

Taming Inference Workloads at Global Scale: Foundation Model Serving in Amazon Bedrock

Pratik Pankaj Raichura, Somu Perianayagam, Rama Krishna Sandeep Pokkunuri, and Divya Jain

Amazon Web Services

Seattle, USA

bedrock-paper@amazon.com

Abstract

Foundation model (FM) inference platforms must manage scarce accelerator capacity distributed unevenly across regions. They must also handle requests whose token consumption varies widely and may be revealed progressively during generation, while sharing capacity across workloads with different latency and throughput objectives. This paper describes the architecture of the Workload Management System (WLM) deployed in Amazon Bedrock, which serves millions of inference requests per minute across more than 30 regions. The architecture follows two principles. First, decouple request ingress from execution to form a logical execution pool within permitted geographic and model-placement scopes, with prompt-cache stability as a first-class routing constraint. Second, favor bounded inconsistency with defense in depth over synchronous global coordination. WLM comprises five cooperating subsystems: admission control, traffic classification, cross-region routing, request scheduling, and orchestration. Admission and routing decisions use local or cached state; quota consumption is reconciled periodically across regions, while routing weights are refreshed off the request path from regional capacity and health signals. We evaluate WLM using production observations, benchmark measurements, and controlled load tests comparing bounded-staleness with synchronous admission. The evaluation characterizes accelerator utilization, cache stability, request latency, priority behavior, throughput, request-path overhead, quota-enforcement accuracy, and behavior under selected failures.

CCS Concepts: • **Computer systems organization** → **Cloud computing**; **Distributed architectures**; *Availability*; • **Software and its engineering** → **Scheduling**; • **Networks** → *Traffic engineering algorithms*; • **Computing methodologies** → *Machine learning*.

Keywords: foundation model serving, workload management, cross-region routing, admission control, distributed

rate limiting, bounded staleness, fair scheduling, traffic classification, multi-tenant cloud systems, prompt caching, inference serving, large language models

ACM Reference Format:

Pratik Pankaj Raichura, Somu Perianayagam, Rama Krishna Sandeep Pokkunuri, and Divya Jain. 2026. Taming Inference Workloads at Global Scale: Foundation Model Serving in Amazon Bedrock. In *ACM SIGOPS 32nd Symposium on Operating Systems Principles (SOSP '26)*, September 29–October 02, 2026, Prague, Czech Republic. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3830418.3843855>

1 Introduction

At global scale, foundation model (FM) inference shares scarce, geographically uneven accelerator capacity among customers with different latency and throughput objectives. Requests may enter through any region, vary by orders of magnitude in resource consumption, and reveal output-token cost only progressively during streaming. The system must therefore decouple ingress from execution while preserving prompt-cache locality and enforcing cross-region rate limits without synchronous coordination on the request path.

This paper presents the design rationale, tradeoffs, and operational lessons from the Workload Management System (WLM) deployed in Amazon Bedrock. Bedrock processes millions of inference requests per minute across more than 30 AWS regions and hosts models whose accelerator capacity is concentrated in a subset of those regions. At the time of this study, Bedrock offered four service tiers that differ in capacity entitlement and latency tolerance. Provisioned Throughput uses model capacity reserved for a customer; On-Demand serves interactive requests from shared capacity; Flex serves non-time-sensitive requests from shared capacity at lower scheduling priority than On-Demand; and Batch runs asynchronous jobs that tolerate delayed completion.

Two architectural principles recur throughout WLM and explain many of our design choices.

1. *Decouple request ingress from execution, with prompt-cache stability as a first-class routing constraint.* Subject to model placement and geographic constraints, WLM selects an execution region from a logical global pool of accelerator capacity, where the destination scheduler applies customer and workload priorities. Decoupling



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

SOSP '26, Prague, Czech Republic

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2585-2/2026/09

<https://doi.org/10.1145/3830418.3843855>

reduces geographic fragmentation but can sacrifice regional prompt caches, requiring prefix recomputation at an accelerator cost proportional to prefix length; Section 7 measures that cost and its first-token effect. WLM therefore uses stable weights and orders routing by availability, prompt-cache hit rate, latency, and utilization.

2. *Favor bounded inconsistency over synchronous global coordination.* Exact global admission would require synchronously consulting current quota state, while globally current routing would query remote capacity and health, adding cross-region latency and dependencies to the critical request path. WLM instead decides from locally cached quota state and routing weights, reconciling consumption through a globally replicated store and refreshing weights asynchronously. Bounded inconsistency means temporary divergence within configured synchronization windows followed by deterministic reconciliation. It may briefly over-admit or route from stale state. Defense in depth bounds the consequences: quota reconciliation records excess as negative balances, weight refresh and circuit breakers correct stale routes, and downstream per-customer fairness, queueing, adaptive concurrency, and shedding contain transient excess.

Four request-path subsystems implement these principles—admission control, traffic classification, cross-region routing, and request scheduling—spanned by orchestration that sequences them, propagates metadata, maintains token accounting, and applies fallbacks. Section 2 describes their composition.

We present an architectural account in three parts:

- **Principles and rationale.** We explain ingress/execution decoupling within model-placement and geographic scopes, with prompt-cache stability, and bounded inconsistency with defense in depth.
- **Composition and tradeoffs.** We explain why established mechanisms are composed in this sequence, using propagated metadata, local or cached decisions, and asynchronous reconciliation, and analyze tradeoffs among availability, enforcement precision, cache stability, latency, accelerator utilization, and customer isolation.
- **Operational experience.** Production observations and benchmark measurements characterize the integrated fleet; controlled load tests compare bounded-inconsistency admission with synchronous admission under matched workloads, measuring request-path latency, enforcement accuracy, and behavior during selected failures.

2 Architectural Overview

WLM sits between the API gateway and the FM hosting layer. In a source region, requests reach WLM after authentication, authorization, and validation. WLM may send them elsewhere for execution, while admission and routing decide

from local or cached state reconciled asynchronously. Decoupling recovers idle regional capacity; keeping coordination off the request path preserves latency and availability. Figure 1 shows the source- and destination-region components.

Stages are ordered by their dependencies. Admission runs first to avoid work for requests that will be throttled and emits accounting context. Classification then labels the request STANDARD or ANOMALOUS, Cross-Region Intelligent Spillover (CRIS) selects an execution region permitted by geographic scope and model placement, and the destination scheduler applies per-customer fairness, service-tier and classification priorities, and adaptive concurrency before dispatch. Orchestration spans the sequence: it invokes each component, propagates context, reports and reconciles token consumption, and applies fallbacks.

Request context accumulates accounting, classification, and destination metadata so the scheduler enforces policy without repeating upstream decisions. Shared quota state is reconciled and routing state refreshed asynchronously, while scheduler queues, fairness state, and concurrency limits remain local to the destination region.

Fail-open means that a management-component failure does not itself fail inference: the orchestrator substitutes the best available local decision because an unavailable subsystem cannot do so itself. This behavior follows WLM’s second principle. While this temporarily weakens the relevant optimization—quota enforcement, anomalous-traffic deprioritization, routing freshness, or scheduler fairness—requests can continue to be served as long as the hosting layer remains available. Background reconciliation and refresh later restore quota and routing state. Requests can still be quota-throttled or shed when the hosting layer lacks capacity, in which case the scheduler acts on the hosting layer’s behalf.

