10.48047/jocaaa.2026.35.06.23

Self-Correcting Dynamic Quota: A Kubernetes Quota Policy Where Hoarding Is Automatically Self-Defeating

Pradeep Kandepaneni†, Saketh Gowerneni‡, Vasu Babu Narra§, Praneeth Ganta¶, Murali M. Chittabathini*

†Independent Researcher, USA, pradeepkandepaneni@gmail.com
‡Independent Researcher, USA, gsaketh369@gmail.com
§Independent Researcher, USA, narravasu77@gmail.com
¶Independent Researcher, USA, praneethganta@gmail.com
*Independent Researcher, USA, murali.mohan.rao@gmail.com

Abstract—Every platform team running shared Kubernetes knows the pathology: tenants over-request. A namespace that needs four cores asks for sixteen, because quotas are sticky, reclamation is painful, and hoarding is individually rational. The cluster appears full at a fraction of real utilisation, the quota system rewards dishonesty, and the platform team referees disputes it has no principled way to resolve. The textbook fix from economics, a resource auction, fails here because there is no money in an internal cluster, teams have no dollar value for a core, and auctions leak value the platform cannot recover. This paper introduces the Self-Correcting Dynamic Quota, a quota policy in which over-requesting is automatically self-defeating. Teams claim contested capacity with a conserved internal credit and earn credits by releasing capacity when their need is low. Because credits are conserved rather than spent externally, the platform runs exactly zero deficit, and across repeated allocation rounds requesting in proportion to true need is the best policy a team can follow: credits squandered on hoarding now are credits unavailable when a genuine peak arrives. Hoarding is not policed; it depletes the hoarder's own future claim. The contribution is the conserved-credit mechanism, its budget-balance and incentive properties argued from repeated-game theory, and a comparative analysis against quotas, fair-share scheduling, and auctions. The quantitative figures reported are projected design targets illustrated on sample scenarios, not measured outcomes; empirical validation is left to future work.

Index Terms—kubernetes, resource quota, multi-tenancy, mechanism design, repeated games, incentive

compatibility, budget balance, fair allocation, platform engineering, cluster utilization

I. Introduction

Shared-cluster allocation today rests on static quotas, priority classes, and fair-share schedulers, and all three are gamed the same way. Because obtaining and expanding quota is slow and painful, the rational tenant grabs as much as possible up front and holds it defensively. The aggregate result is a cluster full of idle, hoarded capacity that reports high utilisation while delivering low real work, and a platform team fighting it with approval workflows and reclamation policies that are expensive, adversarial, and perpetually a step behind.

The pathology is not a failure of the tenants; it is a failure of the incentive the quota system creates. When holding capacity is free and reclaiming it is costly, hoarding is the dominant strategy, and no amount of exhortation will change a dominant strategy. The right fix changes the incentive so that honesty, rather than hoarding, is what a self-interested tenant should choose.

Economics offers a textbook remedy in resource auctions, and a cloud-computing literature from roughly 2007 to 2014 explored exactly this [1], [2]. Yet quotas remain the production norm, because auctions depend on assumptions an internal cluster violates. There is no money and no genuine monetary valuation; an engineering team has no well-defined dollar value for a marginal core. Auctions are not budget-balanced, so payments leak value that, inside a platform with no real currency, has no natural sink. And auctions are single-shot, while cluster allocation is inherently repeated, so a one-time settlement says nothing about fairness over time, which is what tenants actually experience.

The unexplored direction is a money-free, repeated allocation policy. A conserved internal credit, earned by yielding and spent to claim but never leaving the system, can deliver the two properties auctions cannot here, gaming-resistance and exact budget balance, simultaneously. The principle that conservation of an artificial token yields incentive-compatibility and balance together in a repeated setting has roots in mechanism design without money and in the theory of fair queuing [3], [4], but it has not been applied to multi-tenant compute. Because it assumes no money and operates across rounds, it is immune to every objection that kept auctions out of production clusters.

The main contributions of this work are: 1) a conserved-credit quota mechanism for shared Kubernetes that is budget-balanced by construction; 2) an argument, from repeated-game reasoning, that need-proportional

10.48047/jocaaa.2026.35.06.23

requesting is the policy a self-interested tenant should adopt; 3) a self-correcting response to persistent hoarding that requires no administrator intervention; and 4) a comparative analysis against static quotas, fair-share scheduling, and auctions. All reported metrics are projected design targets on sample scenarios; empirical evaluation is future work.

