THE UNIVERSITY OF OSAKA

Hetero-Par 2026 · Euro-Par Workshop
Pisa, Italy · August 2026

RCC-Preserving Online Resource Allocation for Heterogeneous GPU Sharing with Finite Availability Windows

Yuki Gennai[1], Yuichi Ohshita[2], Hideyuki Shimonishi[2]

[1]Graduate School of Information Science and Technology, The University of Osaka, Japan

[2]D3 Center, The University of Osaka, Japan

MOTIVATION

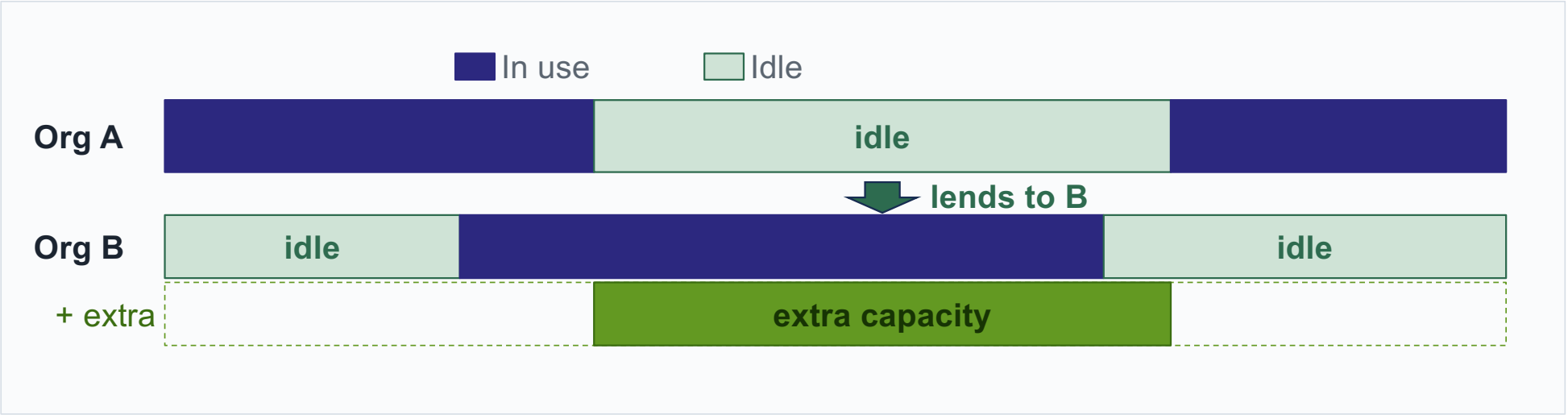
AI Demand Is Outpacing GPU Supply

- AI demand: growing faster than GPU supply
- Research institutions: pay for cloud, or size hardware for peak demand — both costly
- Result: GPU access becomes an economic bottleneck for research