3 Distributed Admission and Quota Accounting

Admission control regulates how much of the shared accelerator fleet each customer may consume. Each customer is assigned request-per-minute (RPM) and token-per-minute (TPM) limits for a model and service tier. These limits protect the shared fleet from any single customer’s demand while giving other customers predictable access to capacity. A quota may span multiple regions, so requests entering different regions consume a common budget.

Foundation-model inference imposes three requirements that distinguish admission control from traditional RPC services. First, every inference request passes through admission control, so the request path must remain short; we target a 99th-percentile latency below 10 ms. Second, decisions across regions must collectively enforce that common budget. Third, unlike conventional RPCs, how much an inference request will consume is not known when admission occurs.

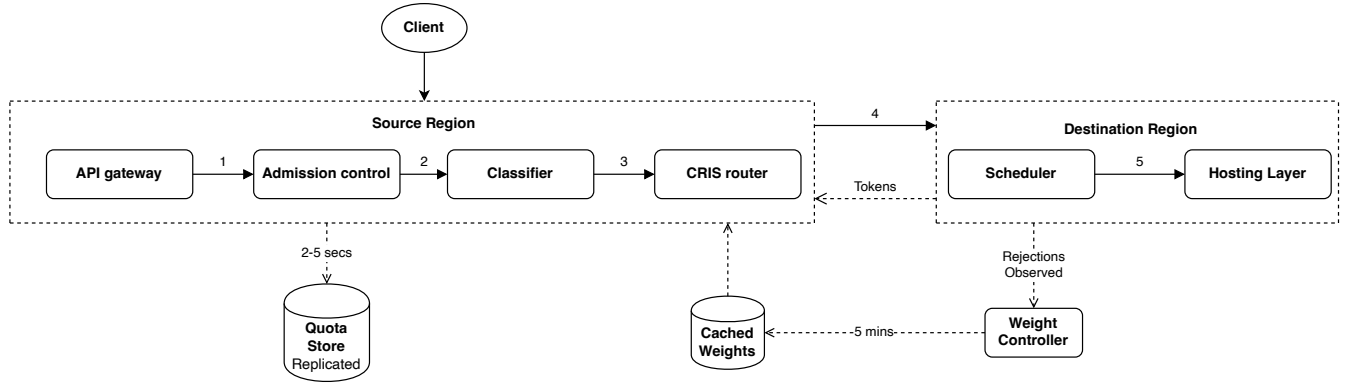


Figure 1. WLM request flow. Numbers order forward stages; dotted edges are asynchronous control and returning tokens.

The number of output tokens depends on model generation and, for streaming responses, is revealed progressively over the lifetime of the request. Admission control must therefore continue accounting for a request long after the initial decision.

These requirements cannot be satisfied with synchronous global coordination. Consulting globally current quota state before every admission decision would add at least one cross-region round trip. Such a round trip costs from roughly ten milliseconds between neighboring regions to hundreds of milliseconds between continents; Section 7 measures 60–170 ms in this fleet. Even the shortest is comparable to the entire 10 ms budget, and the common case exceeds it by an order of magnitude. Likewise, waiting until a streaming request completes before correcting its quota usage would allow inaccurate accounting to persist for minutes. WLM therefore realizes the bounded inconsistency defined in Section 1 through two independent paths: a *fast path* that decides locally from cached quota state without synchronous cross-region communication, and a *background accounting path* that continuously reconciles distributed quota state and corrects consumption during execution.

3.1 Quota Enforcement

Quota enforcement uses continuously refilled token buckets [23, 24]. Unlike fixed-window rate limiting, continuous refill eliminates boundary effects that would otherwise permit bursts approaching twice the intended rate across adjacent windows while still allowing short bursts within the bucket depth.

Every request is evaluated against two buckets: requests per minute (RPM) and tokens per minute (TPM). A quota key consists of the customer ID, model ID, quota scope, and service tier. Separate buckets are maintained for each limit associated with that key. A request is admitted only if sufficient allowance remains in every applicable bucket, after which its estimated token count is deducted from those buckets.

The initial deduction is necessarily an estimate, because the number of tokens a request will generate is unknown at admission time. WLM derives this estimate from the request input and model-specific heuristics. Because output length is not known in advance, the estimate can miss in either direction: too high understates the customer’s remaining allowance until reconciliation returns the difference, and too low lets the request run against an allowance that has not yet accounted for its full consumption. Buffered requests are reconciled once upon completion, while streaming requests periodically report cumulative token consumption throughout execution. A final reconciliation adjusts the buckets to the observed token usage and returns any difference. Admission control therefore accounts for long-running requests throughout their execution rather than only at admission.

3.2 Distributed Quota Accounting

Each admission-control instance maintains quota state in memory and makes admission decisions independently, avoiding synchronous storage operations on the request path. The challenge is ensuring that many independent instances collectively enforce one logical quota. That challenge exists within a single region: a region runs many instances, so even a regionally scoped quota requires shared accounting among them. A quota whose scope spans regions widens the set of instances that must agree rather than introducing a separate cross-region mechanism, and one replicated store carries both cases.

The key scalability constraint is not storing shared quota state but limiting how many instances modify it concurrently. A shared quota record supports only finite write throughput. Allowing every admission-control instance to update every quota would create storage hotspots and couple request latency to storage performance.

WLM instead assigns each quota to a small, stable subset of admission-control instances using consistent hashing [12]. This is a conventional application of consistent hashing, used

here to select writers rather than to place data: the mapping is deterministic, so every instance derives the same subset without consulting another, and it shifts little as the fleet grows or hosts are replaced. Only instances within this subset reconcile that quota's shared state, bounding both storage contention and the number of independent admission decisions that may diverge. The subset size is configurable on a per-quota basis. Small subsets improve enforcement precision, while larger subsets increase admission throughput for high-volume quotas.

Requests are consistently routed to one member of the responsible subset. As a result, the same instance performs the initial admission decision, receives subsequent streaming accounting updates, and performs the final reconciliation, preserving locality for both request state and quota accounting. A request may arrive at any instance, which derives the mapping locally and forwards it once to the chosen member; only after at least two failed attempts does the receiver decide from its own state, so local fallback remains a last resort rather than a routine path.

3.3 Background Reconciliation

Responsible instances periodically reconcile local quota state through Amazon DynamoDB [8]. Every synchronization cycle reads the latest replicated consumption, incorporates locally observed usage and token refills, updates the shared record using optimistic concurrency control, and refreshes the local cache. Synchronization runs every 2–5 seconds with randomized jitter to avoid synchronized write bursts.

The synchronization interval determines the tradeoff between enforcement precision and coordination cost. Shorter intervals reduce temporary quota divergence but increase write traffic to the shared store. Longer intervals reduce storage pressure while widening the interval over which temporary over-admission may occur. Replication latency provides a practical lower bound: the store documents propagation as typically completing within about a second and exposes per-region-pair replication latency as an operational metric [1], so synchronizing substantially faster than replication converges increases write traffic without producing a meaningfully fresher global view.

3.4 Bounded Divergence and Failure Handling

Admission control intentionally provides eventual rather than instantaneous global consistency. During normal operation, temporary quota overshoot is bounded by the reconciliation interval and the number of responsible admission-control instances assigned to a quota. Background reconciliation continuously corrects this divergence, restoring a globally consistent view of quota consumption.

