Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

From Nominal Computing Capacity to Effective Service Capacity: Resource Complementarity, GPU Compatibility, and Investment Allocation

Jie Liu

jieliu@gcc.edu.cn

Guangzhou College of Commerce

Research Article

Keywords:

Posted Date: November 13th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11567328/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

From Nominal Computing Capacity to Effective Service Capacity: Resource Complementarity, GPU Compatibility, and Investment Allocation

Jie Liu[1*]

[1*]Center for Digital Economy and Smart Society, Guangzhou College of Commerce, 300 Zhishi Avenue, Huangpu District, Guangzhou, 511363, Guangdong, China.

Corresponding author(s). E-mail(s): jieliu@gcc.edu.cn;

Abstract

Idle computing capacity can coexist with unserved tasks, making investment efficiency depend on compatible resource combinations and allocation rules. We develop an integrated decision framework for combining hardware investment and allocation management to expand effective service capacity. A mixed-integer model links task admission to node-level resources, graphics processing unit (GPU) compatibility, and placement rules. Computational experiments use Alibaba GPU v2023 data. Under common initial conditions, central processing unit expansion admits 99 tasks versus 97 for GPU expansion without model restrictions, but 77 versus 81 with them. Compatibility thus reverses the service-quantity ranking of the two investments. With 20 additional GPUs and no extra per-node installation cap, targeted investment raises admission from 81 under uniform expansion to 95; pooling and repacking on that hardware reach 98, matching an independent upper bound. Despite overlapping stand-alone gains (a quantity interaction of $-\mathbf{7}$), the combined strategy enables service targets of 96–98 tasks that neither intervention can achieve independently within the specified project menu. Installation constraints and heterogeneous task values further condition investment choices. Verified allocations, independent upper bounds, and cost thresholds translate resource constraints into investment decisions. Compatible resource bundles and allocation flexibility determine effective service capacity and the service returns to computing infrastructure investment.

Keywords: effective service capacity, resource complementarity, GPU-model compatibility, capacity investment, resource allocation, computational economics

MSC Classification (2020): 90C10 , 90B50

ORCID (Jie Liu): 0009-0007-2422-2280

1 Introduction

Idle computing hardware can coexist with tasks that a platform cannot admit, limiting the service return from installed capacity. An enterprise task requires a usable combination of graphics processing unit (GPU) capacity, central processing unit (CPU) capacity, and host memory on an eligible node; some tasks also require particular GPU models. Spare resources may be on different nodes, available cards may be incompatible, or free shares spread across cards may not accommodate a whole-card request. These constraints make the service return to investment depend on where capacity is installed and how existing resources are allocated.

The platform must decide which resources to purchase, where to install them, and when to improve allocation instead. Resource complementarity determines which inputs must be available together; GPU-model compatibility determines where those combinations can serve a task. A platform can expand GPUs or complementary resources, widen access to resource pools, or repack incumbent tasks. We develop a unified investment and allocation framework that links hardware stocks, investment locations, and management flexibility to attainable service targets and project costs. The framework evaluates competing investment strategies under a common task menu while preserving incumbent admission guarantees.

Two mechanisms organize the results. First, compatibility changes the resources that constrain admission and can reverse investment rankings. In the paired experiment, CPU and uniform GPU expansion admit 99 and 97 tasks without model restrictions, but 77 and 81 with them. The relevant scarcity is the resource bundle on compatible nodes. Second, achieving higher service targets can require coordinated hardware investment and allocation management even when their incremental admission gains overlap. With 20 additional GPUs and no extra per-node installation cap, targeted investment admits 95 tasks rather than the 81 attained by uniform expansion; pooling and repacking on that hardware reach 98, matching an independent upper bound. Targeted investment and joint allocation management have a quantity interaction of -7, yet both are needed for targets of 96–98 tasks within the specified project menu. The theoretical analysis connects these comparisons to feasible-set changes, compatibility bottlenecks, and target costs. Conditional task values and project costs determine whether quantity advantages translate into preferred investments.

The first related literature concerns capacity investment and resource flexibility. Fine and Freund (1990) and Van Mieghem (1998) analyze investment in flexible capacity, while Van Mieghem (2003) synthesizes capacity-management decisions. Product–plant links and sparse flexibility networks show why the location of flexibility matters alongside aggregate capacity (Jordan and Graves 1995; Chou et al. 2010, 2011). The model advances capacity-flexibility analysis by integrating node-level resource complementarity, fractional GPU requirements, model-specific eligibility, and investment location into a common decision framework. This integration reveals how investments of equal physical size can generate different service capacities and target costs.

The second literature concerns cloud infrastructure and GPU resource allocation. Armbrust et al. (2010) and Iyoob et al. (2013) discuss provisioning and operational decisions; Maglaras and Zeevi (2003) study pricing and capacity sizing, and Perez-Salazar et al. (2022) analyze allocation with service-level agreements. GPU systems

address introspection and migration (Xiao et al. 2018), scheduling and placement (Gu et al. 2019), accelerator heterogeneity (Narayanan et al. 2020), and adaptive allocation (Qiao et al. 2021). Weng et al. (2023) examine fractional-GPU fragmentation and provide the public data used here. By integrating GPU allocation constraints with hardware investment and management decisions, the framework translates operational resource configurations into verified comparisons of attainable service capacity and conditional investment costs. Common request vectors isolate changes in capacity, eligibility, and placement rules.

Extensions across sampled task menus, workload levels, compatibility patterns, installation constraints, and heterogeneous task values map how operating conditions reshape investment advantages and certified project choices. For platform managers, including operators of public or shared computing infrastructure, the resulting decision framework evaluates investment by the usable resource combinations it creates. It identifies when pooling and repacking can replace hardware expenditure and when a higher service target requires both management flexibility and investment. Sections 2–3 present the model and theoretical results; Sections 4–6 develop the data-based comparisons and cost thresholds; Section 7 examines operating conditions, and Section 8 discusses the implications.

2 Economic Environment and Platform Decisions

2.1 Tasks, decision makers, and available choices

Enterprises generate a menu I of computing tasks. Task i requests g_i GPUs, a share $s_i \in (0, 1]$ of each card, CPU capacity c_i, and host memory m_i. Its acceptable GPU models form the set H_i. Node n has a card set K_n, GPU model t_n, CPU capacity C_n, and memory capacity M_n. Compatibility is $A_{in} = 1\{t_n \in H_i\}$. In the unrestricted case, every node model is acceptable.

The platform observes the exogenous task menu, resource requests, and compatibility conditions. It first chooses hardware and allocation-management projects, then determines admission and placement. A task's value $v_i \geq 0$ is the business service value or net contribution revenue that the platform internalizes. It represents enterprise use value when the platform internalizes that value. The benchmark assigns equal weights; Section 7 specifies heterogeneous-value scenarios and reoptimizes admission. These values are scenario parameters, distinct from observed resource requests.

CPU background tasks remain fixed. Their resource use is deducted from node capacities, leaving $\bar{C}_n$ and $\bar{M}_n$. Incumbent GPU tasks $R \subseteq I$ are guaranteed admission. The pool-access rule $P_{in}(s)$ determines eligible nodes, with $s = 1$ widening access; $r = 1$ permits incumbent GPU tasks to be repacked. A legal initial allocation a^0 satisfies all constraints of the scenario. Management changes preserve incumbent admission. Resource pooling refers to access across pools; fractional GPU sharing refers to several tasks occupying shares of the same card.

Multi-resource allocation is also central to dominant resource fairness (Ghodsi et al. 2011), while Themis studies finish-time fairness in GPU clusters (Mahajan et al. 2020). The present model maximizes platform service value subject to incumbent admission guarantees and deterministic resource requirements. This objective complements the

completion-time perspective, while deterministic requirements provide a distinct allocation setting from the chance-constrained overcommitment studied by Cohen et al. (2019).

2.2 Resource constraints and effective service capacity

Let $x_{in} \in \{0,1\}$ indicate assignment to node n, $z_{ink} \in \{0,1\}$ indicate use of its card k, and $y_i = \sum_n x_{in} \leq 1$ indicate admission. Given hardware project h, feasibility requires

$$x_{in} \leq A_{in} P_{in}(s), \qquad \sum_i c_i x_{in} \leq \bar{C}_n(h), \qquad \sum_i m_i x_{in} \leq \bar{M}_n(h). \tag{1}$$

$$\sum_{k \in K_n(h)} z_{ink} = g_i x_{in}, \qquad \sum_i s_i z_{ink} \leq 1. \tag{2}$$

Equation (1) requires CPU, memory, and compatible hardware on the same permitted node. Equation (2) retains per-card shares and multi-card requirements: free shares on different cards cannot be combined into a whole card, and a multi-card task must obtain distinct cards on one node. For $i \in R$, $y_i = 1$; when $r = 0$, its initial node and cards are also fixed.

These conditions define $\mathcal{F}_p(A)$ for project $p = (h, s, r)$. Nominal GPU capacity is the physical card count, recorded by model. Combinable resources are bundles satisfying (1)–(2). Feasible task supply consists of the admission vectors those bundles support. Effective service capacity is the attainable service frontier generated by compatible resource bundles and feasible task placements.