MOTIVATION

Most Owned GPUs Sit Idle Most of the Time

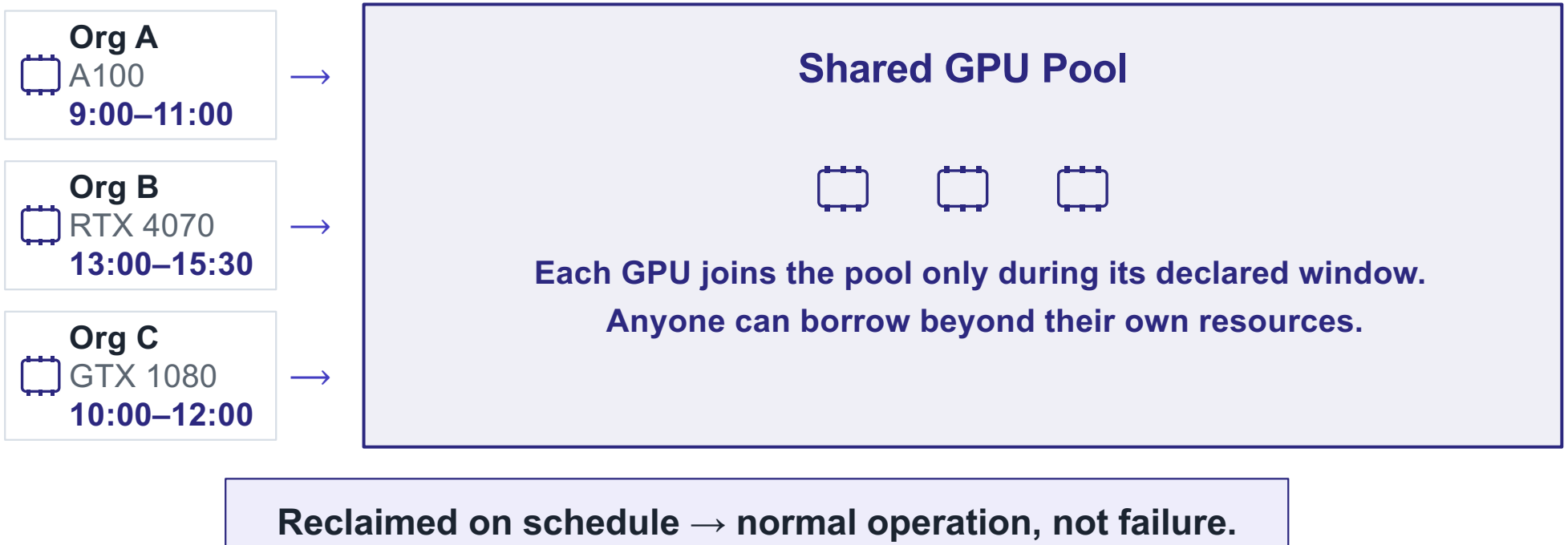
- **Reality:** own GPUs run at low utilization, long idle stretches
- **Ideal:** lend idle time to other orgs — more compute, no new hardware



idle time → shared with other organizations

OPERATING MODEL

Provider-Autonomous GPU Sharing: A Shared GPU Pool



Reclaimed on schedule → normal operation, not failure.

OPERATING MODEL

A Pool With Two Kinds of Heterogeneity

CONVENTIONAL

Compute & memory

different throughput, different memory

NEW

Temporal availability

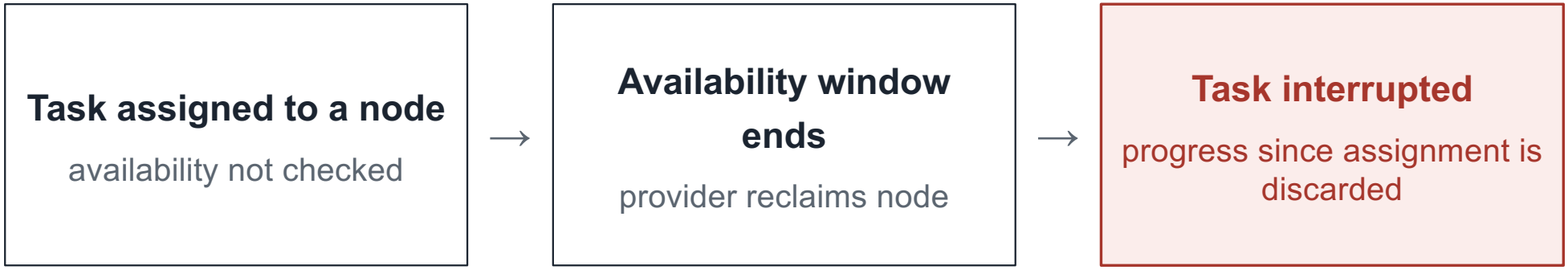
different lending window lengths

> Performance and memory, we know how to handle. Temporal availability is the new problem.

