known to the scheduler. Here the policy under study decides when a job becomes known, without modifying the scheduler's placement or time-sharing rules.

Systems work on latency tails gives reasons to measure at the caller [1]. A placement wait is meaningful for diagnosing slow placement after admission. It does not account for the backlog at an upstream service. The print gate makes this distinction observable because the arrival order at the placement queue is itself an outcome of per-sheet failures. The two clocks are well defined even when both queues eventually drain.

## 6 Discussion

The paper handling rule assigns a scope to the failure. Restart invalidates a whole gang ticket after one bad feed, whereas Resume limits redo to one sheet. This is separate from the question of whether a retry should take its old place in the spool. A production gate would need to log the request ID at submission, preserve that ID across attempts, and reconcile it against the grant log. Recording printer uptime or page throughput alone would not reveal which job widths are still waiting. Counting sheets instead of job IDs would further overweight wide tickets and repeated attempts.

This study uses one print engine, independent faults, and synthetic fixed-width jobs. A damaged roller can concentrate faults in time, and a shared office printer may also receive unrelated jobs. Neither effect appears in the model. We assume successful ejection is observed exactly and that restoration after a paper-out preserves the active ticket. We have no hardware measurements of clearance time, and the load threshold depends on that parameter. Our intervals describe variability across synthetic traces. They do not quantify uncertainty about print hardware or workloads in the field. An elastic scheduler that can resize jobs before grant publication would also change the distribution of ticket lengths. Resizing jobs after admission would leave the initial paper demand intact.

The cleanest operational measure is time from the first submission carrying a logical request ID. It includes tickets waiting

on clearance and requests that later require additional attempts. In the evaluated design a job becomes runnable when a complete ticket arrives, but it becomes a waiting job as soon as the spool accepts it. The two events should have separate timestamps even when both records are eventually kept in the same file.

## References


[1] Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. *Communications of the ACM* 56 , 2 (2013), 74–80. https://doi.org/10.1145/2408776.2408794

[2] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 485–500.

[3] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 947–960.

[4] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and Efficient GPU Cluster Scheduling. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 289–304.

[5] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. 2021. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 1–18.

[6] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. 2006. Open Versus Closed: A Cautionary Tale. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*..

[7] Michael Sweet and Ira McDonald. 2017. Internet Printing Protocol/1.1: Model and Semantics. RFC Editor.

[8] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 595–610.