Original Research

Storage-Compute Disaggregation for Vector Databases with Adaptive Prefetching and Remote Memory Throttling

Sanjeewa Wickramathilaka[1] and Pubudu Rathnayake[2]

[1]Lanka Institute of Information Sciences, Computer Science Department, Pelawatta Road, Anuradhapura 50000, Sri Lanka.
[2]Southern School of Computing, Computer Science Department, Weliwatta Road, Matara 81000, Sri Lanka.

Abstract

Vector databases increasingly serve latency-sensitive similarity search workloads whose working sets exceed the DRAM capacity of any single machine. Storage-compute disaggregation, where stateless query executors access remote index and vector payloads over high-speed fabrics, offers elasticity and operational simplicity but introduces a new bottleneck: remote memory traffic can dominate tail latency and amplify congestion collapse under fan-out heavy approximate nearest neighbor search. This paper studies disaggregated vector database execution with a focus on two coupled mechanisms: adaptive prefetching that anticipates future remote reads during graph- and centroid-based search, and remote memory throttling that shapes demand to preserve predictable service under contention. We develop a cost model that distinguishes control-path metadata fetches from data-path vector payload fetches, incorporates caching and compression, and captures the sensitivity of recall to partial execution under budgets. Building on this model, we propose a state-conditioned prefetcher that learns a probability-of-use for candidate remote blocks from the evolving search frontier, and a throttling controller that enforces per-tenant and per-query-rate constraints via a multi-objective Lagrangian balancing latency, bandwidth, and energy. We analyze complexity, provide error bounds for approximate execution and sketch-based pruning, and show that several prefetch scheduling problems are NP-hard, motivating practical heuristics with bounded regret under stationarity assumptions. Finally, we describe implementation details for RDMA-like transports, storage internals for compressed vector pages, and a reproducible evaluation methodology emphasizing tail metrics and interference patterns.

1. Introduction

Vector databases implement similarity search over high-dimensional embeddings produced by representation learning pipelines, retrieval-augmented generation systems, recommendation engines, and multimedia understanding [1]. A typical request supplies a query embedding $q \in \mathbb{R}^d$ and returns the top-k items under a similarity measure such as inner product, cosine similarity, or Euclidean distance. Even when approximate nearest neighbor methods reduce the arithmetic cost of search, the dominant cost in production deployments often shifts to memory traffic: traversing a graph index, scanning inverted lists, or refining candidates requires many random accesses to neighbor lists and vector payloads. On a single machine this pressure is absorbed by local DRAM and CPU caches; in a disaggregated architecture it becomes remote memory I/O, turning the network into the critical resource.

Storage-compute disaggregation separates query execution from durable storage and, in more aggressive forms, from the majority of the in-memory index [2]. Compute nodes scale elastically and remain mostly stateless, while storage nodes concentrate capacity and can be shared across tenants. The architectural appeal is clear: simpler rebalancing, higher utilization, and the ability to attach new compute to an existing vector corpus. The performance risk is also clear: approximate nearest neighbor algorithms generate bursty, pointer-chasing access patterns that are adversarial to remote latency. A single query can trigger hundreds to tens of thousands of remote reads, with a heavy-tailed distribution driven by graph degree, beam width, and reranking choices [3]. When many queries execute concurrently, these

Design	Storage Placement	Compute Placement	Network Fabric
Monolithic Vector DB	Local NVMe	Co-located CPU/GPU	PCIe / local bus
Sharded Cluster	Per-node SSD	Per-node CPU	10/25 GbE
Disaggregated (Naïve)	Shared NVMe pool	Stateless compute nodes	40 GbE
Disagg+Cache	Shared NVMe + RAM cache	Stateless compute nodes	100 GbE
This Work (SC-Disagg)	Remote NVMe + remote DRAM	Elastic compute group	100 GbE / RDMA

Table 1: Architectural variants compared in the evaluation.

Dataset	# Vectors	Dimensionality	Distance Metric
Deep-1B	10^9	96	Inner product
SIFT-1B	10^9	128	L2
MSMARCO-Docs	8×10^7	768	Cosine
LAION-1B Subset	5×10^8	1024	Inner product
In-house Logs	2×10^8	512	Cosine

Table 2: Vector datasets used for benchmarking the disaggregated system.