RELATED WORK — LIMITATION 1

Ignoring Availability Causes Mid-Execution Interruption

Gandiva[1], Tiresias[2]: handle compute/memory — **assume nodes stay available forever**

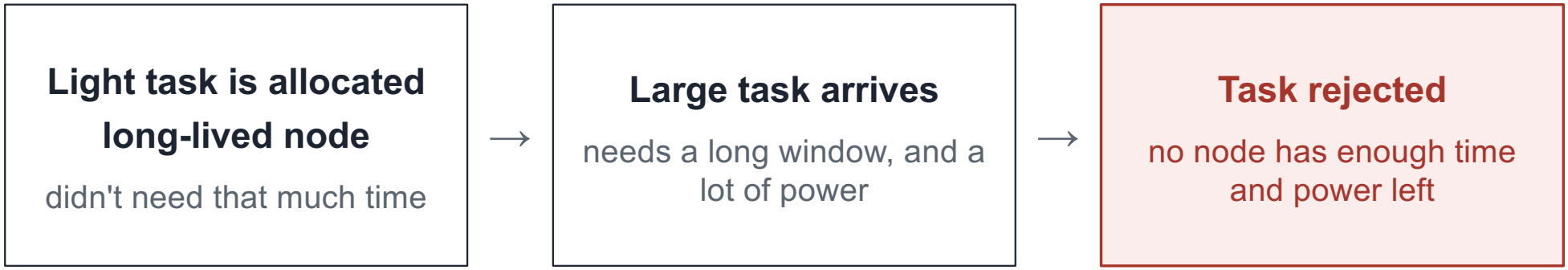


[1] Gandiva [Xiao+, OSDI'18] , [2] Tiresias [Gu+, NSDI'19]

RELATED WORK — LIMITATION 2

Treating Availability as Pass / Fail Wastes Value

Spot/Transient Computing[3], GPUnion[4]: check feasibility — but treat residual availability as pass/fail only, **never as a value**



RESEARCH OBJECTIVE

Two Requirements for This Environment

Goal: raise success rate. Jobs arrive online — decide job by job, with no global view.

1 Don't interrupt

Complete tasks — don't let a reclaimed node cut them off.

2 Don't over-spend capability

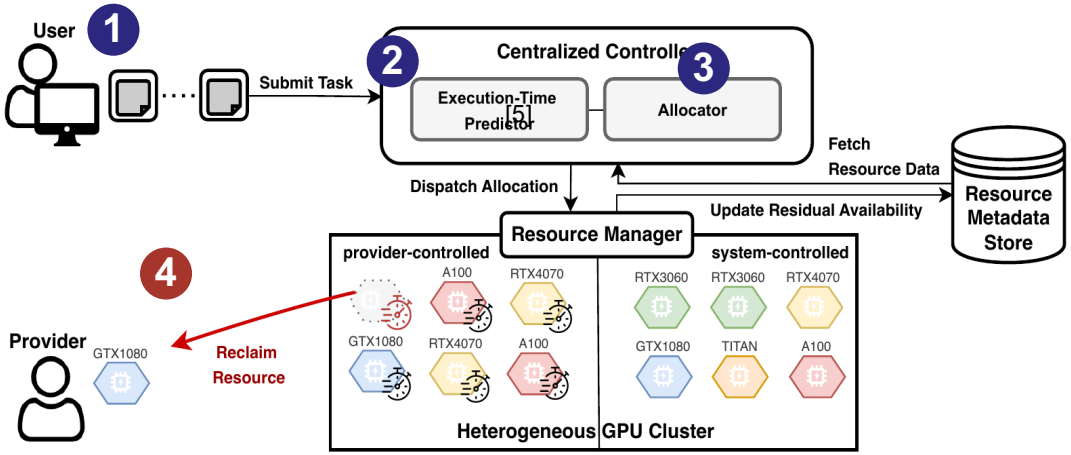
Future jobs unknown — treat lending time as part of a resource's capability. Don't waste it.

Formulate	Define	Propose
System model & problem setting	A new value metric	Value-preserving scheduling

SYSTEM MODEL

Centralized Controller Coordinates the Heterogeneous Cluster

Environment: one controller, one heterogeneous cluster — system-controlled nodes (**always available**) and provider-controlled nodes **(time-limited)**



1. User submits a task with a deadline
2. Controller predicts execution time
3. Controller dispatches the allocation, or rejects if no node fits
4. Provider reclaims the node on schedule

FORMULATION

The Optimization Problem, at Each Task

Decision variables $x_{j,k}$ — node & ring position · r_j — data share per node

OBJECTIVE Eq.1–2 · slide 11

$$\min \Sigma_{j,k}\, x_{j,k}\, C_j \quad \text{where} \quad C_j = w(V_j)\cdot P_j \cdot \tilde{L_j}$$

Minimize total RCC consumed — a node's residual compute budget

subject to —

$$\hat{T} \le D \;,\; \hat{T} \le \tilde{L_j}$$

Fits deadline and residual availability?