When reconciliation cannot proceed because the replicated store is unavailable, admission control continues making decisions from locally cached state rather than rejecting inference requests. Each responsible instance independently

enforces its local quota until reconciliation resumes. Overshoot is then bounded by the size of the responsible subset: with N instances, summed across the regions where the quota is active, the admitted rate cannot exceed N times the limit. Subsetting is what keeps that factor small—had any instance in the fleet been permitted to decide, it would instead be the fleet size. Once the store returns, buffered accounting updates are reconciled and quota state converges automatically. This bound depends on admission control itself still running. When admission control cannot be reached at all, requests are admitted rather than failed, and nothing enforces a limit until it returns. Section 7 evaluates the latency, quota accuracy, and recovery behavior of this design against a synchronous baseline, and measures the divergence this bound permits within a single synchronization window.

4 Traffic Classification

Admission control limits how much of the shared accelerator fleet a customer may consume, but it does not distinguish steady demand from a sudden change in demand. A customer may remain well within its quota while increasing its request rate quickly enough to degrade latency for other customers before the platform can add capacity. Traffic classification addresses this gap by identifying workloads whose demand changes faster than the platform can adapt. The classifier labels each request `STANDARD` or `ANOMALOUS`. Here, `ANOMALOUS` denotes a workload whose current demand exceeds its own recent history by a configured multiple; it does not imply misuse or malicious intent. The label influences scheduling priority rather than admission. An `ANOMALOUS` request is deprioritized only when the destination region is contended and otherwise executes normally. Because a false positive delays work rather than rejecting it, the classifier deliberately favors sensitivity over precision.

4.1 Detecting Demand Changes

Inference workloads create contention in two distinct ways. A customer may submit a small number of requests that generate a large volume of tokens, consuming accelerator time, or it may submit many short requests that saturate request queues and dispatch pipelines. WLM therefore evaluates both token rate and request rate independently. Each signal is compared against an exponential moving average (EMA) [2, 21] of the customer's recent history. The EMA suppresses short-term noise while adapting as the workload's steady-state rate evolves. A request is classified as `ANOMALOUS` if either the token-rate or request-rate classifier exceeds its configured threshold. We combine the two classifiers using logical OR because the costs of the two errors are asymmetric: missing a disruptive burst can degrade every customer sharing the affected capacity, whereas a false positive lowers the priority of only one workload and only while the destination region is contended.

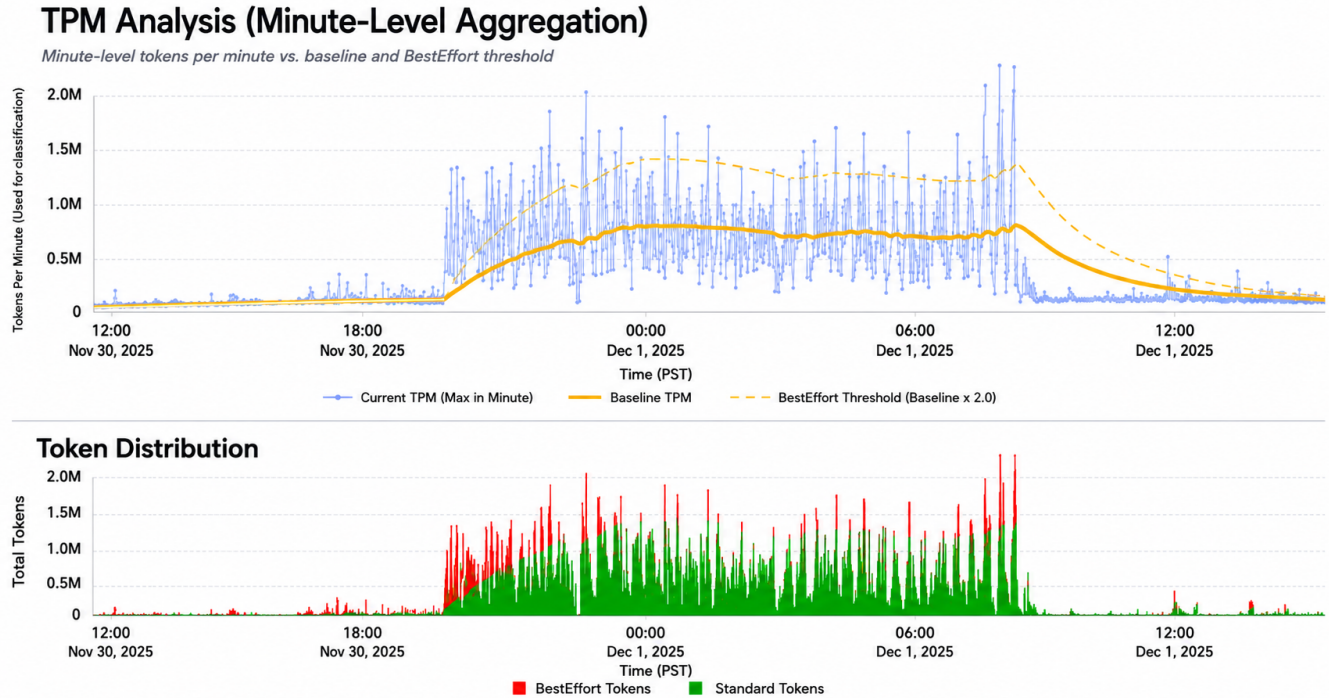


Figure 2. EMA-based traffic classification for a single customer over a 24-hour period

The EMA adaptation window is chosen to approximate the time required for the hosting layer to respond to increased demand. Consequently, the classifier detects demand changing faster than capacity can follow, rather than demand that is merely large. A sudden increase remains ANOMALOUS while the hosting layer provisions additional capacity or the baseline adapts. A gradual increase may never be classified, which is the intended behavior because capacity can grow alongside it. Absolute consumption remains bounded independently by admission control. This design also defines the subsystem’s scope. Traffic classification protects customers during transitions in demand; it does not solve permanent capacity shortages. If capacity cannot increase within the expected adaptation window—for example because a region or model has exhausted its accelerator supply—the baseline eventually adapts even though contention remains. At that point, admission control, per-customer fairness, service-tier priorities, and load shedding continue to isolate workloads.

4.2 Shared Accounting

The token-rate classifier requires the same information as admission control: an estimate of token consumption before execution and corrections as the response is generated. Rather than maintaining a second estimator and accounting pipeline, WLM reuses the admission-control accounting protocol. Model-specific heuristics provide the initial estimate, streaming usage reports update the customer’s current token rate during execution, and completion performs a final

reconciliation. Classification and quota enforcement therefore operate over the same accounting model while avoiding duplicate infrastructure. Because the initial deduction is an estimate, a request’s contribution to the token-rate signal is provisional until usage is reconciled. Consequently, a burst is typically recognized on the requests that follow it rather than on its first request. We accept this delay because the classifier is intended to protect the fleet from sustained demand changes rather than interrupt a single request already in execution. Because both decisions already read the same state, they could be returned by a single call, saving a hop on the request path. They remain independent services instead. If admission control fails open, requests may proceed without quota enforcement. If classification fails open, requests default to STANDARD. Keeping these decisions separate preserves independent failure modes rather than allowing a single failure to disable both controls simultaneously.