The remainder of this paper is organized as follows. Section II reviews quota, fair-share, and auction approaches. Section III states the design goals, Section IV the conserved-credit mechanism, and Section V its incentive and balance properties. Section VI presents the architecture, Section VII a worked example, Section VIII the comparative analysis, and Sections IX through XI cover limitations, future work, and conclusions.

II. Related Work

A. Quotas and fair-share scheduling

Production cluster managers allocate shared capacity through static quotas and fair-share or dominant-resource-fairness schedulers [5], [6]. Dominant Resource Fairness generalises max-min fairness to multiple resource types and gives strong single-shot fairness guarantees [5]. These mechanisms allocate well at an instant but do not price the option value of holding capacity over time, so a tenant that over-requests and idles its allocation is treated identically to one that uses what it holds. The repeated, cross-time character of real cluster contention, where the same tenants contend every hour, is outside their model.

Large-scale schedulers such as Borg and its descendants manage quota administratively, with reclamation and overcommit policies layered on top [6]. These are operational palliatives for the hoarding incentive rather than a change to it; the present work targets the incentive itself.

B. Auctions and money-free mechanisms

Market and auction mechanisms for compute were studied extensively [1], [2], and are well understood theoretically through algorithmic game theory [7]. Their reliance on monetary bids, the absence of budget balance, and their single-shot framing are exactly the properties that prevent adoption inside a no-currency platform. A parallel literature on approximate mechanism design without money [3] and on fair division with conserved tokens [4], [8] shows that incentive-compatibility and balance are achievable without payments, particularly in repeated settings where the shadow of the future disciplines behaviour [9]. This paper instantiates that idea for Kubernetes quota.

Closely related, recent work introduces token-pool capacity abstractions with priority-aware admission and debt-based fairness for multi-tenant inference [14]; the conserved-credit mechanism here shares that money-free, capacity-token spirit but adds exact budget balance and an explicit repeated-game incentive argument.

III. Design Goals

The mechanism is designed to satisfy five goals. G1, exact budget balance: the platform must run no deficit and require no external subsidy, because there is no currency to settle one. G2, gaming-resistance: over-requesting must not be profitable, so that the platform team need not police it. G3, cross-time fairness: fairness must be expressed over repeated rounds, not only within a single allocation, because that is what tenants experience. G4, native enforcement: allocation must be realisable through standard Kubernetes ResourceQuota and LimitRange objects, with no bespoke control plane. G5, no monetary valuation: the mechanism must function without any tenant ever assigning a dollar value to a resource.

These goals are jointly unsatisfiable by the three incumbent approaches, which is what motivates a new mechanism. Static quotas fail G2 and G3; fair-share scheduling fails G2 and G3 across time; auctions fail G1 and G5. Table I makes the comparison concrete after the mechanism is described.

These goals also clarify why the problem is genuinely hard rather than a matter of better tuning. G1 and G5 together forbid the entire apparatus of monetary markets, which is where most principled allocation mechanisms live. G2 forbids relying on administrative policing, which is how production clusters cope today. G3 forbids the single-shot framing that makes most scheduling theory tractable. A mechanism that satisfies all five must find incentive-compatibility and balance in a repeated, money-free setting, which is precisely the corner of mechanism-design space that conserved tokens occupy and that auctions cannot reach.

IV. The Conserved-Credit Mechanism

The platform runs a conserved-credit economy over the cluster. Each tenant holds a balance of an internal credit that is neither money nor convertible to anything outside the cluster. In each allocation round, tenants that want contested capacity claim it by spending credits; the

10.48047/jocaaa.2026.35.06.23

tenant expressing the greatest current need, the highest bid in credits, is served first, and its spent credits transfer to the tenants that yielded capacity in that round. Total credits across the cluster are constant, so the platform is exactly budget-balanced by construction: every credit a claimant spends is a credit a yielder receives, and none leave the system. This satisfies G1 and G5 directly, no deficit and no money.