Eq.8–9 · slide 12

$$T = T_{init} + T_{comp} + T_{comm} + T_{fin}$$

Execution time model

Eq.3–6 · slide 13

Structural constraints slide 14

$$\Sigma_j x_{j,k} \le 1,\ \Sigma_k x_{j,k} \le 1 \quad \Sigma_j x_{j,k} \ge \Sigma_j x_{j,k+1} \quad r_j \le \Sigma_k x_{j,k},\ \Sigma_j r_j = 1 \quad \Sigma_k x_{j,k} \cdot V_{req} \le V_j$$

FORMULATION

Objective & A Node's Value: “Residual Computational Capacity (RCC)”

EQ. 2 — OBJECTIVE

$$\min \sum_{j \in \mathcal{J}} \sum_{k=1}^{K^{\max}} x_{j,k} C_j$$

Minimize the total RCC consumed by the current allocation

EQ. 1 — “RCC” DEFINITION

$$C_j = w(V_j) \cdot P_j \cdot \tilde{L}_j$$

The total computation a node can deliver before its availability window closes.

$w(V_j)$	P_j	$\tilde{L}_j$
Weight favoring more VRAM	Compute throughput [TFLOPS]	Time left before reclaim [sec]

long-running, tight-deadline, high-performance jobs can still be accepted later.

FORMULATION

Constraints: Deadline & Residual Availability

EQ. 8 — DEADLINE

$$\hat{T} \leq D$$

Finishes within the deadline?

Estimated finish time must not exceed user-specified deadline D.

EQ. 9 — AVAILABILITY

$$\hat{T} \leq \tilde{L}_j + M_{\text{big}}\left(1 - \sum_{k=1}^{K^{\max}} x_{j,k}\right)$$

Finishes within each assigned node's residual availability?

Estimated finish time must fit each selected node's residual availability. M_{big} cancels the constraint when a node isn't selected.

$\hat{T}$: the estimated finish time of task

FORMULATION

Execution Time: Four Components

$$T = T^{\text{init}} + T^{\text{comp}} + T^{\text{comm}} + T^{\text{fin}}$$

EQ. 3 — INIT

$$T^{\text{init}} = \sum_{j \in \mathcal{J}} \left(\frac{8\left(M \sum_k x_{j,k} + r_j N s^{\text{in}}\right)}{B_u^{\text{up}}} + \left(\sum_k x_{j,k}\right) \ell_{u,j} \right)$$

Initialization — transfer time for model & data

EQ. 4 — COMP

$$T^{\text{comp}} \geq \sum_{k=1}^{K^{\max}} x_{j,k}\, \theta_j^{\text{setup}} + E\, r_j N\, \theta_j^{\text{unit}}, \quad \forall j \in \mathcal{J}$$

Startup overhead + processing — slowest assigned node

EQ. 5 — COMM

$$T^{\text{comm}} = 2\tau \frac{EN}{b} \sum_{j,j'} \left(c_{j,j'} + \sum_k z_{j,j',k} \right) \left(\frac{8M}{B_{j'}^{\text{up}}} + \ell_{j,j'} \right)$$

Synchronization — training only (τ = 0 for inference)

EQ. 6 — FIN

$$T^{\text{fin}} = \frac{8}{B_u^{\text{down}}} \left(\tau M + (1-\tau) N s^{\text{out}} \right)$$

Result transfer back to user

T^{comp} is a prediction → add a safety margin for robustness:

$$\hat{T} = T^{\text{init}} + (1+\delta)\, T^{\text{comp}} + T^{\text{comm}} + T^{\text{fin}}$$