4.3 Classification Propagation

Classification executes at the ingress region because that is where customer behavior is observed before routing partitions requests across execution regions. A destination region sees only the subset of requests selected by CRIS, and that subset changes as routing weights evolve. A destination-local baseline would therefore reflect routing decisions as much as customer behavior. The classification label is attached to the

request metadata and propagated to the destination scheduler, which combines it with service-tier and customer context when ordering work. This preserves a judgment made where the complete customer workload is visible rather than reconstructing it from routing-shaped traffic at the destination. Classification baselines are maintained independently in each ingress region and, unlike quota consumption, are not replicated across regions. When a customer begins sending requests through a new ingress region, no historical baseline exists there, so the classifier initially labels the traffic ANOMALOUS. The label returns to STANDARD once the EMA establishes a baseline and the workload falls within the configured threshold. We chose this conservative default because a false negative during cold start can expose other customers to an unobserved traffic spike, whereas a false positive only lowers scheduling priority during contention; it neither rejects requests nor limits throughput when spare capacity exists. Replicating classification history across ingress regions would reduce this cold-start penalty, but would introduce another cross-region state stream for a control whose errors affect priority rather than admission.

Figure 2 shows the token-rate classifier operating on one customer’s traffic over 24 hours. The top panel plots the current token rate against the EMA baseline and the $2\times$ threshold used for this workload; the bottom panel colors the same traffic by the resulting label. The baseline tracks the workload’s diurnal shape rather than a fixed ceiling, so the same absolute rate is STANDARD at one hour and ANOMALOUS at another.

5 Cross-Region Intelligent Spillover (CRIS)

Accelerator capacity for a model is distributed unevenly across regions, and demand rarely follows the same distribution. Without cross-region routing, a request entering a saturated region may be throttled while capacity for the same model remains idle elsewhere. Cross-Region Intelligent Spillover (CRIS) realizes the first architectural principle of Section 1: it decouples a request’s execution region from its ingress region, selecting among the regions where the model is deployed and the request’s geographic scope permits execution.

Three properties complicate this decision. First, regional capacity changes as hosts fail, recover, or are rebalanced within and across regions. Second, prompt caches are regional, so moving a request may force prefix recomputation and reduce effective accelerator capacity through additional computation and cache churn. Third, geographically constrained and globally eligible requests share regional capacity, but only globally eligible traffic can move elsewhere when that capacity becomes scarce. CRIS must therefore adapt to capacity changes, preserve prompt-cache locality when movement is unnecessary, and protect capacity needed by workloads with narrower execution scopes.

CRIS applies these objectives in priority order: availability first, prompt-cache locality second, followed by latency and utilization. The order is fixed rather than blended into a single score, which makes conflicts resolvable without case-by-case judgment: CRIS accepts higher latency to keep a request with its warm cache, and abandons that cache when the alternative is failing the request.

5.1 Eligibility and Geographic-Scope Protection

A request may execute only where its model is deployed and its geographic scope permits processing. Model placement is discovered from a metadata store rather than configured statically, so newly provisioned capacity enters the routing pool automatically and removed capacity leaves it. Hosting systems decide where model capacity is placed; CRIS decides how requests are divided among eligible regions.

Geographically constrained and globally eligible requests have asymmetric alternatives. Both may consume capacity in the same region, but global traffic can move elsewhere while constrained traffic cannot. A single utilization-based distribution can therefore allow movable traffic to consume capacity needed by a narrower scope.

CRIS computes separate routing distributions. Constrained traffic is distributed within its eligible scope according to model capacity. Global traffic is distributed according to residual headroom, steering it away from regions under pressure and preserving local capacity for traffic with fewer alternatives.

For region r , headroom is the capacity remaining below a target utilization, $H_r = \max(0, C_r \cdot U_{\text{target}} - L_r)$, where C_r is model capacity and L_r is observed load; the clamp at zero means a region already above its target draws no global traffic. The target remains below saturation to absorb bursts and delays in utilization and placement signals. A geographic penalty reduces the effective headroom of distant regions, favoring nearby capacity while still allowing distant spillover when local headroom is exhausted.

5.2 Off-Path Weight Computation

CRIS separates global planning from per-request placement. A controller in each ingress region periodically collects model placement, capacity, utilization, and dispatch outcomes for eligible destinations. It computes a distribution describing how traffic entering that region should be divided and publishes the result to a regional store. Routers serve the distribution from a host-local cache.

Cross-region signal collection therefore remains off the request path. A request uses precomputed local state rather than synchronously querying remote regions. Controllers do not need to agree on one global distribution. Each controls only the traffic entering its own region. Their weights may differ because geographic penalties depend on the ingress region, while failures remain isolated because one controller can stop updating without preventing other regions from

routing. CRIS combines two measures of destination pressure. Reported utilization is predictive and allows traffic to move before a region begins rejecting work. Dispatch failures are authoritative and capture overload or operational faults not reflected in utilization; because they are observed on the router's own traffic, they remain available even when the reporting path goes stale or silent. Together, they let the controller react proactively while retaining an independent check on whether requests are actually being served, applying the defense in depth of Section 1 to the controller's inputs.

5.3 Routing Stability and Prompt-Cache Locality

Prompt caches [9] are local to a region. Moving requests that share a prefix may replace a cache hit with prefix recomputation, increasing service time and consuming accelerator capacity; Section 7 measures that recomputation cost and the rate at which reuse survives. Repeated movement can also create cache churn as newly inserted prefixes displace existing entries. CRIS therefore treats routing stability as a capacity concern. For requests with cache affinity, the router receives a sticky-routing key derived from the prompt prefix. It uses weighted consistent hashing on this key so related requests remain mapped to the same region while regional weights control the fraction of keys assigned to each destination.

Changing weights remaps part of the key space and abandons the corresponding warm caches. The controller therefore moves weights gradually toward each target and suppresses updates when the expected benefit does not justify the locality cost. Damping alone preserves the sum; what breaks it is a region leaving the eligible set after the targets were computed, which leaves the remaining weights short of one. Restoring the sum by scaling them back up is where the controller's intent can be lost, because naive normalization can raise a region whose target weight decreased. CRIS records whether each region should increase, decrease, or hold and constrains normalization not to reverse that direction. Residual weight is redistributed only among regions that can continue moving consistently with their targets. The controller skips updates when all regions have sufficient headroom, when the aggregate change is too small to justify remapping, or when every region is overloaded and redistribution cannot create capacity.

5.4 Request Placement

Weights determine candidate order rather than placing a request directly. Requests without a sticky-routing key use weighted random ordering. Requests with a sticky-routing key use weighted consistent hashing to preserve affinity while respecting the configured regional proportions. CRIS issues no separate synchronous probe. It dispatches to the first candidate with a queue-delay tolerance determined by the request's service tier. The destination scheduler accepts

the request or declines it when its expected delay exceeds that tolerance, after which the router tries the next candidate.

A successful first attempt adds only the dispatch needed for execution; a declined remote attempt adds one cross-region round trip, exposing stale or inaccurate weights as increased time to first token. If all candidates decline, CRIS applies a service-tier terminal queue bound: affinity requests wait in their preferred region, while others wait where expected delay is lowest. The scheduler may still reject when that bound is exceeded.