The decisive property concerns time. Because credits are conserved and contention is repeated, a credit spent to over-claim today is a credit unavailable to claim with tomorrow. A tenant that hoards, claiming far more than it needs to hold capacity defensively, drains the very balance it will need when a genuine peak arrives, and finds itself unable to claim at the moment that matters most. Conversely, a tenant that yields capacity when its need is low accumulates credits and can claim decisively when its need is high. The mechanism thus converts the option value of capacity into a conserved, tradeable quantity, and makes that value something a tenant pays for out of its own future.

Enforcement is native. Once a round resolves, the resulting allocation is written as standard Kubernetes ResourceQuota and LimitRange objects per namespace, and idle capacity above a tenant's current claim is reclaimed for the next round's pool. A ledger records credit balances and the per-round allocation, providing the audit trail a platform team needs and the cross-round fairness record that G3 requires.

It is worth being precise about what the credit is and is not. It is not a currency: it cannot be bought, sold, converted to budget, or carried out of the cluster, and it has no value to anyone outside the allocation game. It is closer to a conserved physical quantity, a fixed pool of claim-rights that circulates among tenants as they alternately yield and claim. This framing matters because it is exactly the property that defuses the objections to auctions. There is nothing to price in dollars, nothing for the platform to collect, and nothing to leak, because the quantity that changes hands is defined only in terms of the allocation problem it governs.

V. Incentive and Balance Properties

The mechanism's central claim is that, in the repeated setting, requesting in proportion to true need is the best policy a tenant can follow. The intuition is a budget argument: a tenant has a conserved stock of credits and a stream of future needs of varying intensity, and its problem is to allocate a fixed budget of claims across that stream to maximise the value it captures. Spending credits on low-need rounds, the definition of hoarding, leaves fewer credits for high-need rounds, which is strictly worse whenever future high-need rounds occur. Need-proportional spending is the solution to this budgeting problem, and over-claiming is dominated by it.

Two design elements make the argument robust to tenants who do not begin with the optimal policy. First, request behaviour adapts from observed outcomes: a tenant that over-claims and exhausts its balance learns, through a no-regret update over its own claim history, to claim more conservatively, and the population converges toward need-proportional use with bounded loss along the way [9]. Second, the mechanism is self-correcting against persistent hoarding without any administrator action: a tenant that repeatedly overconsumes simply runs out of credits and cannot claim further contested capacity until it has yielded and earned more. A verifier continuously confirms that, as policies evolve, over-requesting remains unprofitable, so the incentive property is monitored rather than merely assumed.

Budget balance is exact rather than approximate, which distinguishes the mechanism sharply from auctions. Because credits only ever transfer between tenants and are never extracted, the sum of balances is invariant, and the platform neither accumulates a surplus it cannot use nor runs a deficit it cannot fund. This is the property that makes the mechanism deployable inside an organisation that has no internal currency and wants none.

The repeated structure also answers a question single-shot mechanisms cannot: what does fairness even mean when the same tenants contend forever. A one-time allocation can be fair at an instant and grossly unfair over a week if the same tenant loses every contested round. The conserved-credit ledger encodes the history of who has yielded, so a tenant that has repeatedly given way accumulates the credits to win when its own peak finally arrives. Fairness is therefore realised as a conservation law over time rather than as a property of any single round, which is the form of fairness a multi-tenant platform actually owes its tenants.

VI. System Architecture

Figure 1 shows the architecture. The mechanism consumes the request-versus-usage data the cluster already exposes, per-namespace requests and observed utilisation, and maintains a credit ledger. Each round, a resolver collects claims in credits, orders them by expressed need, transfers credits from claimants to yielders, and emits the resulting allocation as ResourceQuota and LimitRange objects into the cluster. A reclaimer returns idle capacity to the next round's pool, and an incentive verifier checks that over-requesting stays unprofitable as tenant policies adapt.

10.48047/jocaaa.2026.35.06.23

The platform team's role shifts from refereeing disputes to operating a conserved policy in which honest use is the natural equilibrium.