FORMULATION

Constraints: Assignment, Topology & Memory Constraints

Structural constraints

$$\Sigma_j\, x_{j,k} \le 1\ ,\ \ \Sigma_k\, x_{j,k} \le 1$$

Node position

Prevents double-assignment

$$\Sigma_j\, x_{j,k} \ge \Sigma_j\, x_{j,k+1}$$

Ring topology

Fills positions contiguously — no gaps in the ring.

$$r_j \le \Sigma_k\, x_{j,k}\ ,\ \ \Sigma_j\, r_j = 1$$

Data partition

Only selected nodes get data; all data is covered

$$\Sigma_k\, x_{j,k} \cdot V^{\mathrm{req}} \le V_j$$

VRAM

Rejects models that don't fit in memory

EVALUATION SETUP

Simulation Environment

- 35-node heterogeneous GPU cluster, 5 simulated days, 10 trials
- Performance & availability negatively correlated: high-end GPUs scarce, stay briefly; low-end GPUs abundant, stay long (Table 1)
- Workload: real Alibaba PAI traces; tasks classified Tier S / M / L by execution time
- Vary mix of lent idle-time vs. always-available nodes, 0–100% — sensitivity analysis of volatility

Table 1: GPU configuration and availability parameters used in the simulation

GPU	Perf. [TFLOPS]	VRAM [GB]	BW [Gbps]	Stay [hr]	Avail.
A100	77.97	40	10	8.0	0.5
Titan RTX	32.62	24	10	9.0	0.6
RTX 4070	29.15	12	1	10.0	0.7
RTX 3060 Ti	16.20	8	1	10.0	0.7
RTX 2080	20.14	8	1	12.0	0.8

Stay Longer

EVALUATION SETUP

COMPARED METHODS

Existing work: two ablations below — differ only in objective and availability constraint.

Method	Objective	Availability constraint
Avail-Agnostic	$\min\ T$	—
Avail-Aware	$\min\ T$	$\hat{T} \le \tilde{L}_j + M_{\text{big}}\left(1 - \sum_{k=1}^{K^{\max}} x_{j,k}\right)$
RCC-Preserving (proposed)	$\min \sum_{j \in \mathcal{J}} \sum_{k=1}^{K^{\max}} x_{j,k} C_j$	$\hat{T} \le \tilde{L}_j + M_{\text{big}}\left(1 - \sum_{k=1}^{K^{\max}} x_{j,k}\right)$

Constraint → does it cut interruption, raise success? （Avail-Agnostic→Avail-Aware）
RCC objective → does it raise success further? （Avail-Aware→RCC-Preserving）

EVALUATION SETUP

Evaluation Metrics

➢ Task Conditions

SUCCESS
Finishes by deadline

TARDY
Finishes, but after deadline

INTERRUPTED
Terminated by node departure

REJECTED
Infeasible at arrival

➢ Metrics

SUCCESS RATE
Share of tasks that met their deadline

$$N_{success} / N_{total} \times 100\%$$

WASTED COMPUTATION
Compute lost to interruption

$$\Sigma_{interrupted} P_j \cdot t^{elapsed}$$

TOTAL OCCUPANCY
Time allocated to any task — success or not

$$\Sigma_j t^{alloc} / (N \cdot T^{hor}) \times 100\%$$

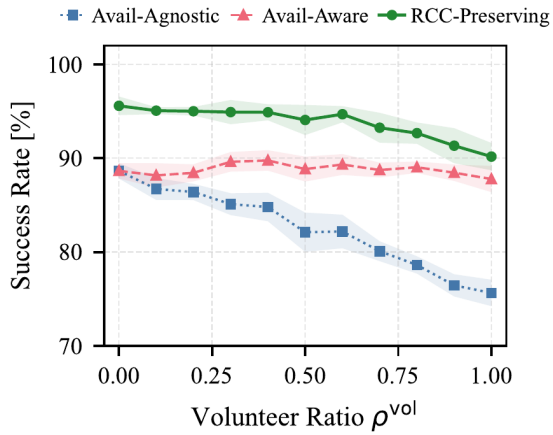
Symbols P_j — node j's throughput · t — elapsed / allocated time

RESULTS (1/5)

RCC-Preserving Leads on Success Rate

Valuing availability in the objective sustains the highest success rate at every volatility level.