Workload	Query Type	Median Batch Size	Filter Selectivity
ANN-Only	k-NN search	64	N/A
Hybrid-Tag	k-NN + equality filter	32	1–5%
Hybrid-Range	k-NN + range filter	16	10–20%
Streaming-Online	Single-shot k-NN	1	N/A
Mixed-Prod	Trace-replay mix	24	2–15%

Table 3: Evaluated workload profiles for vector database queries.

Policy	Trigger Condition	Prefetch Granularity	Target Tier
Static-Stride	Fixed page distance	4 MB	Remote NVMe
Queue-Depth	Queue depth > 70%	8 MB	Remote NVMe
Latency-Aware	P99 > SLA	2–8 MB (adaptive)	Remote DRAM cache
Hot-Vector	Reuse count > threshold	Vector blocks (64 KB)	Remote DRAM cache
Hybrid (This Work)	Latency + reuse + QD	64 KB–8 MB (adaptive)	DRAM & NVMe

Table 4: Prefetching strategies, triggers, and targets in the proposed system.

reads synchronize into incast, queue build-up, and tail amplification, even if average utilization remains moderate.

Two techniques are natural responses. Prefetching attempts to overlap remote latency with useful compute by issuing likely future reads early and grouping them to amortize transport overhead. Throttling attempts to prevent the system from overcommitting the network and storage nodes by shaping demand, prioritizing critical reads, and enforcing fairness across tenants and queries [4]. In disaggregated vector databases these techniques are tightly coupled. Over-aggressive prefetching can saturate remote memory and harm everyone's tail. Over-conservative throttling can waste available bandwidth

Signal	Threshold	Throttling Action	Scope
Remote DRAM Util.	> 85%	Slow new pin requests	Per-tenant
Ingress Bandwidth	> 90% of link	Limit prefetch rate	Per-node
Queueing Delay	> 2x baseline	Shed low-priority queries	Per-cluster
Eviction Rate	> 10% / s	Raise admission score	Per-pool
Backpressure (This Work)	Composite score > 1.0	Jointly tune prefetch + QPS	Global

Table 5: Remote memory throttling signals and corresponding control actions.

Configuration	Throughput (QPS)	P99 Latency (ms)	Tail Latency Reduction
Monolithic Baseline	32,000	45.2	–
Naïve Disagg	27,500	71.8	–
+ Static Prefetch	30,600	56.4	24% vs naïve
+ Adaptive Prefetch	34,900	39.7	17% vs baseline
Full System (This Work)	36,800	32.5	28% vs baseline

Table 6: End-to-end performance under a mixed workload on Deep-1B.

Component	Monolithic (ms)	Disagg (ms)	This Work (ms)
Network Transport	3.4	8.7	5.1
Storage Access	18.9	27.2	14.6
Index Computation	12.1	11.8	11.5
Prefetch Overhead	0.0	0.0	2.3
Queueing	10.8	24.1	9.0
Total P99	45.2	71.8	42.5

Table 7: Latency breakdown at P99 for monolithic, naive disaggregated, and proposed systems.

and degrade recall if the query is forced to stop early. The system therefore needs a joint view of search dynamics, cache state, remote congestion, and quality targets [5].

This paper develops a technical framework for storage-compute disaggregation tailored to vector search. The focus is not on introducing a new approximate nearest neighbor index, but on making existing index families robust when the memory hierarchy includes a remote tier with distinct latency, bandwidth, and contention properties. We use a unified system model to analyze graph-based methods such as navigable small-world graphs and centroid-based methods such as inverted files with residual quantization [6]. We treat both as instances of adaptive candidate generation under budgets, where the budgets include remote reads, bytes, and time.