5.5 Failure Handling and Metadata Propagation

A circuit breaker tracks repeated dispatch failures and elevated latency for each destination. After multiple failed attempts cross a configured threshold, it temporarily removes the region from the candidate set, reducing incoming CRIS traffic while the region recovers.

Routing continues if a controller or signal source becomes unavailable. Routers use the last published distribution rather than rejecting requests because fresh weights cannot be computed. The cost of that choice is that weights which stop being recomputed grow stale, steering traffic using a capacity picture that no longer holds. Destination declines and circuit breakers provide request-path correction until discovery and weight computation converge, but they act only once a misdirected request has already paid the round trip; Section 8 returns to this. Traffic-classification labels and service-tier metadata accompany the request across the regional boundary. The destination scheduler therefore preserves the customer fairness, traffic-classification priority, and service-tier policy established at ingress.

6 Request Scheduling

An admitted, classified, and routed request reaches a destination scheduler that orders accelerator work and applies dispatch, delay, preemption, and refusal through hosting-layer mechanisms; it owns no capacity. Queueing absorbs brief spikes, and per-customer fairness limits how customer surges affect others. Ordering lets delay-tolerant work use capacity not needed by latency-sensitive work, avoiding fixed partitions that idle when demand falls short.

6.1 Sharding and Failure Recovery

Regional queues are divided into shards, each owned by one lease-holding leader. A metadata store records leadership with conditional writes, preventing a former leader from committing after its lease is lost. A leader renews every 200 ms against a one-second lease; before another instance can claim it, the lease has gone five renewals without refresh. This threshold balances two recovery risks: waiting longer strands requests behind a failed leader, while replacing sooner can displace a live leader, pause dispatch, and trigger repeated leadership changes. Leaders do not hand

off; the former leader stops renewing, and subsequent writes fail.

Orchestrators poll the scheduler for requests awaiting completion. A new leader reconstructs interactive queues from those polls without a recovery read: waiting callers represent requests, while requests whose callers have left disappear; asynchronous and long-running work has no caller and remains durably stored. Dispatch resumes with the first poll because no complete-queue signal exists. Original arrival times preserve exact ordering among requests seen, but membership remains incomplete until callers re-poll roughly a second later. Waiting would idle accelerators and delay every queued request. A shard recovers in seconds. If the regional scheduler is unavailable, the orchestrator dispatches directly to the hosting layer, preserving inference but losing fairness, priority, and adaptive concurrency.

6.2 Per-Customer Fairness

Every dispatch decision determines whose work waits. Fairness limits how much regional capacity one customer can hold, while priority selects the class served first. Stochastic fair queueing (SFQ) [14] hashes customers into a fixed number of round-robin queues, bounding state but making customers in the same queue contend. Periodic seed rotation makes collisions between heavy customers temporary.

6.3 Service-Tier and Classification Priority

Service tier and traffic classification jointly set priority from P_0 to P_k : Provisioned Throughput and On-Demand outrank opportunistic Flex and Batch, while within On-Demand, STANDARD outranks ANOMALOUS. Deficit Round-Robin (DRR) [19] gives every level a dispatch share, delaying lower-priority work without indefinite starvation; ANOMALOUS requests wait behind STANDARD requests rather than being rejected. Metadata from ingress preserves this policy after cross-region spillover. Preemption can reclaim occupied capacity: when higher-priority demand arrives, existing priority levels select the lowest-priority running Batch request, which the hosting layer interrupts and requeues; the scheduler alone cannot stop accelerator work.

6.4 Adaptive Concurrency

A static concurrency setting derived from provisioned capacity can become stale. The scheduler instead discovers an outstanding-work level the hosting layer currently sustains using the additive-increase/multiplicative-decrease (AIMD) feedback structure from congestion control [5, 11], adapted here with proportional rather than fixed increases and severity-scaled reductions, while configured bounds around provisioned capacity constrain the search range. When hosting accepts work, the limit grows as $C_{t+1} = C_t + \lceil \alpha C_t \rceil$; when hosting throttles or latency rises, it falls as $C_{t+1} = C_t \times \beta$, with $\beta < 1$ scaled by throttle severity.

Overshoot causes queueing, rejection, and retries that can lose prompt-cache hits; undershoot idles accelerators until subsequent increases, so the loop favors temporary underutilization. Increases also require sufficient utilization and completed requests, preventing upward drift on sparse traffic.

Because feedback determines the operating point within those bounds, the loop adapts to host, software, and model-server throughput changes. Upward probing needs headroom, one reason the routing targets in Section 5 remain below saturation.

6.5 Load Shedding

Queueing absorbs bursts but not sustained excess. When hosting refusals persist and a shard's queue exceeds a configured depth, the scheduler stops accepting new requests for that queue, preventing waits beyond caller tolerance and retry-generated load. It delays shedding while transient pressure may clear; if pressure persists, it sheds ANOMALOUS before STANDARD traffic and opportunistic tiers before either. Section 5's weight controller observes these refusals as the region's rejection rate.

7 Evaluation

Evaluation combines three evidence types. Controlled load tests reconfigure the same fleet to compare local bounded-inconsistency admission with a single global authority consulted synchronously; parameters, workloads, and failure injections are set deliberately. Public-endpoint benchmarks measure prompt-cache reuse and time to first token; production observations cover routing convergence, cross-region round-trip latency, priority scheduling, and adaptive concurrency. Benchmarks and production observations characterize the integrated fleet without isolating mechanisms. Table 1 summarizes the parameters used in these results; the referenced sections discuss their rationale and sensitivity.

7.1 Admission Control

We evaluate bounded-staleness admission against a synchronous baseline using replayed production traffic on production infrastructure. The bounded-staleness design makes admission decisions from local state and reconciles periodically; the synchronous design consults a single authority in one region for every request. We use two topologies, each with three responsible instances per region: a two-region topology with $N = 6$ independent deciders and a three-region topology with $N = 9$. For each topology, 1,000 globally scoped accounts are driven under request-rate-constrained load. Sustained five-minute phases sweep offered load from $0.8 \times$ quota up to $N \times$ quota. A separate one-second burst test completes before reconciliation and directly tests the $N \times$ short-term overshoot bound from Section 3. Synchronization

Parameter	Value
<i>Admission control</i> (§3)	
Synchronization interval	5 s, 2 s under load
Responsible instances, N	3 per region
<i>Traffic classification</i> (§4)	
EMA smoothing, α	0.01 per active minute
Token-rate anomaly multiple	2.0× baseline
Request-rate anomaly multiple	1.5× baseline
<i>Cross-region routing</i> (§5)	
Weight recomputation	every 5 min
<i>Request scheduling</i> (§6.1)	
Shard leader lease	1 s, renewed every 200 ms
<i>Adaptive concurrency</i> (§6.4)	
AIMD increase, α	0.0266, about +2.7%
AIMD decrease, β	0.85–0.95, severity-scaled

Table 1. WLM parameters in force for the reported results

is configured for 5 s, with early synchronization allowing the effective interval to fall to 2 s under high load.

Latency is independent of distance. Bounded-staleness admission remained at 1–2 ms median latency in every region, with a 99th percentile below 3.4 ms. In the three-region topology, synchronous median latency instead tracked distance to the authority: approximately 2, 60, and 211 ms across the three regions. Server-side instrumentation at the authority showed the same split, attributing the added latency to the cross-region round trip rather than queueing. Median latency within a region changed by less than 1 ms across a 15-fold range of offered load. Only bounded staleness remained within the 10 ms request-path budget from Section 3.