2.3 Objective and optimal investment allocation

For a given project, optimal service value is

$$V_p(A; v) = \max_{(x,z) \in \mathcal{F}_p(A)} \sum_{i \in I} v_i y_i. \tag{3}$$

With hardware cost $C_H(h)$ and allocation-management cost $F(s, r)$, the platform chooses

$$p^* \in \arg\max_{p \in \mathcal{P}} \{V_p(A; v) - C_H(h) - F(s, r)\}. \tag{4}$$

Costs are expressed over a common service window and horizon; the finite project set $\mathcal{P}$ includes no investment. Pooling and repacking may share implementation expenses or create additional migration expenses, so their joint cost need not equal the sum of their stand-alone costs. Task-specific migration costs can be represented by subtracting $\sum_{i \in R} \mu_i 1\{a_i \neq a_i^0\}$ inside (3). Quantity counterfactuals measure attainable service gains under lossless repacking, while the economic framework represents migration and implementation costs through explicit parameters, including μ_i.

Hardware projects can be indexed by a node-level integer vector $d = (d_n)$. The targeted GPU experiment imposes

$$d_n \in \mathbb{Z}_+, \qquad \sum_n d_n = 20, \qquad |K_n(d)| = |K_n| + d_n. \tag{4a}$$

Additional cards have the model already present on the receiving node. CPU and memory remain unchanged. Uniform expansion, $d_n = 1$, is a special case. The constraint fixes physical quantity; model-specific prices and deployment costs determine expenditure. Endogenous placement weakly improves the quantity frontier relative to uniform expansion, with its cost advantage depending on those prices and deployment costs. This distinction follows the treatment of multiple capacity types in Van Mieghem (2003).

Equation (4) trades service value against cost. With a specified service target, the platform instead minimizes expenditure among projects meeting that target; with a budget, it maximizes (3) over affordable projects. For indivisible projects delivering different service quantities, the relevant comparison combines target feasibility or budget feasibility with service value and total expenditure.

3 Theoretical Results

3.1 Aggregate resources and compatibility-constrained bottlenecks

Proposition 1 (Feasible-set expansion and contraction) *Holding the task menu, background tasks, incumbent set, and legal initial allocation fixed, weakly increasing capacity, widening access, or permitting repacking weakly increases optimal service value. Adding GPU-model restrictions weakly decreases it.*

Proof The feasible sets are nested. An expansion retains every original allocation, whereas tighter compatibility removes task–node matches. Maximizing the same objective over these sets gives the result. □

The proposition evaluates service value before project costs, holding initialization fixed to isolate changes in capacity, access, and placement flexibility.

To locate the binding resource combination, consider the special case in which each candidate task requires one whole GPU and identical, strictly positive CPU and memory amounts c, m. After deducting fixed tasks, with no guaranteed tasks left to be repacked, node n provides

$$b_n = \min\{G_n, \lfloor \bar{C}_n/c \rfloor, \lfloor \bar{M}_n/m \rfloor\}$$

task slots, where G_n is its available whole-card count. Let $N(U)$ be the union of compatible, permitted nodes for tasks in U.

Proposition 2 (Compatibility-constrained capacity) *In this homogeneous-task case, maximum admission is*

$$Q = \min_{U \subseteq I} \{|I \setminus U| + \sum_{n \in N(U)} b_n\}. \tag{5}$$

All tasks can be served if and only if every task subset has no more tasks than slots in its compatible neighborhood. An expansion that leaves the capacity of an original minimum cut unchanged leaves maximum admission unchanged.

The proof in Appendix A applies classical matching and maximum-flow/minimum-cut arguments (Hall 1935; Ford and Fulkerson 1956). Equation (5) identifies the compatible resource neighborhood that limits admission. Pooling can enlarge the permitted neighborhood; CPU expansion can increase slots on compatible nodes; GPU expansion elsewhere may leave the bottleneck unchanged. The heterogeneous, fractional-GPU case is evaluated with the integer model (1)–(3).

3.2 Compatibility and project rankings

Let A^0 impose no model restrictions and A^1 impose them, using a common legal initial allocation. Define project p's compatibility loss as $L_p = V_p(A^0; v) - V_p(A^1; v) \geq 0$.

Proposition 3 (Ranking reversal) *Suppose project p has a net-value advantage $D^0_{pq} > 0$ over q without model restrictions. With project costs unchanged, adding compatibility restrictions strictly reverses this ranking if and only if*

$$L_p - L_q > D^0_{pq}. \tag{6}$$

Proof Subtract each project's compatibility loss from its unrestricted value. The restricted net-value difference is $D^1_{pq} = D^0_{pq} - (L_p - L_q)$, which is negative exactly under (6). □

Compatibility changes project rankings through differential value losses. Hardware expansion loses effectiveness when it adds models outside the binding compatible neighborhood; pooling loses effectiveness when newly accessible nodes are incompatible. A ranking reversal occurs when the difference in losses exceeds the original net-value advantage.

3.3 When allocation management substitutes for hardware

For a quantity target q, let $H_g(q)$ be the least hardware expenditure that attains it under management regime g, with $H_g(q) = \infty$ if it is unattainable.

Proposition 4 (Costs at a common service target) *Minimum total expenditure is*

$$C^*(q) = \min_g \{F_g + H_g(q)\}. \tag{7}$$

If management alone attains q on existing hardware, it is cheaper than hardware investment without management when $F_g < H_0(q)$. If maximum admission after pooling and repacking remains below q, management on existing hardware cannot replace all additional hardware investment.

Proof Partition all target-feasible choices by management regime. Minimize hardware expenditure within each group, then compare total costs across groups. This gives (7). A group attaining the target without hardware has zero required hardware expenditure. When its frontier falls short, zero hardware investment is not target-feasible. □

The ability of allocation management to substitute for hardware investment is governed by the service frontier it creates. Management can replace additional capacity when its feasible allocations meet the target, while more demanding targets may require both interventions.

3.4 Heterogeneous values and project interactions

Proposition 5 (Value evaluation and project interactions) *For a fixed project, $V_p(A; v)$ is continuous, convex, and piecewise linear in task values. Equal-weight quantity rankings do not generally determine net-value rankings with heterogeneous values. Even when task production requires complementary resources, the value interaction $V_{ab} - V_a - V_b + V_0$ between two projects can be positive, zero, or negative.*

Proof Each of finitely many feasible admission vectors defines a linear value function. Their maximum is a continuous, convex, piecewise-linear envelope; changing values may change the maximizing vector. Simple constructions give all three interaction signs. When both GPU and CPU are insufficient, neither individual expansion adds service but their combination does. When either of two expansions enables the same task, each adds one task and the combination still adds only one, giving a negative interaction. Dedicated expansions for two independent tasks yield zero interaction. □

The economic literature on complementarity links technology and organizational choices (Milgrom and Roberts 1990). At the task level, production requires complementary resources; at the project level, optimized gains may overlap or reinforce each other. Negative interactions capture overlapping gains. With heterogeneous task values, reoptimizing (3) or constructing valid value bounds identifies the resulting investment comparison.

4 Data and Quantitative Design

4.1 Resource requests and the fixed samples

The analysis uses Alibaba GPU v2023 (Alibaba 2023), frozen at repository commit `3858992ca41bf92082e6d6522ddf0bebd14e838b`. The node table contains 1,523 nodes and the default pod table 8,152 tasks. Node fields `gpu`, `cpu_milli`, `memory_mib`, and `model` identify card counts, CPU millicores, host memory in mebibytes (MiB), and GPU models. Task fields `num_gpu`, `gpu_milli`, `cpu_milli`, and `memory_mib` construct request vectors. The task unit is a pod, which may form part of a larger enterprise job.

The data and fractional-GPU constraints are directly related to Weng et al. (2023). Other production studies show why these distinctions matter operationally: Jeon et al. (2019) analyze locality, gang scheduling, and utilization in training clusters, while Li et al. (2026) document stranded capacity, CPU mismatch, and network constraints in

a later Alibaba production system. The latter study supplies operational context; all calculations here use the dataset specified above.

Identifiers are prefixed with `A3-v1:` and sorted by their 256-bit Secure Hash Algorithm (SHA256) hashes. The main sample takes the first 20 GPU nodes, 100 GPU tasks, and 20 CPU background tasks. The comparable-load scenario takes the first 40 nodes, 200 GPU tasks, and 40 CPU background tasks under the same ordering. Thus GPU tasks per node equal five and background tasks per node equal one in both samples. Two resource pools are formed by fixed sorting and hashing rules. Background tasks are placed first, followed by attempts to place the first half of the GPU menu; successful placements form the guaranteed incumbent set. These rules were fixed before examining the corresponding counterfactual gains.