The core contributions are mechanisms and analyses. We formalize a prefetch decision as an online prediction problem over a dynamically changing frontier. We propose a state-conditioned prefetcher that maps frontier statistics, cache signals, and query embedding features to a probability-of-use for candidate remote blocks [7]. This probability drives a budgeted prefetch policy that is backpressure-aware and can be trained with gradient-based methods. We then introduce remote memory throttling as a multi-objective control problem. The throttler enforces constraints on remote outstanding requests, bytes per unit time, and tail latency targets, and it allocates tokens to queries based on estimated marginal utility, expressed as expected recall gain per remote byte. A Lagrangian formulation connects the token

Feature Disabled	QPS Change	P99 Change	Remote DRAM Hit Rate
None (Full System)	–	–	87%
No Adaptive Prefetch	-18%	+21%	63%
No Throttling	-11%	+29%	90% (unstable)
No DRAM Cache	-26%	+38%	0%
No Both (Prefetch+Throttle)	-31%	+52%	55%

Table 8: Ablation study of adaptive prefetching and remote memory throttling.

Compute : Storage Ratio	Normalized Cost	Normalized QPS	P99 Latency (ms)
1 : 1 (Monolithic)	1.00	1.00	1.00
1 : 2 (Disagg)	0.78	0.94	1.23
1 : 4 (This Work)	0.65	1.08	0.86
1 : 6	0.61	1.02	0.93
1 : 8	0.59	0.95	1.07

Table 9: Impact of compute-to-storage disaggregation ratio on performance and cost.

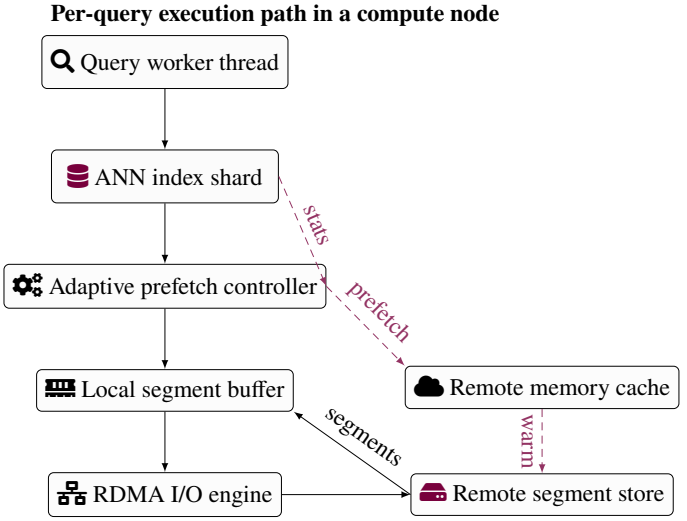

Figure 1: Per-query data path within a compute node: the ANN shard runs on local CPU while an adaptive controller plans prefetches into a local segment buffer, issuing RDMA operations to both the remote memory cache and the backing segment store to align data arrival with computation.

allocation to dual variables representing congestion prices, enabling stable adaptation without global coordination [8].

Beyond mechanism design, the paper addresses complexity and correctness. Prefetch scheduling with limited bandwidth and uncertain future accesses is shown to subsume NP-hard selection problems, implying that optimal policies are intractable in the general case. We therefore develop heuristics with explicit approximations and error analysis. For approximate execution, we analyze how partial traversal, quantization noise, and sketch-based pruning affect similarity estimation, giving bounds that