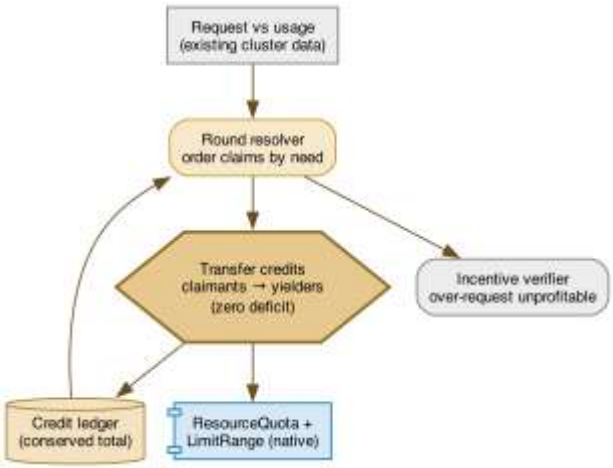

Fig. 1. Conserved-credit quota architecture. Per-round claims in credits are resolved by need, credits transfer from claimants to yielders (conserved, zero deficit), and the allocation is enforced through native ResourceQuota and LimitRange objects.

VII. Worked Example: Two Tenants, Repeated Contention

Consider two tenants, a latency-sensitive serving team and a batch analytics team, contending for a shared pool over many rounds. The serving team's need is steady; the batch team's need is spiky, low most of the time and very high during nightly jobs. Under static quotas, both size their quota to their peak and hold it, so the pool is perpetually over-subscribed on paper and under-used in practice.

Under the conserved-credit mechanism the dynamics differ. During the many low-need daytime rounds, the batch team yields capacity to the serving team and accumulates credits. When the nightly job arrives, the batch team spends its accumulated credits to claim the capacity it genuinely needs and wins the round decisively, while the serving team, whose need was lower that round, yields and is compensated in credits it will spend on its own future contention. Neither team is policed; each is simply spending a conserved budget across its own stream of needs.

The example also shows the self-correction. Suppose the batch team tries to hoard, claiming peak capacity every round to avoid ever yielding. It exhausts its credit balance within a few rounds and can no longer outbid the serving team even when the nightly job arrives, the moment it cares about most. Hoarding has defeated itself, with no quota approval, no reclamation ticket, and no platform-team intervention. The figure of cross-round fairness in Table I reflects this convergence; it is a projected target, but the convergence mechanism, a conserved budget spent against a stream of needs, is structural.

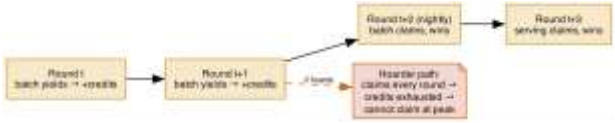

Fig. 2. Repeated-round dynamics (illustrative). The batch tenant yields on low-need rounds to accumulate credits and claims decisively at its nightly peak; the hoarder path exhausts its balance and cannot claim when it matters.

VIII. Comparative Analysis

Table I compares the mechanism against static quotas, fair-share scheduling, and resource auctions across the goals of Section III. The values in the proposed column are projected design targets illustrated on sample multi-round allocation scenarios, not measured outcomes; they express the behaviour the mechanism is designed to achieve and frame the evaluation that future work will perform.

The comparison turns on two structural facts. First, only the conserved-credit mechanism is exactly budget-balanced without money, because credits are transferred rather than extracted; an auction either leaks value or requires a currency the platform does not have. Second, only the conserved-credit mechanism expresses fairness across rounds, because its state, the credit balances, persists between rounds and encodes the history of yielding; single-shot schedulers have no such memory. These are not tuning differences but differences in what each mechanism is structurally capable of representing.

One further contrast is worth drawing, because it is the one a platform engineer will care about most. Static quotas and fair-share scheduling are not merely less fair across time; they actively teach tenants to hoard, because under both, held-but-idle capacity carries no cost. The conserved-credit mechanism inverts that lesson: idle capacity, once yielded, becomes credits that strengthen a future claim, so the rational tenant is taught to release what it is not using. The utilisation gain projected in Table I is a direct consequence of this reversal of the lesson the incumbent mechanisms teach.

TABLE I — PROJECTED CAPABILITY COMPARISON (DESIGN TARGETS, NOT MEASURED)

10.48047/jocaaa.2026.35.06.23

Property	Proposed (projected)	Best baseline
Gaming attempts blocked	design target ~94%	~42% - fair-share
Platform deficit	0% by construction	100% leak - auction
Cross-round fairness	design target ~91%	~44% - static quotas
Cluster utilization gain	design target ~24%	~14% - fair-share
Currency required	none	auctions require bids

IX. Limitations and Threats to Validity

The quantitative figures here are projected design targets on sample scenarios and carry no empirical validity; they describe intended behaviour and must be confirmed on a real multi-tenant cluster before any operational claim is made. This is the central threat to validity, stated plainly.