Table 1 Sample resources and requested capacity

	20-node sample	40-node scenario
GPU tasks / CPU background tasks	100 / 20	200 / 40
GPU cards	112	219
CPU cores	1,720	3,480
Host memory (GiB)	7,912	15,872
Fractional / whole single-card / multi-card tasks	48 / 50 / 2	93 / 103 / 4
Total requested GPU shares	91.19	179.33
CPU requests including background (cores)	1,326.996	2,609.190
Memory requests including background (GiB)	4,481.565	8,951.010

Source: frozen Alibaba GPU v2023 inputs and deterministic sample rules. GiB denotes gibibytes. GPU shares are requested fractions summed across cards; they are not performance-equivalent units.

Observed capacities and requests define the experimental resource constraints. A standardized service window, simultaneous task entry, pool boundaries, initial placement, and expansion sizes define the allocation scenario. Service quantity measures simultaneous admission under these conditions. GPU capacity is recorded as model-specific card counts and shares, preserving hardware heterogeneity.

4.2 The compatibility scenario

The official file `openb_pod_list_gpuspec33.csv` augments GPU-model requirements. Task-by-task comparison with the default table at the same commit confirms matching row counts, identifiers, and an 11-field schema. Apart from 2,388 changes in `gpu_spec`, requests, quality-of-service (QoS), status, and time fields are identical. In the main menu, 39 GPU tasks acquire model requirements and 61 remain unrestricted. The official file defines the model-eligibility scenario applied to the observed task requests.

Adding compatibility invalidates 18 placements among the original 50 GPU incumbents. Keeping background tasks fixed, the same ordered first-fit procedure attempts to place the first 50 GPU tasks again; 40 are placed legally and become the common incumbent set. Unplaced tasks remain in the menu. A paired baseline uses this

new allocation but removes only model requirements. The original 91-task baseline is retained separately.

4.3 Projects, management regimes, and evaluation

Uniform GPU expansion adds one same-model card to each node. CPU and memory expansion each add 25%; joint hardware expansion adds GPUs and CPUs together. Resource pooling removes the simulated pool boundaries. Joint allocation management combines pooling with GPU task repacking, preserving background positions and incumbent admission. The same integer-allocation framework maximizes admission in every scenario; compatibility enters task groups and eligible-node matches.

The paired experiment uses a common legal initial allocation, 40 guaranteed GPU incumbents, and 20 fixed CPU background tasks. Each project's two sides differ only in GPU-model requirements. Baseline admission is 90 without restrictions and 74 with restrictions. Targeted investment uses the restricted side and (4a), comparing fixed incumbent placements and original pools with joint pooling and repacking.

The objective counts admitted GPU tasks in the standardized window; CPU background tasks consume resources but are not counted. Optimality is established either by integer-solver bounds or by a feasible allocation matching an independent capacity bound. Unclosed repacking cases retain lower and upper bounds. Multiple allocations or hardware distributions can attain the same optimal count. Appendix C identifies the certificates and reproduction entry points.

5 Quantitative Results

5.1 Idle resources and excluded tasks

The original 20-node baseline admits 91 tasks while leaving 29.62 GPU shares, 507.524 CPU cores, and 3,785.372 GiB of memory unused. None of the remaining nine tasks can be directly inserted into that allocation. Resource bundles and node eligibility prevent aggregate slack from translating into service.

Table 2 compares projects under the original unrestricted menu and initialization. GPU and CPU expansion each add seven tasks; memory expansion adds none. Pooling adds six. Repacking alone has a feasible count of 92 and an upper bound of 93. Joint management reaches 100, three above pooling and seven to eight above the optimal repacking count.

The GPU–CPU quantity interaction is $100 - 98 - 98 + 91 = -5$ for 20 nodes and $200 - 200 - 200 + 197 = -3$ for 40 nodes. The finite menu and overlapping standalone gains limit the joint increment. The negative project-level interaction reflects overlapping admission gains, whereas task-level resource complementarity determines which GPU–CPU combinations are feasible. In contrast, the 20-node management interaction is $100 - 97 - Q_R + 91 \in [1, 2]$, where Q_R is optimal repacking-only admission. Pool access and changes in incumbent placement reinforce one another in this scenario.

The 40-node scenario preserves the task-to-node ratio but changes the node mix, task composition, and initialization. Baseline admission rises from 91% to 98.5%. Under these comparable task-count loads, the main projects still improve admission, while

Table 2 Admission and project gains without GPU-model restrictions

Project	20 nodes / 100 tasks Admission	Gain	40 nodes / 200 tasks Admission	Gain
Baseline	91	0	197	0
Uniform GPU expansion	98	7	200	3
CPU expansion	98	7	200	3
Memory expansion	91	0	197	0
GPU and CPU expansion	100	9	200	3
Resource pooling	97	6	199	2
Task repacking	[92,93]	[1,2]	[197,200]	[0,3]
Joint allocation management	100	9	—	—

Counts are GPU tasks. Gains use the original 91-task and 197-task baselines. Brackets report feasible lower and valid upper bounds; other reported values are optimal. Joint management combines pooling and repacking. The 40-node joint-management case was not computed. The 91-task initialization differs from the paired experiment's 90-task baseline.

overlapping gains eliminate the quantity advantage of joint hardware expansion over either individual expansion.

5.2 Bottlenecks on compatible nodes

Table 3 holds the legal initial allocation fixed. Admission falls from the paired unrestricted baseline of 90 to 74 when model requirements are imposed. This 16-task difference isolates compatibility; the difference between the original 91 and 74 also includes initialization changes.

Table 3 Admission in the 20-node compatibility scenario

Project	Admission	Gain	Fractional / whole / multi
Paired unrestricted baseline	90	—	47 / 41 / 2
Restricted baseline	74	0	36 / 36 / 2
Uniform GPU expansion	81	7	38 / 41 / 2
CPU expansion	77	3	38 / 37 / 2
Resource pooling	76	2	38 / 36 / 2
Joint allocation management	84	10	41 / 41 / 2

All rows use 20 nodes, 100 GPU tasks, 20 fixed CPU background tasks, and 40 guaranteed GPU incumbents. The first two rows differ only in compatibility. Gains are relative to 74; lower and upper bounds coincide in every row. Task categories are fractional single-card, whole single-card, and multi-card requests. The original 91-task baseline belongs to Table 2.

Among uniform GPU expansion, CPU expansion, pooling, and joint management, joint management achieves the highest admission, serving 84 tasks. The 16 remaining tasks reveal binding resource constraints on nodes hosting acceptable GPU models.

Five T4 nodes contain ten T4 cards, whereas 24 T4-only tasks request 20.44 GPU shares. Per-card constraints strengthen the shortage: 22 tasks request more than 0.5 of

a card, and the other two request 0.47 and 0.32. The 0.47 task cannot share with any of the first 22; the 0.32 task can share with the 0.65 task or with the 0.47 task. Thus these 24 tasks require at least 23 T4 cards. Idle cards of other models cannot meet this requirement.

Joint management raises T4-only admission from five to ten and P100-only admission from four to six, and admits three previously excluded tasks accepting the V100 series. Fourteen T4 tasks and two P100 tasks remain unadmitted, despite 34.64 unused GPU shares overall. Every task individually fits at least one compatible node in the absence of competition. The bottleneck concerns simultaneous resource combinations. The 23-card T4 bound is necessary for that subset, not a sufficient plan for serving all 100 tasks.

5.3 Compatibility reverses CPU and GPU investment rankings

The paired counterfactuals in Table 4 differ only in model requirements, retaining identical tasks, requests, incumbent guarantees, initial placements, and project rules. Their losses therefore implement Proposition 3 directly.

Table 4 Paired project outcomes with and without GPU-model restrictions

	Admission			Gain over baseline		
Project	Without	With	Loss	Without	With	Change
Baseline	90	74	16	0	0	0
Uniform GPU expansion	97	81	16	7	7	0
CPU expansion	99	77	22	9	3	−6
Resource pooling	97	76	21	7	2	−5
Joint allocation management	100	84	16	10	10	0

All quantities are optimal GPU-task counts. The two sides share a legal initial allocation and 40 GPU incumbents. Gains use 90 without model restrictions and 74 with them. Loss is unrestricted minus restricted admission; change is restricted minus unrestricted gain. The original 91-task baseline is not used here.

CPU expansion serves two more tasks than GPU expansion without restrictions, but loses six more tasks to compatibility and consequently serves four fewer with restrictions. This is a strict reversal of the quantity ranking. GPU expansion and pooling move from a tie to a five-task GPU advantage. Joint management remains first, and CPU expansion remains slightly ahead of pooling. Compatibility changes some rankings while preserving the quantity advantage of joint management.

The change in each project's incremental admission equals the baseline compatibility loss minus its own loss: $\Delta Q_p^1 - \Delta Q_p^0 = 16 - L_p$, where L_p is evaluated at unit task values. CPU and pooling gains fall by six and five tasks. GPU and joint-management gains remain unchanged, although their total admission each falls by 16. Additional complementary capacity and wider access still have to match acceptable GPU models, so compatibility affects projects differently.

5.4 Targeted investment and management with 20 additional GPUs

Table 5 fixes additional GPUs at 20 and lets the platform choose their locations and corresponding models. Uniform expansion adds one per node; targeted investment chooses the distribution. CPU, memory, and incumbent guarantees are unchanged.