Sustained enforcement converges to quota. Under sustained overload, both designs converged to the configured limit. Across both topologies and all reported overload levels, final-phase admitted load remained within 1.1% of quota. Bounded staleness therefore changes short-term enforcement without increasing long-run entitlement.

Short-term divergence scales with the number of deciders. Before reconciliation, each of the N responsible instances makes decisions from its own view of the quota. In the worst case, all N instances can each admit up to one bucket before learning what the others admitted, giving a short-term ceiling of N times the configured allowance. We test this bound with one-second bursts that finish before reconciliation can propagate admissions among instances. Table 2 reports the admitted volume per account in multiples of one bucket. With bounded staleness, admitted volume follows the offered burst: three bucket-equivalents of demand result in 3.00 admitted buckets, while bursts at the predicted ceiling result in 5.92 of 6 buckets for $N = 6$ and 8.65 of 9

for $N = 9$. Synchronous admission, by contrast, remains near one bucket because every request observes the same authority. These results confirm that the maximum short-term overshoot is determined by the number of independent deciders. Values slightly above 1.0 for synchronous admission result from token-bucket refill during the one-second test. After reconciliation, excess consumption is recorded as a negative balance and subsequent requests are throttled until refill repays it. For a one-minute bucket, a worst-case $N \times$ burst can therefore require up to roughly $N - 1$ minutes to repay, but it does not increase the customer’s long-run quota.

Offered in 1 s	Bounded staleness	Synchronous	Ceiling
<i>Two regions, $N = 6$</i>			
1 bucket	1.00	1.00	1
3 buckets	3.00	1.01	3
6 buckets (N)	5.92	1.02	6
<i>Three regions, $N = 9$</i>			
9 buckets (N)	8.65	1.04	9

Table 2. Admitted buckets per account within one synchronization window

Instance loss preserves availability. In a separate failure injection, requests targeting terminated responsible instances were retried and redirected after failed attempts. Receivers then decided from local state rather than returning errors. We observed no client-visible failures, and replacement instances were provisioned within approximately 120 s, after which normal coordinated reconciliation resumed.

7.2 Cross-Region Routing

Adaptation after capacity changes. We first measure how quickly CRIS adapts after regional capacity changes. The controller recomputes weights every five minutes; in production, published weights converged to within 1% of a new distribution in about 45 minutes. In one high-traffic deployment, the controller adjusted weights nine times over ten days without operator intervention. Stale routing state has a measurable cost: production cross-region round trips were 60–170 ms, so a declined remote dispatch incurs one such round trip before retry and may also lose prompt-cache locality.

Locality under weight recomputation. We next measure whether periodic weight changes disrupt sticky routing. A black-box benchmark using a public cross-region inference profile replayed 6,000 documents from the PMC Open Access Subset [17], with prefixes from 1,024 to roughly 50,000 tokens, sending each once cold and five additional times. For prefixes of at least 2,048 tokens, reuse succeeded on 99.94% of repeats in a five-hour run and 99.78% in a full-day run; the corresponding failure rates were 0.060% and

0.222%. Because the serving region is not externally observable, a prompt-prefix reuse failure may reflect either routing movement or cache eviction; these failure rates are therefore upper bounds on routing-induced locality loss. A separate run spaced repeated requests ten minutes apart, crossing multiple controller recomputation intervals between uses. Reuse succeeded on 99.86% of 4,992 checks, with no observed degradation as requests crossed successive weight updates. The runs differ in cache lifetime, load, and warm-set size, so we do not compare their reuse rates causally.

Cost of losing locality. Finally, we quantify the latency benefit of prompt reuse. The prefix-dependent component of time to first token fell by 45%, from 33.9 to 18.5 ms per thousand prefix tokens. In the full-day run, a warm prefix reduced median client-observed time to first token by 10.7%, from 1,583 to 1,413 ms, and the 99th percentile by 61%, from 8,687 to 3,397 ms. These measurements cover one model, account, and workload, but they quantify the penalty that CRIS avoids by preserving prompt-cache locality.

7.3 Priority Scheduling and Classification Effects

We evaluate whether priority scheduling is work-conserving when capacity is available and whether service tiers and classification labels produce the intended separation under contention. The classifier used $\alpha = 0.01$ for both EMA baselines, updated once per active minute, with anomaly thresholds of $2.0\times$ for token rate and $1.5\times$ for request rate. The window tracks capacity-provisioning time: smaller α responds more slowly, while lower thresholds catch smaller changes but flag more ordinary variation. Intermittent traffic stretches the window, and classification weakens if capacity cannot grow before the baseline adapts.

Work conservation. Before priority-aware scheduling and preemption, Batch was capped at approximately 15% of hosting capacity. With WLM, Batch consumed 70–100% of capacity that would otherwise have remained idle and was preempted when higher-priority demand returned. Weekly Batch throughput increased by approximately $7\times$ during the six-month rollout that removed the static cap and enabled work-conserving preemption. Because workload demand, fleet configuration, and other capabilities changed during the same period, this longitudinal result does not isolate the scheduler’s contribution.

Priority under contention. Figure 3 shows queueing consistent with the configured priority order during periods of contention. Provisioned Throughput (pr10) and Standard On-Demand (pr50) see little queueing in this trace, while Anomalous On-Demand (pr70), Flex (pr90), and Batch (pr150) wait during peaks. The separation between STANDARD and ANOMALOUS on-demand traffic reflects classification priority; Flex and Batch receive lower priority because of their service tiers. Figure 4 shows the same effect in availability during

a capacity-constrained period: STANDARD traffic retained higher availability than ANOMALOUS traffic. These measurements evaluate how the scheduler acts on assigned labels, not whether the classifier assigned those labels correctly.

7.4 Adaptive Concurrency

We evaluate whether the scheduler adjusts concurrency in response to changing hosting-layer pressure without an explicit capacity estimate. The production controller uses $\alpha = 0.0266$ (about 2.7% per accepted decision) and a severity-scaled β between 0.85 and 0.95. Selected in pre-launch load tests, these defaults have not been retuned. Higher α probes faster but risks overshoot; lower β relieves pressure faster but can leave capacity idle longer. Figure 5 shows a production trace of the concurrency limit and hosting-layer feedback. The limit increases while requests are accepted and decreases when throttling or elevated latency indicates pressure. The trace therefore shows that the controller reacts to downstream conditions using feedback alone, without a model-specific capacity signal. This measurement establishes responsiveness of the control loop, but not optimality of the selected limit, because the hosting layer’s instantaneous sustainable capacity is not directly observable.

7.5 Behavior During Failures

We examine three failures to identify which workload-management protection is lost while inference continues. When quota synchronization to the replicated store was throttled, admission instances continued deciding from cached state but could not reconcile consumption across deciders; no request failed because of the store condition itself. When the classifier’s in-memory store became unavailable, requests defaulted to STANDARD, preserving inference while disabling anomaly deprioritization. When a regional scheduler became unavailable, circuit breakers removed the region from routing candidates and requests shifted to other regions within seconds, at the cost of additional cross-region latency and possible loss of prompt-cache locality.