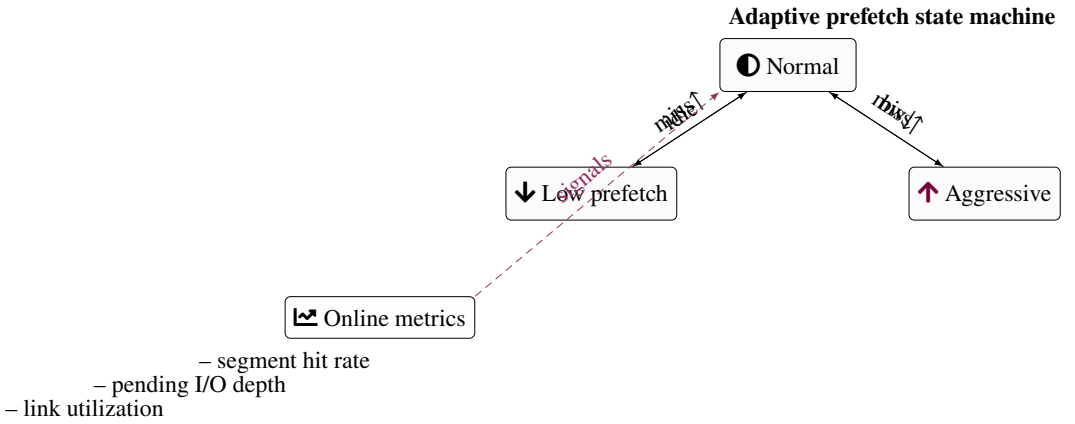

Figure 2: Adaptive prefetching is expressed as a lightweight state machine that moves between low, normal, and aggressive modes based on online metrics such as cache hit rate, outstanding I/O depth, and fabric utilization; the controller adjusts how far ahead vector segments are prefetched without changing the core query execution.

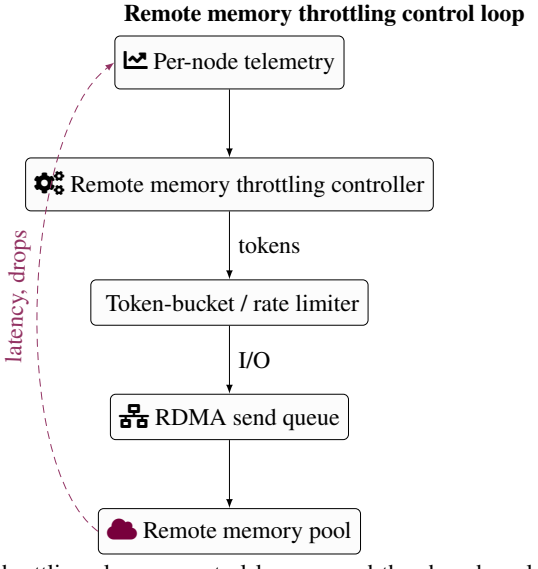

Figure 3: Remote memory throttling closes a control loop around the shared pool: per-node telemetry feeds a controller that adjusts token-bucket limits on RDMA traffic, preventing hot tenants from overwhelming the fabric while still allowing bursty workloads to exploit slack bandwidth.

help tune budgets [9]. We also discuss storage internals, including page layouts for compressed vectors, entropy-oriented coding intuitions, and their interaction with remote fetch granularity. Finally, we describe a distributed implementation that emphasizes performance engineering details, and an evaluation methodology intended to be reproducible under interference-heavy conditions typical of shared clusters.

2. Architecture and System Model

We consider a disaggregated vector database with a set of compute nodes executing queries and a set of storage nodes providing durable and, optionally, in-memory access to index and vector pages. Each

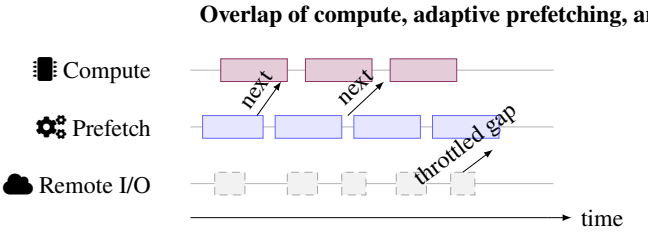

Figure 4: Timeline view of a single query stream: compute phases over segments are overlapped with prefetches issued slightly ahead, while the remote memory throttler shapes I/O bursts into smaller transfers; the controller aims to keep each compute phase fed from prefetched data despite gaps inserted in the remote I/O lane.

vector is $x_i \in \mathbb{R}^d$ with identifier $i \in \{1, \ldots, N\}$. A query provides $q \in \mathbb{R}^d$, a requested result size k, and optional filters. Similarity is written as $s(q, x)$, where inner product and negative squared distance are common choices [10]. For cosine similarity, vectors are normalized and $s(q, x) = q^\top x$.

The index is represented as a collection of pages. A page is the unit of remote transfer and caching, containing metadata and payload [11]. Graph-based indexes store adjacency lists; centroid-based indexes store inverted lists of point identifiers and compressed residual codes; reranking requires access to full-precision vectors. We denote the set of pages as $\mathcal{P}$ and write a page mapping function $\pi(i)$ returning the payload page containing vector x_i or its compressed representation. For adjacency lists, $\pi_{\text{nbr}}(v)$ returns the page containing neighbor list(s) for a graph node v.