Table 5 GPU investment allocation and joint allocation management

Project	New GPUs	Admission	Upper bound	New-card mix
Uniform GPU expansion	20	81	81	5 / 3 / 11 / 1
Targeted investment	20	95	95	17 / 0 / 0 / 3
Joint management only	0	84	84	0 / 0 / 0 / 0
Targeted investment + joint management	20	98	98	17 / 0 / 0 / 3

New-card mix is T4 / P100 / G2 / V100M32. All rows use the restricted paired scenario with baseline 74. Uniform and targeted investment retain pools and incumbent GPU positions; joint management relaxes both. New cards match the receiving node's model. The targeted row displays one quantity-optimal allocation, not a cost-minimal allocation.

Targeting serves 14 more tasks than uniform expansion. One optimal allocation adds three, nine, and five cards to T4 nodes `openb-node-0995`, `openb-node-0914`, and `openb-node-1460`, respectively, and three cards to V100M32 node `openb-node-0230`. No other node receives cards. This allocation maximizes service quantity; model-specific procurement and deployment costs determine which quantity-optimal placement is economically preferred.

The same hardware admits 98 tasks with pooling and repacking. This allocation satisfies all resource and incumbent constraints and attains a P100 CPU upper bound valid for any placement of additional GPUs. Hence 98 is the global optimum of the joint problem. The two management regimes share a quantity-optimal hardware allocation: the same investment locations support both optima.

The independent bound uses eight P100-only tasks. The three P100 nodes provide 48 CPU cores; fixed background tasks consume eight, leaving 40. The seven smallest CPU requests among the eight tasks sum to 45.06 cores. At most six P100-only tasks can therefore be admitted under any GPU investment and management choice, bounding total admission by 98. The 98-task allocation still leaves 41.75 GPU shares and 416.82 CPU cores unused. These resources cannot substitute for CPU capacity on the P100 nodes. Targeted GPU investment thus shifts the limiting constraint toward complementary resources on compatible nodes.

Targeted investment adds 21 tasks over the 74-task baseline, management alone adds ten, and the combination adds 24. Their quantity interaction is $98 - 95 - 84 + 74 = -7$. The stand-alone gains overlap. Within the choice set of 20 added GPUs and unchanged CPU and memory, targeted investment alone reaches at most 95 tasks and management alone reaches 84. The combined strategy attains service targets of 96–98 tasks that neither intervention achieves independently. This establishes a distinction between

overlapping incremental gains and the joint intervention required to achieve higher service targets.

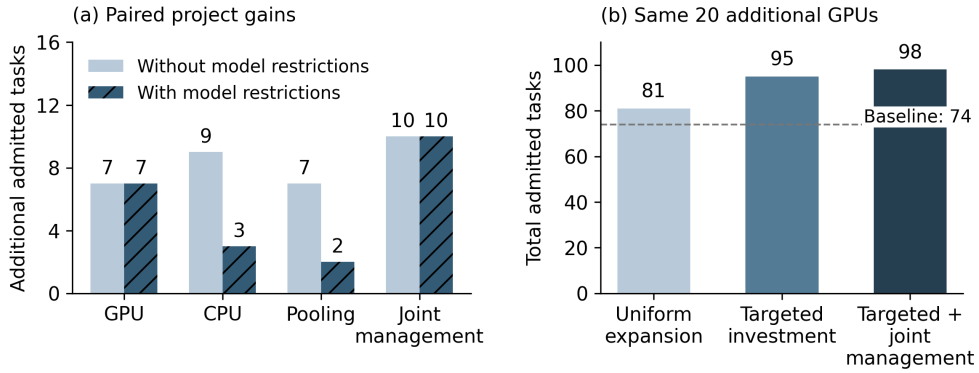


Fig. 1 Compatibility changes incremental gains; investment allocation changes service capacity. Left: project gains relative to each paired baseline (90 unrestricted, 74 restricted), from Table 4. Right: total admission in the restricted scenario, from Table 5; the dashed line marks baseline 74. Both panels report model-based simultaneous admission, using different vertical-axis quantities.

6 Cost Thresholds and Investment Choices

6.1 Management as a substitute in the original unrestricted experiment

For the original 20-node experiment with baseline 91, write GPU and CPU expansion costs as $C_G = 20p_G$ and $C_C = 430p_C$, and pooling and joint-management costs as F_S, F_{SR}. Model-specific GPU costs can be summed, with p_G the project's mean cost per added card. Both individual expansions add seven tasks. CPU expansion has a lower cost per additional task if and only if $p_C/p_G < 20/430 \simeq 0.046512$. Joint management and joint hardware expansion both attain 100, so management has a same-target cost advantage among these computed projects when

$$F_{SR} < C_G + C_C. \tag{8}$$

For a target of 98, either individual expansion suffices, and the least cost is $\min\{C_G, C_C, F_{SR}\}$. In the 40-node scenario, GPU, CPU, and joint expansion all attain 200. With positive component costs, joint expansion is dominated by either single expansion. CPU is cheaper than GPU expansion when $p_C/p_G < 40/870 \simeq 0.045977$. These are minimum-cost comparisons within the specified project menu.

6.2 Choice boundaries under compatibility restrictions

GPU expansion adds seven tasks and CPU expansion three in the restricted scenario. The CPU cost-efficiency condition tightens to

$$\frac{430p_C}{3} < \frac{20p_G}{7} \iff \frac{p_C}{p_G} < \frac{60}{3010} \simeq 0.019934. \tag{9}$$

Joint management adds ten tasks. Its cost per additional task is below those of GPU expansion, CPU expansion, and pooling simultaneously when

$$F_{SR} < \min\{10C_G/7, 10C_C/3, 5F_S\}. \tag{10}$$

At a common service target, the cost boundary differs. Both uniform GPU expansion and joint allocation management can meet a target of 81 tasks, although their maximum admissions are 81 and 84, respectively. Management is less costly at this target when $F_{SR} < C_G$. Among uniform GPU expansion, CPU expansion, pooling, and joint management, only the last reaches 82–84. Targets above 84 require additional hardware alongside any changes in allocation management.

For a common task value v, joint management maximizes net value among these four projects and no investment if all four inequalities hold:

$$\begin{aligned} F_{SR} &\le 10v, \qquad & F_{SR} - C_G &\le 3v, \\ F_{SR} - C_C &\le 7v, \qquad & F_{SR} - F_S &\le 8v. \end{aligned} \tag{11}$$

They compare, respectively, no investment, GPU expansion, CPU expansion, and pooling. Equation (11) selects projects by net service value, while (10) compares expenditure per additional task. These inequalities translate verified service-capacity gains into explicit investment-choice thresholds for task values and project costs.

6.3 Net-value reversal intervals in the paired experiment

Hold project costs fixed across compatibility states and let all tasks have value $v > 0$. CPU expansion's net-value advantage over GPU expansion is $2v - (C_C - C_G)$ without restrictions and $-4v - (C_C - C_G)$ with restrictions. Thus the pairwise preference switches from CPU to GPU if and only if

$$-4v < C_C - C_G < 2v. \tag{12}$$

Other pairwise reversals follow from the same paired counts: pooling to GPU when $-5v < F_S - C_G < 0$; CPU to pooling when $v < C_C - F_S < 2v$; CPU to joint management when $v < F_{SR} - C_C < 7v$; and pooling to joint management when $3v < F_{SR} - F_S < 8v$. Joint management serves three more tasks than GPU expansion in both states, so there is no compatibility-induced strict reversal interval for that pair at a fixed cost difference. Endpoints imply a tie in one state.

Relative to no investment, CPU expansion changes from a positive to a negative net increment when $3v < C_C < 9v$; pooling does so when $2v < F_S < 7v$. Compatibility can

therefore change whether investment is worthwhile as well as which project is preferred. Optimal choice over the entire menu requires comparison with every available project.

6.4 Model-specific costs of targeted investment

Let p_T, p_P, p_{G2}, p_V denote per-card costs for T4, P100, G2, and V100M32, and let D_U, D_T be other deployment costs for uniform expansion and the displayed targeted allocation. Their cost difference is

$$C_T - C_U = 12p_T + 2p_V - 3p_P - 11p_{G2} + D_T - D_U. \tag{13}$$

Model composition and deployment costs determine expenditure at a fixed GPU count. At a common task value, targeting's 14-task advantage justifies its cost premium when $C_T - C_U < 14v$. Comparing costs per additional task instead gives $C_T < 3C_U$, because targeting adds 21 and uniform expansion seven. The first condition evaluates a premium; the second evaluates cost efficiency.

Let F_T be the joint-management cost on targeted hardware and F_0 its cost on original hardware. Adding management after targeting yields three tasks and is worthwhile when $F_T < 3v$. Targeting rather than management alone yields 11 more and is preferred when $C_T - F_0 < 11v$. The combined project yields 14 more than management alone and is preferred when $C_T + F_T - F_0 < 14v$. For targets of 96–98 within the evaluated 20-GPU project menu, combining targeted investment with joint management makes the target attainable. Total procurement and implementation costs determine the expenditure of target-feasible choices.