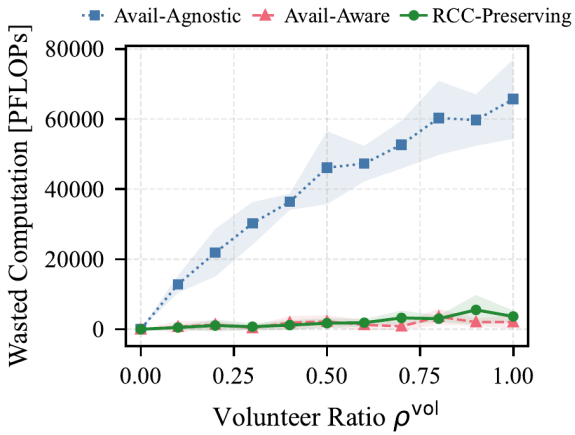
Success rate



(a) Overall success rate

Stays above 90% while the others fall to 75%.

Wasted computation



(b) Wasted computation

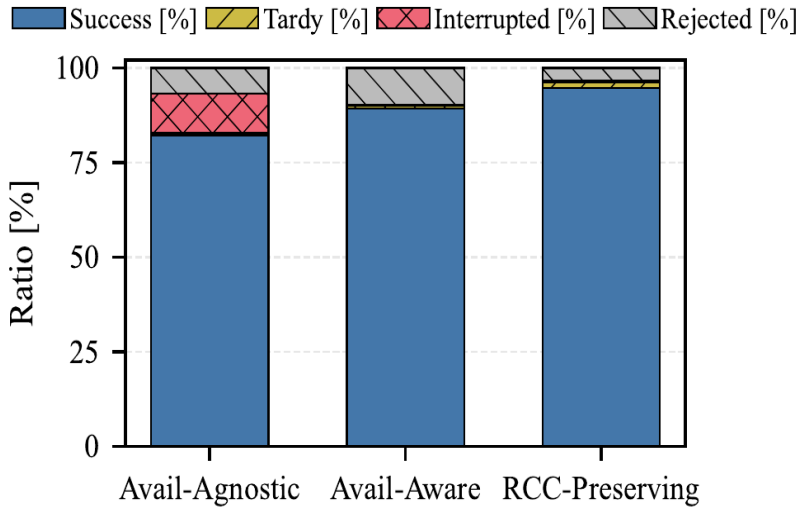
~1K PFLOPs, versus 65K for Avail-Agnostic.

RESULTS (2/5)

RCC-Preserving Trades a Little Tardiness for Fewer Rejections

Rejections fall sharply. Tardiness rises only a little in exchange.

Task outcome breakdown ($\rho^{vol} = 0.6$)



(a) Task outcome breakdown

9.90% → 3.28%

Rejection rate, Avail-Aware to RCC-Preserving.

1.54%

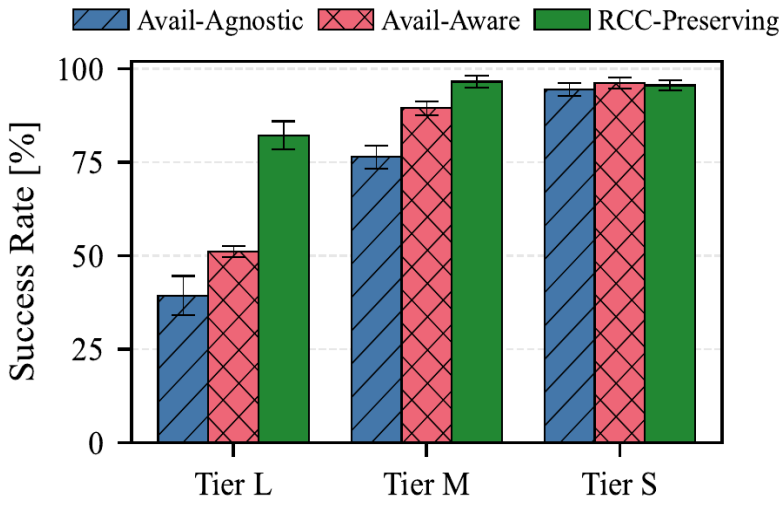
Tardiness under RCC-Preserving. A small trade for the rejection drop.