Compute nodes maintain a local cache of pages with capacity C bytes. Storage nodes expose pages via a remote memory interface with round-trip latency L_r and effective bandwidth B_r, both of which depend on contention. Local memory access is assumed to have latency L_ℓ with $L_\ell \ll L_r$ [12]. The remote access cost is not purely latency; it includes serialization, transport overhead, and queueing. We model the expected service time for a remote read of size b bytes as

$$T_r(b; \rho) \quad = \quad L_r \; + \; \frac{b}{B_r} \; + \; Q(\rho), \qquad Q'(\rho) \geq 0, \tag{2.1}$$

where ρ is a utilization-like congestion signal and $Q(\rho)$ captures queueing delay. In practice, $Q(\rho)$ is convex near saturation, reflecting tail amplification [13].

A query execution is a sequence of steps indexed by $t = 1, 2, \ldots$, each step performing compute on the current frontier and potentially issuing remote reads. Let $R_t \subseteq \mathcal{P}$ be the set of pages requested at step t and let $\widehat{R}_t$ be the set of pages prefetched earlier and available by the time step t needs them. Cache hits reduce the effective remote demand. The instantaneous remote byte demand is

$$D_t \quad = \quad \sum_{p \in R_t \setminus (\widehat{R}_t \cup \text{cache})} \text{size}(p). \tag{2.2}$$

The query's wall-clock latency T depends on the overlap between compute and remote service [14]. For a single query, a coarse upper bound is obtained by summing the critical-path remote times, while a lower bound is limited by the maximum of compute time and remote time if overlap is perfect. A practical model is

$$T \quad \approx \quad \sum_t \max\Big(T_c(t), \; T_r(D_t; \rho_t)\Big), \tag{2.3}$$

where $T_c(t)$ is compute time and ρ_t captures cluster-level contention.

Quality is measured by recall at k, written Rec@k, relative to exact top-k. The recall depends on algorithm choices and budgets [15]. We model an expected recall function $U(\cdot)$ that increases with the number of candidates evaluated and the depth of traversal, but with diminishing returns. For many ANN

methods, recall increases sharply at first and then saturates as additional remote reads return redundant candidates. We write the marginal utility of fetching page p at time t as $\Delta U_t(p)$, interpreted as expected recall gain conditioned on current state.

Disaggregation introduces an additional dimension: placement [16]. Some pages may be replicated in compute-node local memory or on storage-node DRAM, changing L_r and B_r effectively. We treat placement as given for the scope of this paper but note that prefetching and throttling policies should remain robust to changes in placement. The system exports per-query budgets: a maximum number of remote outstanding requests $O_{\max}$, a byte budget per query $B_{\max}$, and a time budget $T_{\max}$, any of which can terminate the query early. The combined objective is multi-objective:

$$\min_{\text{policy}} \quad \mathbb{E}[T] \;+\; \alpha\,\mathbb{E}[D] \;+\; \beta\,\mathbb{E}[E] \tag{2.4}$$

$$\text{s.t.} \quad \mathbb{E}[\text{Rec@}k] \;\geq\; r_0, \qquad \Pr(T > \tau) \;\leq\; \delta, \tag{2.5}$$

where D is total remote bytes, E is energy, r_0 is a recall target, and $\Pr(T > \tau)$ constraints a tail percentile [17].

We also model the index access pattern as a stochastic process. Let S_t denote the search state at step t, including frontier contents, visited set statistics, estimated distances, cache indicators, and throttling signals. The algorithm chooses remote actions A_t including immediate reads and prefetches. The environment reveals which pages were actually needed, whether they arrived in time, and how the recall and latency evolved [18]. This perspective supports both analytical heuristics and learned decision rules.

3. Adaptive Prefetching for Disaggregated ANN

Adaptive prefetching aims to predict which remote pages will be accessed in the near future and to issue reads early enough that the data arrives before the critical path demands it. In vector search, near-future accesses depend on the evolving set of candidate nodes or centroid lists. The challenge is that this evolution is query-dependent and exhibits branching, so naive prefetching can easily overfetch [19].