Across these cases, fail-open preserved inference availability by temporarily dropping the failed subsystem’s protection rather than preserving full WLM semantics. The quota case was observed during controlled admission tests on production infrastructure under replayed traffic; the classifier and scheduler cases are production incidents.

8 Limitations

WLM trades strict global coordination for low request-path latency and availability. This permits temporary over-admission during reconciliation; when admission control is unavailable, requests may proceed without quota enforcement. Coordinated multi-region bursts can exploit this temporary staleness, though downstream classification, fairness,

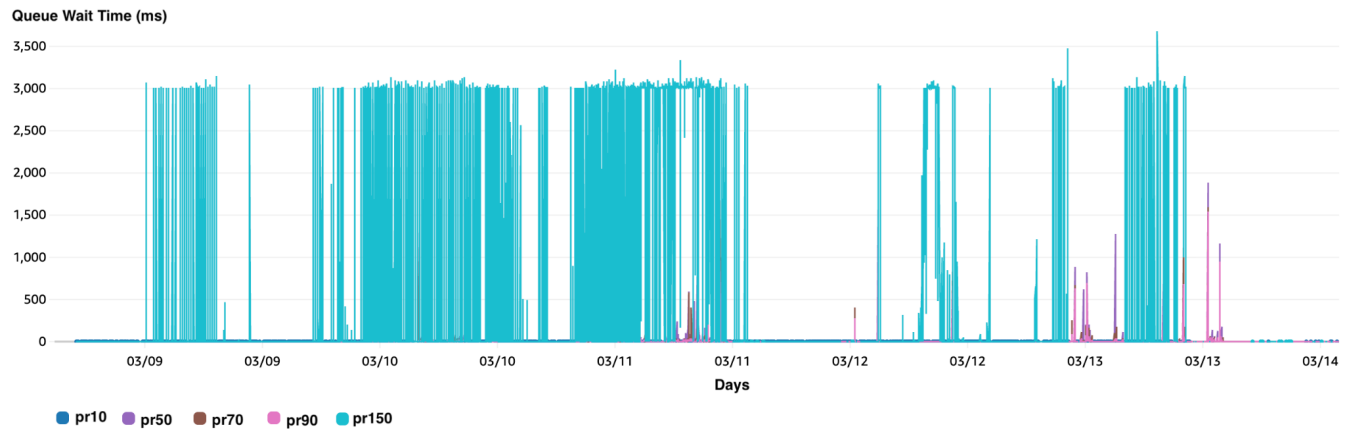


Figure 3. Queue wait time by scheduling priority over a five-day period

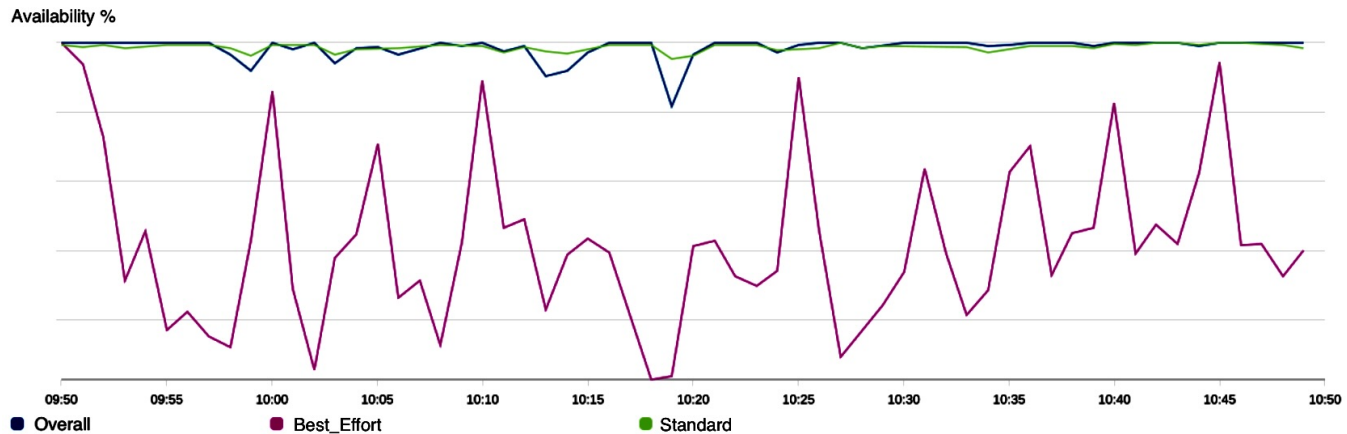


Figure 4. Availability by traffic classification during a capacity-constrained period

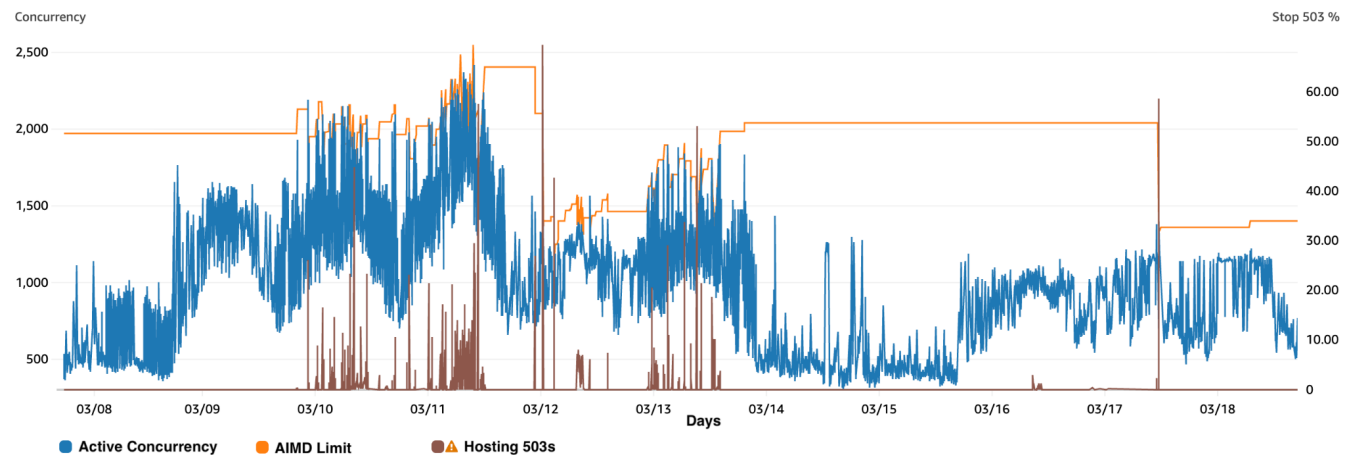


Figure 5. Adaptive concurrency limit over time

adaptive concurrency, and shedding may prevent some admitted excess from reaching execution. Excess can be repaid through future allowance, but accelerator capacity already consumed cannot be recovered. Selectively tightening coordination for high-risk quotas or periods of sustained contention is one path to reducing this exposure.

Token accounting remains approximate until generation reveals actual consumption. Agentic workloads are especially difficult to estimate: tool use and iterative generation make output length highly variable, allowing concurrent requests to outrun both quota and classification corrections. Better online token prediction and more frequent in-flight accounting could reduce this gap without requiring synchronous admission.

Classification baselines are regional, avoiding another global state stream but potentially assigning otherwise similar requests different priorities when their ingress histories differ. Selectively sharing compact baseline state across regions could reduce cold-start and regional-history effects while preserving local classification.