If hardware cost does not depend on management, the net-value interaction is $-7v-(F_T-F_0)$. It becomes positive only if management cost savings exceed $7v$. Model prices and deployment costs determine the least-cost allocation among quantity-optimal solutions.

6.5 Heterogeneous task values and implementation costs

With heterogeneous values, a platform may reserve compatible capacity for high-value tasks at the expense of total admission. Equation (3) defines each project's value envelope. Specifying task values turns this envelope into a project-value comparison. A feasible allocation $\hat{y}_p$ supplies the lower bound $v \cdot \hat{y}_p$. A quantity upper bound U_p supplies a conservative value upper bound by ignoring resources: add the value of all guaranteed tasks to the values of at most $U_p - |R|$ highest-value non-incumbent tasks. If one project's net lower bound exceeds another's net upper bound, the comparison is robust.

Pooling implementation, migration downtime, code modification, and validation enter as separate cost parameters. The public model field measures hardware eligibility. Pooling widens permitted-node access while preserving H_i. These implementation costs and task values remain explicit parameters throughout the economic comparisons.

7 Investment Choices across Operating Conditions

The benchmark establishes the central mechanisms linking compatibility, investment location, and effective service capacity. We extend the analysis across workload intensity, compatibility networks, installation constraints, and heterogeneous task values to trace how these operating conditions reshape investment decisions.

7.1 Experimental design and the interpretation of bounds

The repeated-sample experiment draws 20 samples at 20 nodes and five samples at each of 40 and 80 nodes from the same public v2023 tables. Each sample contains five GPU tasks and one CPU-background candidate per node. Nodes are stratified by GPU model; GPU tasks are stratified by fractional single-card, whole single-card, or multi-card requests and by the presence of a model requirement. Largest-remainder allocation fixes stratum sizes, and sampling within strata is without replacement. Resource vectors and model requirements are sampled together. A fixed seed schedule determines all samples before project outcomes are evaluated.

Two pool labels are deterministic functions of original identifiers. First-fit placement of the CPU background candidates and the first half of the GPU menu defines the legal initial allocation; successfully placed GPU incumbents are guaranteed admission. All sampled menus are retained for evaluation, irrespective of admission levels or investment rankings. This initialization rule changes the incumbent count across samples. Every restricted/unrestricted pair nevertheless shares its own legal initial allocation, background placement, guarantees, and resource requests. The repeated configurations measure sensitivity to the predetermined sampling scheme.

Each sample compares a baseline, one additional same-model GPU per node, and a 25% CPU expansion, both with and without compatibility restrictions. Restricted samples also compare joint allocation management and targeted GPU investment, the latter adding as many cards as there are nodes, subject to at most two new cards per node. Installation limits are specified implementation scenarios applied to the observed resource configurations.

Every saved lower bound has a task-level allocation certificate. Valid optimization upper bounds are retained when the time limit prevents closure. If $[L_p, U_p]$ bounds project p's service quantity, its gain over the baseline lies in $[L_p - U_0, U_p - L_0]$, intersected with the nonnegative real line for a feasible-set expansion. A strict CPU-to-GPU reversal is certified only when $L_C^0 > U_G^0$ and $U_C^1 < L_G^1$. The count of certified reversals gives a lower frequency bound; samples in which the bounds still permit reversal give an upper bound. This rule certifies rankings using the full feasible and upper-bound information.

7.2 Repeated menus: investment gains and ranking uncertainty

Table 6 reports the resulting reversal-frequency interval and the mean advantage of targeted over uniform GPU investment. Table 7 gives mean and median gain intervals. Intervals bound optimization uncertainty; they are distinct from statistical confidence intervals. Their width identifies where the available allocation certificates determine a project ranking and where several rankings remain feasible.

In the 20-node experiment, 3 of 20 samples certify the strict reversal; accounting for unresolved samples gives a frequency interval of 15–30%. Targeted investment's mean advantage over uniform expansion is bounded by 1.80–2.45 tasks under the common two-card node limit. The corresponding targeting advantage in the original menu under the same two-card node limit is five tasks. Investment location therefore remains consequential in these repeated menus, while its magnitude varies with the menu and initial placement. The 40- and 80-node experiments extend the comparison to larger sampled clusters, with explicit bounds distinguishing certified investment rankings from unresolved alternatives.

Table 6 Investment rankings across repeated, stratified task menus

Nodes	Reversal frequency (%)	Mean targeting advantage	GPU incumbents
20	15–30	1.8–2.5	41–48
40	0–40	0.0–14.2	90–95
80	0–80	0.0–32.2	192–197

Twenty samples at 20 nodes; five at each larger size. Frequency bounds concern a strict CPU-to-GPU ranking reversal when official model restrictions are added. Targeting and uniform investment add the same number of cards; targeting has a two-card per-node limit. Incumbents are determined before policy evaluation.

Table 7 Additional admissions under compatibility: distribution bounds

Nodes	Project	Mean gain	Median gain
20	Uniform GPU	5.7–6.2	5.0
20	CPU expansion	4.7–5.0	4.5–5.0
20	Joint management	3.7–16.2	2.5–14.5
20	Targeted GPU	7.7–8.3	7.5–8.0
40	Uniform GPU	14.6–24.4	13.0–22.0
40	CPU expansion	15.4–18.2	14.0–15.0
40	Joint management	14.6–46.0	17.0–42.0
40	Targeted GPU	15.6–30.6	14.0–31.0
80	Uniform GPU	0.0–55.8	0.0–48.0
80	CPU expansion	0.0–51.4	0.0–49.0
80	Joint management	0.0–56.2	0.0–50.0
80	Targeted GPU	0.0–55.8	0.0–48.0

Gains use each sample's own restricted baseline. Joint management is pooling plus repacking; targeted investment has a two-card per-node limit. Quartile bounds and every sample are provided in the reproducibility workbook.

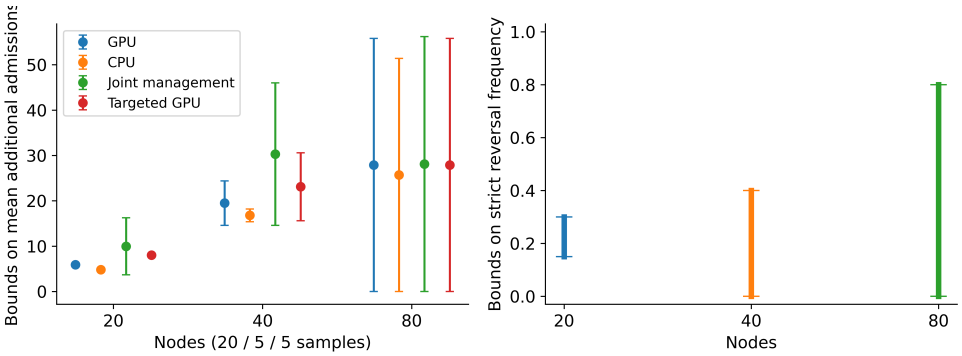


Fig. 2 Bounds on investment gains and reversal frequencies across the repeated samples. Vertical segments reflect feasible and upper-bound results; their midpoints are plotting devices, not point estimates.

7.3 Workload and the compatibility network

We next retain the original 20 nodes, 20 fixed CPU background tasks, 40 GPU incumbents, and their legal initial allocation. Non-incumbent tasks enter in a fixed identifier-based order until GPU request load first reaches a target in $\{0.6, 0.8, 1.0, 1.2, 1.4\}$. Load is $\rho_G = \sum_i g_i s_i / \sum_n G_n$. Because tasks are indivisible, realized load may exceed the target; the workbook records both. CPU, GPU, and memory requests remain jointly observed task vectors.

Within each menu, nested subsets retain 0, 25, 50, 75, or 100% of the nonempty official model requirements. This parameter is the fraction of existing restrictions retained, rather than the fraction of all tasks restricted. Removing a requirement preserves legality of the common initial allocation. The 25 cells separate greater aggregate demand from narrower task–node eligibility.

Figure 3 illustrates project choice under one explicit relative-cost scenario. A baseline task has value one; costs are 20 for uniform GPU investment, 12 for CPU expansion, two for pooling, and six for joint management. These dimensionless parameters specify the cost scenario. A project is certified as strictly preferred only when its feasible net value exceeds every competing project's upper net value. Equality can certify weak optimality but does not identify a unique choice. Otherwise the cell remains unresolved. With admission bounds L_p and U_p, the strict decision criterion is

$$L_p - C_p > \max_{r \neq p} \{U_r - C_r\}.$$

Selective refinement certifies a strict choice in 16 of the 25 cells; 4 additional cells have a certified weakly optimal project without strict separation. Solver effort is directed toward projects whose bounds can still change the choice, using limits of 60, 180, and, for at most six remaining boundary cells, 300 seconds. This criterion directs computation toward certifying the management choice, allowing individual project bounds to remain open when they no longer affect that choice.