RESULTS (3/5)

The Gains Concentrate on Large Tasks (Tier L)

Preservation matters only where it is needed: large, high-value tasks.

Tier-wise success rate ($\rho^{vol} = 0.6$)



(b) Tier-wise success rate

39% → 51% → 82%

Tier L (large, long-running tasks) success rate.

94–96%

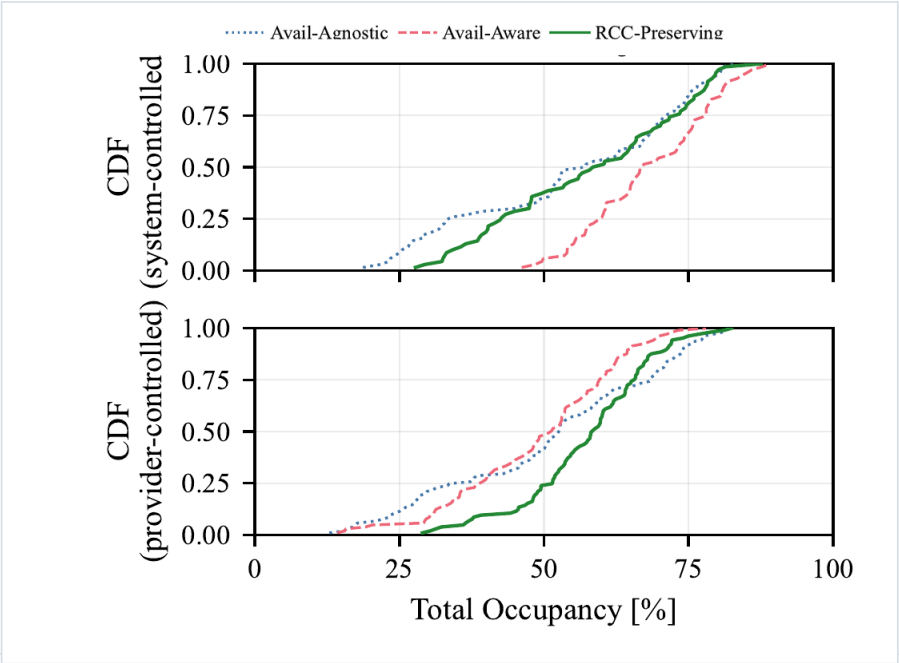
Tier S. All three methods land here already.

RESULTS (4/5)

Short-Lived Nodes Get Used First

This is the mechanism behind the success-rate gains shown earlier.

Compute allocation by node type ($\rho^{vol} = 0.6$)



30.6% → 47.2%

Usage of the least-used provider-controlled (short-lived) nodes. RCC-Preserving puts them to work too.

RESULTS (5/5)

RCC-Preserving Is Also the Fastest to Solve

RCC is a precomputed constant, so the objective stays linear and easy to solve.

Table 2: Per-decision solver overhead at $\rho^{\text{vol}} = 0.6$, aggregated over 10 trials.

Method	Median [s]	95th pct. [s]	Max [s]
Avail-Agnostic	0.082	1.166	10.732
Avail-Aware	0.058	1.467	18.833
RCC-Preserving	**0.013**	**0.075**	**0.905**

Baselines optimize execution time directly. That makes the objective nonlinear in the assignment and data-partition variables, a harder subproblem for the solver.

CONCLUSION

RCC and RCC-Preserving

RCC: value metric unifying performance, memory, residual availability. **RCC-Preserving**: MILP strategy minimizing RCC consumption. Availability as a **constraint** nearly eliminates interruption; in the **objective**, it further raises success rate — especially for large, deadline-constrained tasks.

FUTURE WORK

Lightweight heuristics
for larger clusters, without per-arrival MILP solving

Uncertainty-aware availability
when declared windows deviate from actual reclamation

This work was supported in part by JSPS KAKENHI Grant JP24K14914.