Finally, preserving prompt-cache locality slows routing adaptation. Damped weight changes reduce cache churn but allow routing state to lag capacity changes, shifting some correction to request-time retries. Future routing controllers could incorporate richer cache and destination feedback to move weights faster when the expected capacity gain outweighs the locality cost.

9 Related Work

Token buckets [23, 24] and distributed rate limiters [10, 18] regulate aggregate consumption across multiple enforcers. WLM differs primarily in what must be accounted: inference requests consume both requests and tokens, and final token consumption is unknown at admission and revealed progressively during execution. WLM therefore makes local admission decisions and reconciles consumption through replicated state, accepting bounded short-term divergence rather than adding synchronous cross-region coordination to every request.

Anomaly detection can regulate unwanted traffic [4, 15, 21], while Breakwater [6] controls overload from server feedback. WLM instead classifies changes from each customer’s history to set scheduling priority, without inferring intent or determining admission. Its scheduler uses fair queueing [14, 19] for customer shares and AIMD [5, 11] for capacity control. Singularity [20] schedules preemptible distributed training; FastServe [25] and other serving systems [13, 22, 26, 27] optimize inference within one cluster. WLM instead selects execution regions and carries quota and scheduling policy from ingress to execution.

Geographic and randomized load balancing [3, 16] distribute traffic across destinations, while systems such as Maglev [7] use affinity to preserve connections under backend

churn. WLM preserves a different form of locality: moving a request can discard reusable prompt state, requiring accelerator computation to reconstruct it. Prompt Cache [9] makes such state reusable; WLM’s routing policy attempts to keep it reusable while adapting to regional capacity. This couples global routing to an inference-specific resource cost that does not arise in conventional geographic load balancing.

10 Conclusion

WLM enables globally distributed FM inference without placing synchronous cross-region coordination on every request. Its two architectural principles appear as three recurring design ideas: decisions use local state, policy travels with each request, and temporary inaccuracies are corrected continuously. Admission control reconciles estimated and actual token consumption, routing adapts cached regional weights using asynchronous signals and destination outcomes, and schedulers absorb transient excess through fairness, priority, queueing, and load shedding. Prompt-cache locality remains part of the routing policy because moving work can itself consume accelerator capacity through recomputation and cache churn. Our evaluation shows that this design preserves low request-path latency while maintaining quota enforcement and adapting to changing regional capacity, and that it can reclaim otherwise fragmented capacity for lower-priority workloads. More generally, the experience suggests that global FM serving does not require a choice between regional fragmentation and globally serialized control: local decisions can compose into useful global behavior when their errors are bounded, policy is preserved across execution boundaries, and the system includes explicit mechanisms for correction.

Acknowledgments

Bedrock has benefited greatly from its customers whose continuous feedback pushed us to innovate on their behalf. We have been lucky to have an amazing team working with us on this journey. We thank Marc Brooker, Swami Sivasubramanian, Rajesh Sheth, Anthony Liguori, and the entire Bedrock team for impactful contributions. We are grateful to the engineers who designed, built, and operate the systems described in this paper, including Jeremy Allston, Jan Catarata, Emily Jung, Yuvasri Nandagopal, Ujjwal Prakash, Vasudev Purandare, Richard Scherrer, and Allen Yuan. Many people have helped improve this paper, and we thank the anonymous reviewers who helped shape the paper.

References

- [1] Amazon Web Services. 2025. Amazon DynamoDB global tables: multi-active, multi-Region replication. <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GlobalTables.html>.
- [2] R. G. Brown. 1963. *Smoothing, Forecasting and Prediction of Discrete Time Series*. Prentice-Hall.

- [3] V. Cardellini, M. Colajanni, and P. S. Yu. 1999. Dynamic load balancing on web-server systems. *IEEE Internet Computing* 3, 3 (1999), 28–39.
- [4] V. Chandola, A. Banerjee, and V. Kumar. 2009. Anomaly detection: A survey. *Comput. Surveys* 41, 3 (2009), 1–58.
- [5] D. Chiu and R. Jain. 1989. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems* 17, 1 (1989), 1–14.
- [6] I. Cho, A. Saeed, J. Fried, S. J. Park, M. Alizadeh, and A. Belay. 2020. Overload control for μ s-scale RPCs with Breakwater. In *Proc. OSDI*.
- [7] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilengiroglu, B. Cheyney, W. Shang, and J. D. Hosein. 2016. Maglev: A fast and reliable software network load balancer. In *Proc. NSDI*.
- [8] M. Elhemi et al. 2022. Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service. In *Proc. USENIX ATC*.
- [9] I. Gim, G. Chen, S.-s. Lee, N. Sarda, A. Khandelwal, and L. Zhong. 2024. Prompt Cache: Modular attention reuse for low-latency inference. In *Proc. MLSys*.
- [10] Google. 2016. Doorman: Global distributed client-side rate limiting infrastructure. <https://github.com/youtube/doorman>.
- [11] V. Jacobson. 1988. Congestion avoidance and control. In *Proc. SIGCOMM*.
- [12] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. 1997. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. STOC*.
- [13] W. Kwon et al. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proc. SOSP*.
- [14] P. E. McKenney. 1990. Stochastic fairness queueing. In *Proc. IEEE INFOCOM*.
- [15] J. Mirkovic and P. Reiher. 2004. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review* 34, 2 (2004), 39–53.
- [16] M. Mitzenmacher. 2001. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems* 12, 10 (2001), 1094–1104.
- [17] National Center for Biotechnology Information. 2026. PMC Open Access Subset. <https://pmc.ncbi.nlm.nih.gov/tools/openftlist/>. Snapshot retrieved August 11, 2026.
- [18] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren. 2007. Cloud control with distributed rate limiting. In *Proc. SIGCOMM*.
- [19] M. Shreedhar and G. Varghese. 1996. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on Networking* 4, 3 (1996), 375–385.
- [20] D. Shukla, M. Sivathanu, S. Viswanatha, B. Gulavani, R. Nehme, A. Agrawal, et al. 2022. Singularity: Planet-scale, preemptive and elastic scheduling of AI workloads. <https://arxiv.org/abs/2202.07848>. arXiv:2202.07848.
- [21] V. A. Siris and F. Papagalou. 2006. Application of anomaly detection algorithms for detecting SYN flooding attacks. *Computer Communications* 29, 9 (2006), 1433–1442.
- [22] B. Sun et al. 2024. Llmunix: Dynamic scheduling for large language model serving. In *Proc. OSDI*.
- [23] A. Tanenbaum and D. Wetherall. 2011. *Computer Networks* (5th ed.). Pearson.
- [24] J. Turner. 1986. New directions in communications (or which way to the information age?). *IEEE Communications Magazine* 24, 10 (1986), 8–15.
- [25] B. Wu, Y. Zhong, Z. Zhang, S. Liu, F. Liu, Y. Sun, G. Huang, X. Liu, and X. Jin. 2023. Fast distributed inference serving for large language models. <https://arxiv.org/abs/2305.05920>. arXiv:2305.05920.
- [26] G. Yu et al. 2022. Orca: A distributed serving system for Transformer-based generative models. In *Proc. OSDI*.
- [27] Y. Zhong et al. 2024. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proc. OSDI*.