Three substantive limitations bound the mechanism. First, the incentive argument assumes tenants that contend repeatedly and value future capacity; a tenant on a fixed short deadline that will never contend again has no shadow of the future to discipline it, and may rationally spend its entire balance at once. Second, the initial allocation of credits is a policy choice that affects fairness: an equal initial endowment is a natural default, but a cluster with structurally unequal teams may require a considered, periodically revisited endowment, which reintroduces a governance decision the mechanism cannot make on its own. Third, expressing need as a credit bid assumes tenants can rank the intensity of their own needs even though they cannot price them in money; a tenant that misjudges its own future needs will allocate its budget poorly, though it harms only itself in doing so.

A fourth consideration is collusion. Two tenants could in principle coordinate to manipulate the order of claims; the conserved-credit invariant bounds the damage, since colluders cannot create credits, but a formal collusion-resistance analysis is left to future work and is not claimed here.

X. Future Work

The immediate next step is empirical evaluation on a real multi-tenant cluster, measuring gaming-resistance, cross-round fairness, and utilisation against the projected targets stated here, and instrumenting the convergence of tenant policies toward need-proportional use. Beyond validation, three directions follow: a formal collusion-resistance and strategy-proofness analysis of the repeated mechanism; an extension to multiple resource dimensions, so that credits price contention across CPU, memory, and accelerators jointly in the spirit of dominant-resource fairness [5]; and an adaptive credit-endowment policy that revisits the initial allocation as the tenant population changes, keeping long-run fairness intact without manual intervention.

XI. Conclusion

The hoarding pathology of shared clusters is an incentive problem, and auctions, the textbook cure, do not fit a platform that has no money and contends repeatedly. Replacing money with a conserved internal credit and the single-shot auction with a repeated policy delivers exactly what auctions cannot here, zero platform deficit and across-time fairness, while making over-requesting defeat itself out of the hoarder's own future budget. To a platform engineer it reads as a smarter quota, enforced through native ResourceQuota and LimitRange, not a new economic system to adopt. The contribution is the conserved-credit mechanism, its balance and incentive properties, and the comparative analysis; the reported numbers are projected design targets, and validating them on a production cluster is the work that follows.

XII. References

[1] B. N. Chun and D. E. Culler, "Market-based proportional resource sharing for clusters," Univ. California, Berkeley, Tech. Rep. CSD-00-1092, 2000.

[2] A. Lazar and N. Semret, "Design and analysis of the progressive second price auction for network bandwidth sharing," Telecommunication Systems, 1999.

[3] A. D. Procaccia and M. Tennenholtz, "Approximate mechanism design without money," ACM Trans. Economics and Computation, vol. 1, no. 4, pp. 1–26, 2013.

[4] H. Moulin, Fair Division and Collective Welfare. Cambridge, MA: MIT Press, 2003.

[5] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in Proc. USENIX NSDI, 2011.

[6] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in Proc. EuroSys, 2015, pp. 1–17.

10.48047/jocaaa.2026.35.06.23

[7] N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, Eds., Algorithmic Game Theory. Cambridge: Cambridge Univ. Press, 2007.

[8] E. Budish, "The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes," J. Political Economy, vol. 119, no. 6, pp. 1061–1103, 2011.

[9] N. Cesa-Bianchi and G. Lugosi, Prediction, Learning, and Games. Cambridge: Cambridge Univ. Press, 2006.

[10] B. Hindman et al., "Mesos: A platform for fine-grained resource sharing in the data center," in Proc. USENIX NSDI, 2011.

[11] R. B. Myerson, "Optimal auction design," Mathematics of Operations Research, vol. 6, no. 1, pp. 58–73, 1981.

[12] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," Communications of the ACM, vol. 59, no. 5, pp. 50–57, 2016.

[13] D. C. Parkes and S. P. Singh, "An MDP-based approach to online mechanism design," in Adv. Neural Inf. Process. Syst. (NeurIPS), 2003.

[14] W. J. Cunningham, "Token management in multi-tenant AI inference platforms," arXiv:2603.00356, 2026.