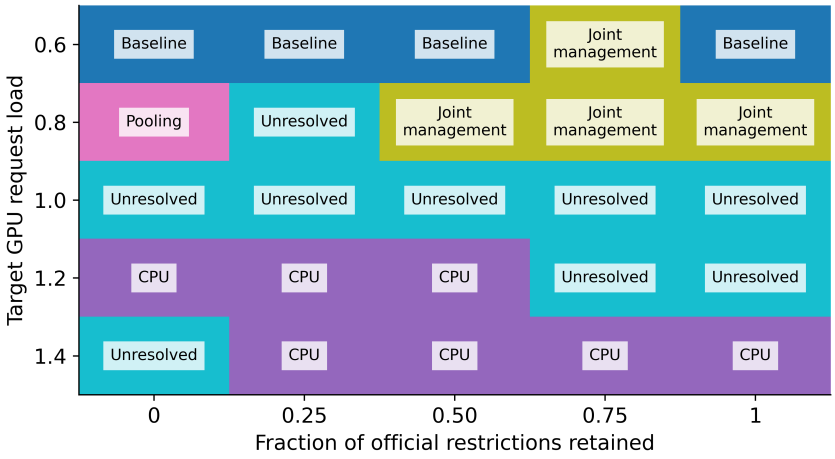


Fig. 3 Certified project choices conditional on load, compatibility, and stated relative costs. "Unresolved" means that optimization bounds do not select a unique project; it does not establish an economic tie. Baseline means no investment or management change.

7.4 Installation constraints and the quantity of investment

The original 95-task optimum permits nine new cards on one T4 node. We now impose $d_n \leq \bar{d}$, with $\bar{d} \in \{1, 2, 4\}$, alongside additional-card totals of 5, 10, 20, and 30. A fourth case retains the original absence of an additional node cap. Combinations with total investment above $20\bar{d}$ are infeasible by construction. Existing certificates establish the original 20-card unrestricted results.

Table 8 holds investment at 20 cards. With a one-card limit, every node must receive exactly one card: targeted investment then has precisely the uniform feasible hardware choice. Larger limits permit the platform to redirect investment toward compatible bottlenecks. Figure 4 shows how this allocation flexibility interacts with the total number of new cards and with pooling and repacking. A strict advantage is certified when the relevant lower bound exceeds the competing upper bound.

The targeted-investment optima are 81, 86, and 91 tasks under one-, two-, and four-card node limits, respectively, compared with the unrestricted benchmark of 95. The corresponding advantages over uniform expansion are zero, five, and ten tasks, versus 14 in the benchmark. Installation space on compatible nodes is a scarce input that determines the service return to a given hardware quantity. With joint management, the admission bounds are 88 under the 1-card limit, 92 under the 2-card limit, 98 under the 4-card limit. Each feasible configuration satisfies the same incumbent guarantees. The independent compatibility-constrained CPU upper bound of 98 continues to apply; installation constraints reduce this bound only if a tighter certificate is available. These comparisons identify which installation limits remain binding after pooling and repacking expand allocation flexibility.

Table 8 Twenty additional GPUs with node-level installation limits

New cards per node	Project	Admission bounds
1	Targeted	81
1	Targeted + management	88
2	Targeted	86
2	Targeted + management	92
4	Targeted	91
4	Targeted + management	98
unbounded	Targeted	95
unbounded	Targeted + management	98

All rows retain the restricted benchmark's task menu and guarantees. "Unbounded" is the original model's absence of an extra node cap; 95 and 98 are frozen exact results. The finite caps are scenarios, not measured chassis specifications.

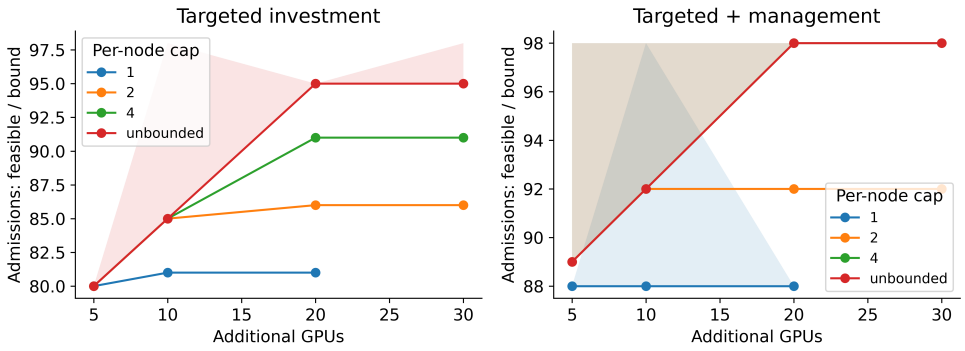


Fig. 4 Hardware quantity, installation flexibility, and allocation management. Lines show certified feasible admissions; shaded regions extend to valid upper bounds. A decrease in a time-limited feasible solution across nested choices is not a decrease in the true optimum.

7.5 Heterogeneous task values and investment ranking robustness

The weighted experiments resolve the allocation problem under three transparent value scenarios. Random values are drawn from $\{0.5, 1, 2\}$ with equal probabilities using ten predetermined seeds, 20261028 through 20261037. The first draw preserves the original realization. Resource-related value is the mean of GPU request shares and CPU demand measured in units of 12 cores, rounded to 0.01 with a minimum of 0.01. The third scenario assigns value two to tasks with a nonempty official model requirement and one to all other GPU tasks. It specifies a scenario premium for tasks with model restrictions. The value scenarios apply to the same task identifiers across projects.

Table 9 reports optimized value bounds together with the task count of the saved feasible allocation. For the two targeted projects, the 20-card budget is combined with the two-card node limit. Each scenario reoptimizes task admission and placement for the specified value objective. Differences in admitted counts show the composition

margin on which value and quantity objectives can diverge; an exact value ranking requires nonoverlapping bounds or a closed optimum.

The resource-related scenario supplies a strict example of this distinction. Under uniform GPU expansion, the value optimum admits 80 tasks and attains 72.86 index units. A separate solve requiring the quantity-maximal 81 admissions has an exact value optimum of 72.73. Thus every quantity-maximizing allocation loses at least 0.13 value units in this scenario. Reoptimization establishes a strict composition trade-off: retaining the last admission displaces a resource combination with higher total service value.

Table 9 Reoptimized admission with heterogeneous task values

Value scenario	Project	Feasible admissions	Value bounds
Random	CPU expansion	77	89.50
Random	Uniform GPU	81	96.00
Random	Targeted GPU	86	101.50
Random	Targeted + management	92	108.50
Resource	CPU expansion	77	69.06
Resource	Uniform GPU	80	72.86
Resource	Targeted GPU	86	77.64
Resource	Targeted + management	86	77.64–86.58
Scarcity	CPU expansion	77	93.00
Scarcity	Uniform GPU	81	101.00
Scarcity	Targeted GPU	86	111.00
Scarcity	Targeted + management	89	117.00–136.00

Values are conditional index units. Targeted + management combines targeted investment and joint management. Equal values remain the original quantity benchmark. The Random rows retain the first draw for comparison; Table 10 summarizes all ten draws. These are scenario values.

Ten common value draws make the ranking comparison explicit. Every draw uses the same values across all seven projects and retains the task menu and initial guarantees. Project net values subtract the stated common-horizon costs. Table 10 reports bounds on winner frequencies and average ranks, so an unresolved weighted allocation is not assigned a definite rank. The admission range describes the saved value-oriented configurations, not a uniquely determined task count at every weighted optimum. Together with the certified resource-value trade-off above, these comparisons separate robustness of investment choice from sensitivity of the admitted task mix. Under these costs, joint management is strictly preferred in all ten draws. CPU expansion has higher net value than uniform GPU expansion in 7 draws; the reverse holds in 2, with 1 tie. Among the 70 project–draw pairs, 9 saved value-optimal configurations admit fewer tasks than the corresponding quantity maximum. Value heterogeneity therefore affects the composition of admission, while the repeated net-value comparisons show how much of the project ordering survives that variation under the specified costs.

Table 10 Investment ranking robustness across ten task-value draws

Project	Strict-best frequency	Mean rank bounds	Feasible admissions
Baseline	0%	3.1–3.5	74
Uniform GPU	0%	6.7–6.8	80–81
CPU expansion	0%	6.2–6.3	77
Pooling	0%	2.3–2.8	75–76
Joint management	100%	1.0	83–84
Targeted GPU	0%	4.8–4.9	84–86
Targeted + management	0%	3.1–3.5	91–92

The lower frequency counts strictly certified winners; the upper frequency retains every project not ruled out by the bounds, including possible ties. Rank intervals use all seven projects. Counts describe saved feasible weighted allocations. Costs are 0, 20, 12, 2, 6, 20, and 26 for the listed projects, respectively; GPU prices equal one and targeted investment adds 20 cards with a two-card node cap.

7.6 A bounded cost–capacity frontier

For each service target from 80 through 98, we compare the baseline, uniform GPU investment, CPU expansion, pooling, joint management, and the computed targeted projects with a two-card per-node limit. Targeted budgets are 5, 10, 20, and 30 cards. Let other GPU-model costs equal one unit and the T4 cost vary over $\{0.5, 1, 2\}$. Joint-management cost varies over $\{2, 6, 12\}$; CPU expansion and pooling retain costs 12 and two. These common-horizon relative costs define a reproducible sensitivity grid. Procurement and management costs enter as scenario parameters.

For a computed project family p, let $\underline{C}_p$ be its budget multiplied by the lowest eligible per-card price plus management cost, and let $\overline{C}_p$ be the cost of its saved feasible hardware mix. A cost–capacity interval over this finite family is

$$\min_{p:U_p \geq q} \underline{C}_p \;\leq\; C^*_{\mathcal{P}}(q) \;\leq\; \min_{p:L_p \geq q} \overline{C}_p. \tag{14}$$

The lower envelope is optimistic about unresolved feasibility and hardware mix; the upper envelope is an implementable candidate cost. This distinction matters when different mixes attain the same maximum admission. At equal card prices the hardware expenditure of a given budget is exact, but unresolved allocation bounds can still leave the minimum target cost uncertain.

Figure 5 reports equal-card-price cost frontiers and heterogeneous service-value bounds. If the upper envelope is finite, its selected project has a checked configuration meeting the target. A missing finite upper value leaves target feasibility open unless an admission upper bound rules it out. A target is infeasible within the specified project family when every project admission upper bound falls below it. A discrete incremental target cost is bounded by $\max\{0, C_L(q+1) - C_U(q)\}$ and $C_U(q+1) - C_L(q)$ where these terms are finite. This discrete difference measures the cost of raising the admission target by one task.

At unit card prices and a joint-management cost of six, the finite-menu cost bounds are as follows: target 80 requires 5 cost units; targets 81–84 require 6 cost units; target 85 requires 10 cost units; targets 86–89 require 11 cost units; targets 90–92 require 16

cost units; targets 93–98 are infeasible in this finite project family. Twenty targeted cards without management under the two-card node limit admit at most 86 tasks, while five targeted cards with joint management admit 89. The latter project costs eleven units at the stated prices, compared with twenty for the former; at target 86 it saves nine scenario cost units relative to that particular hardware-only alternative.

The bounded cost–capacity frontier quantifies how allocation management can substitute for hardware expenditure at a given service target. GPU-model compatibility, installation flexibility, and implementation costs jointly determine the magnitude of this substitution advantage. Within the specified project family and cost scenarios, the resulting thresholds identify certified opportunities to attain service targets through less costly combinations of hardware and management.

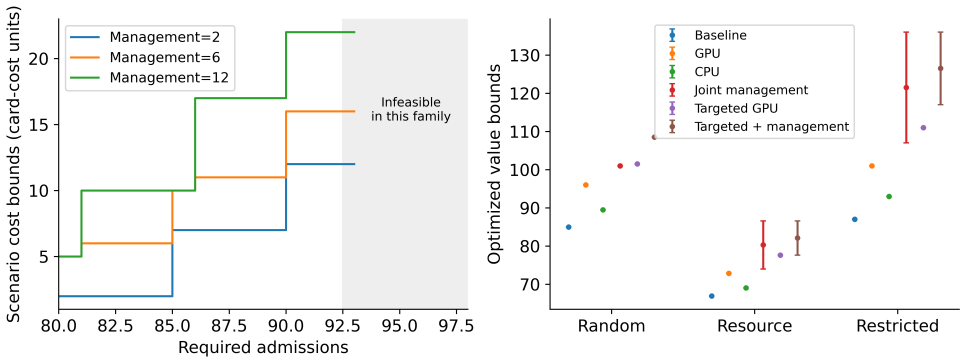


Fig. 5 Conditional cost and value frontiers. Left: feasible cost upper envelopes and optimistic lower bounds with equal card prices and a two-card node limit. Right: value bounds after reoptimization. Costs and values are scenario indices, not estimated monetary returns.

8 Discussion and Conclusion

Effective service capacity depends on resource combinations that can serve tasks on eligible nodes. CPU capacity, host memory, GPU-model compatibility, and placement rules determine how much of an installed hardware stock becomes usable. The experiments connect these constraints to three platform decisions: which resources to expand, where to install them, and when to improve allocation. Aggregate GPU counts provide a measure of physical capacity; the feasible task frontier supplies the corresponding measure for service planning.

Compatibility changes the return to different investments by changing where resources are scarce. In the paired comparison, it reverses the CPU–GPU quantity ranking while preserving the common task menu and legal initial allocation. The investment implication is to evaluate complementary capacity on compatible nodes. The v2023 counterfactuals provide an investment-level counterpart to the stranded-capacity mechanisms documented in subsequent Alibaba production evidence (Li et al. 2026).

Investment location and allocation management jointly shape the feasible service frontier. With the same number of additional cards, targeted investment improves admission by directing capacity toward compatible bottlenecks. Pooling and repacking then expand the assignments supported by that hardware. Their gains overlap, producing a negative quantity interaction, yet higher service targets still require both interventions within the specified project menu. Installation limits govern how far investment can be concentrated, while complementary resources on compatible nodes determine the remaining upper bound. The distinction reveals two dimensions of investment strategy: the efficiency of incremental capacity gains and the combination of interventions required to achieve higher service targets.

The economic choice depends on the service target, task values, and implementation costs. Target-cost comparisons identify when management can replace hardware expenditure and when additional capacity remains necessary. Weighted reoptimization identifies changes in the admitted task mix, and the finite-menu cost frontier connects those allocation choices to expenditure at specified scenario prices. The certified benchmark, repeated small-cluster samples, and load–compatibility grid support these comparisons under different operating conditions. Across operating conditions, reproducible feasible allocations and valid upper bounds map the economic conditions governing investment rankings, target feasibility, and management substitution.

For shared computing platforms and public computing services, these results support assessing projects by the usable resource combinations they create and the service targets they attain. Operators can compare expanded pool access and repacking with investment on compatible nodes before choosing a hardware programme. Chinese policy provides one application: Chapter 12 of the 15th Five-Year Plan addresses computing infrastructure, access, and scheduling (People's Republic of China 2026), corresponding to hardware stocks and permitted-node sets in the model. The account of CUDA adaptation and software-stack migration in Qiushi Economic Editorial Department and CCID Joint Research Group (2026) highlights a related implementation issue. Software-stack adaptation and migration represent a separate implementation-cost dimension in the broader economics of deploying usable computing capacity.

Effective service capacity is measured through simultaneous task admission, combining observed resource requests and node capacities with explicit compatibility and allocation rules. Scenario task values and implementation costs translate this service frontier into economic project choices. The central investment principle is to evaluate hardware and management projects by the compatible resource bundles they make available, the service targets they support, and the total cost of delivering those targets. Integrating capacity procurement with allocation management provides a unified economic framework for expanding effective computing services in shared infrastructure systems.

A Proof of Proposition 2

Construct a bipartite flow network. Source-to-task arcs have capacity one; task-to-permitted-node arcs have capacity greater than every finite cut capacity; and node-to-sink arcs have capacity b_n. Integral flows correspond to feasible task assignments. In

any finite minimum cut, if task subset U remains on the source side, its neighborhood $N(U)$ must remain there too. The cut capacity is $|I \setminus U| + \sum_{n \in N(U)} b_n$. The maximum-flow/minimum-cut relation and integrality give (5), applying the classical results of Hall (1935) and Ford and Fulkerson (1956). If an original minimum cut does not increase after an expansion, the new optimum is at most the old value. Proposition 1 supplies the reverse inequality, so the gain is zero.

B Bounds with Heterogeneous Task Values

All guaranteed tasks must be admitted, and at most $U_p - |R|$ other tasks can be admitted. Ignoring resource constraints and selecting the highest-valued such tasks yields an upper bound. Any saved legal allocation yields a lower bound. The argument uses nonnegative values, a common task menu, and a quantity upper bound valid for the same project. It does not interpret an equal-weight optimum as a weighted optimum.

C Results, Certificates, and Reproduction

The original 20-node unrestricted experiment in Table 2 has baseline 91. Tables 3–5 use a common legal initial allocation with baselines 90 unrestricted and 74 restricted. The reproducibility materials map data fields, variables, allocation rules, and result files.

The 95-task targeted result has a feasible allocation, integer-residual checks, and a solver global-bound record. Forty GPU incumbents are fixed, and the minimization dual bound for the remaining objective is -55, implying an upper bound of 95 total tasks. This certificate uses the numerical integer-optimization standards of SciPy/HiGHS. The 98-task joint result combines a full feasible allocation with the independent P100 CPU bound, which reads only capacities, requests, and fixed background positions. The time-limited unrestricted investment search is retained as solver history but does not enter the 98-task upper-bound calculation.

The reproducibility package contains a certificate-verification entry point that does not invoke an optimizer. It checks raw fields, deterministic sampling, paired initialization, additional cards by node, task assignments, and card-level loads, and recomputes table differences, interactions, and cost thresholds. Optional optimization entry points write to new directories. Repacking alone remains bounded by 197–200 tasks for 40 nodes and 92–93 for 20 nodes. Proposition 2 is supported by its general proof and a finite-enumeration check.

SHA256 manifests identify the archived inputs, code, and result files. The result tables and allocation certificates link the reported comparisons to their computational records. Certificate verification is separate from optimization; the experiments in Section 7 retain their instance-specific solver records and bounds.

D Reproducing the Decision Extensions

The extension directory records the experiment design, source-data hashes, deterministic sampling rules, software versions, every task menu, initial allocation, installed-card vector, and task-to-node/card assignment. Initial solves used a two-second optimization

limit for ordinary calls and eight seconds for installation and weighted experiments. Model construction and certificate verification add elapsed time. When no solver incumbent is returned, a legal initial placement and deterministic first-fit extension supply a feasible lower bound; the total menu value supplies a conservative upper bound. Solver bounds are used only when available. Twelve tiny instances were independently enumerated to check weighted objectives, compatibility, multi-card allocation, and node installation limits.

An independent certificate checker recomputes CPU and memory use, card shares, model and pool eligibility, incumbent guarantees, investment totals, node caps, and the objective. The grouped model includes task value in its grouping key and decomposes per-card degree counts into distinct-card task assignments. For strict ranking frequencies and policy comparisons, unresolved bounds are propagated rather than discarded. The workbook supplies raw scenario outcomes, gain-distribution bounds, actual request loads, cost envelopes, and conditional value-choice sets. The reported reversal frequencies characterize the predefined sampled task menus, while project costs and task values enter the economic analysis as explicit scenario parameters.

Selected unresolved instances are refined using saved feasible allocations, with each optimization attempt recorded separately. Grid calls use staged optimization limits of 60, 180, and 300 seconds, with an additional 45-second process allowance. A later call never weakens an existing feasible lower bound or valid upper bound. Every accepted configuration is independently checked against integer card indices, GPU shares, node CPU and memory, compatibility, pool access, incumbent guarantees, investment totals, and installation caps. Ten random-value draws use seeds 20261028 through 20261037 and common task values across projects. Time-limited results retain explicit bounds. The main script and checkpoints distinguish recomputation from certificate-only verification.

Refinement supplies saved feasible allocations to the bundled HiGHS solver as integer-programming starting solutions. The starting vector is checked against the original matrix before optimization, including the ordering of newly installed cards. The model and feasible set are unchanged. Weighted refinement calls use a 60-second optimization limit; previously certified first-draw optima are reused. Across the ten draws, 70 of the 70 project value optima are certified in the recorded results. Equal net values are retained as ties in the rank intervals.

Statements and Declarations

Funding. No funding was received for this research.

Competing Interests. The author declares no financial or non-financial competing interests.

Data Availability. The original cluster traces can be obtained from the Alibaba ClusterData repository, GPU v2023, at commit `3858992ca41bf92082e6d6522ddf0bebd14e838b` (Alibaba 2023). Derived experimental inputs, allocation certificates, and result summaries are included in the accompanying reproducibility package (Online Resource 1).

Code Availability. Online Resource 1 contains code for instance generation, optimization, certificate verification, and figure reproduction, together with environment information and running instructions.

Author Contributions. Jie Liu is the sole author and carried out all aspects of the research and manuscript preparation.

References

Alibaba (2023) Cluster Trace GPU v2023. URL https://github.com/alibaba/clusterdata/tree/3858992ca41bf92082e6d6522ddf0bebd14e838b/cluster-trace-gpu-v2023, commit 3858992ca41bf92082e6d6522ddf0bebd14e838b

Armbrust M, Fox A, Griffith R, et al (2010) A View of Cloud Computing. Communications of the ACM 53(4):50–58. https://doi.org/10.1145/1721654.1721672

Chou MC, Chua GA, Teo CP, et al (2010) Design for Process Flexibility: Efficiency of the Long Chain and Sparse Structure. Operations Research 58(1):43–58. https://doi.org/10.1287/opre.1080.0664

Chou MC, Chua GA, Teo CP, et al (2011) Process Flexibility Revisited: The Graph Expander and Its Applications. Operations Research 59(5):1090–1105. https://doi.org/10.1287/opre.1110.0987

Cohen MC, Keller PW, Mirrokni V, et al (2019) Overcommitment in Cloud Services: Bin Packing with Chance Constraints. Management Science 65(7):3255–3271. https://doi.org/10.1287/mnsc.2018.3091

Fine CH, Freund RM (1990) Optimal Investment in Product-Flexible Manufacturing Capacity. Management Science 36(4):449–466. https://doi.org/10.1287/mnsc.36.4.449

Ford LRJr., Fulkerson DR (1956) Maximal Flow Through a Network. Canadian Journal of Mathematics 8:399–404. https://doi.org/10.4153/CJM-1956-045-5

Ghodsi A, Zaharia M, Hindman B, et al (2011) Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In: 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11). USENIX Association, pp 323–336, URL https://www.usenix.org/conference/nsdi11/dominant-resource-fairness-fair-allocation-multiple-resource-types

Gu J, Chowdhury M, Shin KG, et al (2019) Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In: 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). USENIX Association, pp 485–500, URL https://www.usenix.org/conference/nsdi19/presentation/gu

Hall P (1935) On Representatives of Subsets. Journal of the London Mathematical Society s1-10(1):26–30. https://doi.org/10.1112/jlms/s1-10.37.26

Iyoob I, Zarifoglu E, Dieker AB (2013) Cloud Computing Operations Research. Service Science 5(2):88–101. https://doi.org/10.1287/serv.1120.0038

Jeon M, Venkataraman S, Phanishayee A, et al (2019) Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In: 2019 USENIX Annual

Technical Conference (USENIX ATC 19). USENIX Association, pp 947–960, URL https://www.usenix.org/conference/atc19/presentation/jeon

Jordan WC, Graves SC (1995) Principles on the Benefits of Manufacturing Process Flexibility. Management Science 41(4):577–594. https://doi.org/10.1287/mnsc.41.4.577

Li S, Yang L, Yu H, et al (2026) Heterogeneity at Hyperscale: Characterization and Scheduling of Large Production AI Clusters at Alibaba (Operational Systems). In: 20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26). USENIX Association, pp 2187–2203, URL https://www.usenix.org/conference/osdi26/presentation/li-suyi

Maglaras C, Zeevi A (2003) Pricing and Capacity Sizing for Systems with Shared Resources: Approximate Solutions and Scaling Relations. Management Science 49(8):1018–1038. https://doi.org/10.1287/mnsc.49.8.1018.16402

Mahajan K, Balasubramanian A, Singhvi A, et al (2020) Themis: Fair and Efficient GPU Cluster Scheduling. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). USENIX Association, pp 289–304, URL https://www.usenix.org/conference/nsdi20/presentation/mahajan

Milgrom P, Roberts J (1990) The Economics of Modern Manufacturing: Technology, Strategy, and Organization. American Economic Review 80(3):511–528. URL https://www.gsb.stanford.edu/faculty-research/publications/economics-modern-manufacturing-technology-strategy-organization

Narayanan D, Santhanam K, Kazhamiaka F, et al (2020) Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). USENIX Association, pp 481–498, URL https://www.usenix.org/conference/osdi20/presentation/narayanan-deepak

People's Republic of China (2026) Outline of the 15th Five-Year Plan for National Economic and Social Development. URL https://wb.beijing.gov.cn/home/gjjwzx/xxzl/202603/t20260316_4557729.html, in Chinese; title translated

Perez-Salazar S, Menache I, Singh M, et al (2022) Dynamic Resource Allocation in the Cloud with Near-Optimal Efficiency. Operations Research 70(4):2517–2537. https://doi.org/10.1287/opre.2021.2138

Qiao A, Choe SK, Subramanya SJ, et al (2021) Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning. In: 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21). USENIX Association, pp 1–18, URL https://www.usenix.org/conference/osdi21/presentation/qiao

Qiushi Economic Editorial Department and CCID Joint Research Group (2026) Seizing the Commanding Heights of the Intelligent Era: A Survey of China's Artificial Intelligence Industry. Qiushi URL https://www.qstheory.cn/20260430/fcfeb89054fe453ebe265f7e270917e9/c.html, no. 9. In Chinese; title translated

Van Mieghem JA (1998) Investment Strategies for Flexible Resources. Management Science 44(8):1071–1078. https://doi.org/10.1287/mnsc.44.8.1071

Van Mieghem JA (2003) Commissioned Paper: Capacity Management, Investment, and Hedging: Review and Recent Developments. Manufacturing & Service Operations Management 5(4):269–302. https://doi.org/10.1287/msom.5.4.269.24882

Weng Q, Yang L, Yu Y, et al (2023) Beware of Fragmentation: Scheduling GPU-Sharing Workloads with Fragmentation Gradient Descent. In: 2023 USENIX Annual Technical Conference (USENIX ATC 23). USENIX Association, pp 995–1008, URL https://www.usenix.org/conference/atc23/presentation/weng

Xiao W, Bhardwaj R, Ramjee R, et al (2018) Gandiva: Introspective Cluster Scheduling for Deep Learning. In: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). USENIX Association, pp 595–610, URL https://www.usenix.org/conference/osdi18/presentation/xiao

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- A3PublicCodeandReproducibility20261029.zip