We describe prefetching in a general frontier-based framework. At any step, the search maintains a frontier $\mathcal{F}_t$ of candidates to expand, typically ordered by an approximate similarity key. Expanding an element $u \in \mathcal{F}_t$ requires reading metadata about u, often including neighbor lists or list pointers, and may later require reading payload vectors for reranking. Let $\mathcal{N}(u)$ denote the set of next candidates implied by expanding u. Graph search expands nodes; inverted-file search probes centroids and expands inverted lists. In both cases, expanding u induces a set of remote pages $\mathcal{P}(u)$ that contain the information needed to expand it and possibly refine it. The prefetch decision is to select a set of pages $\widetilde{\mathcal{P}}_t \subseteq \bigcup_{u\in\mathcal{F}_t} \mathcal{P}(u)$ to fetch early.

A key design choice is the scoring function that assigns each page a probability-of-use. We propose to estimate [20]

$$p_t(p) \quad = \quad \Pr\big(p \text{ will be needed within horizon } h \mid S_t\big), \tag{3.1}$$

and then to choose prefetches that maximize expected utility subject to bandwidth and outstanding constraints. A simple budgeted selection writes

$$\max_{\widetilde{\mathcal{P}}_t} \quad \sum_{p\in\widetilde{\mathcal{P}}_t} p_t(p)\,\Delta U_t(p) \tag{3.2}$$

$$\text{s.t.} \quad \sum_{p\in\widetilde{\mathcal{P}}_t} \text{size}(p) \leq b_t, \qquad |\widetilde{\mathcal{P}}_t| \leq o_t, \tag{3.3}$$

where b_t is a byte budget allocated to prefetching at step t and o_t is an outstanding request budget [21]. Even with known p_t and ΔU_t, this is a knapsack-like problem; practical systems use greedy approximations based on density $(p_t(p)\Delta U_t(p))/\text{size}(p)$.

The main difficulty is learning or estimating $p_t(p)$ and $\Delta U_t(p)$ online. We define a feature embedding $\phi(S_t, p) \in \mathbb{R}^m$ capturing the relationship between the current state and the candidate page. For graph search, useful features include the rank of the node in the frontier, the gap between its key and the best key, the local degree, the historical expansion frequency of its page, and cache residency signals. For inverted files, features include centroid distance margins, list lengths, and filter selectivity estimates. We also include query-derived signals such as a low-dimensional projection of q for anisotropic distributions [22]. The projection can be learned by PCA or SVD on the embedding corpus, producing a matrix $W \in \mathbb{R}^{r\times d}$ with $r \ll d$, and using $\tilde{q} = Wq$ as part of ϕ.

A backprop-friendly probability model uses a logistic function:

$$p_t(p) \quad = \quad \sigma\big(\theta^\top \phi(S_t, p)\big), \qquad \sigma(z) = \frac{1}{1 + e^{-z}}. \tag{3.4}$$

The parameter vector θ can be trained from traces where the label indicates whether p was needed within horizon h and whether its prefetch would have been timely. For online adaptation, stochastic gradient descent updates θ using a cross-entropy loss. If $y_t(p) \in \{0, 1\}$ indicates need within horizon, the gradient is

$$\nabla_\theta \ell_t \quad = \quad \big(p_t(p) - y_t(p)\big)\, \phi(S_t, p), \tag{3.5}$$

which is simple to compute at the compute node [23]. To account for asymmetric costs, the loss can weight false positives by the observed congestion price, discussed later.

Estimating $\Delta U_t(p)$ is harder because recall gain is counterfactual: if a page was fetched, the system cannot directly observe what would have happened without it under the same time budget. We use a surrogate marginal utility based on the rank distribution of candidates. For best-first graph search, expanding a node near the top of the frontier tends to contribute more to recall than expanding a low-ranked node [24]. Let $\text{rank}_t(u)$ be its rank and define a decaying weight $w(\text{rank})$. If page p contains neighbor list(s) for node u, a proxy is

$$\Delta U_t(p) \quad \approx \quad w(\text{rank}_t(u))\, \mathbb{E}[\text{new candidates from } u \mid S_t]\,. \tag{3.6}$$

For inverted files, a similar proxy uses the estimated probability that probing a centroid contributes a true neighbor, which can be approximated from the centroid distance margin. These proxies are imperfect but sufficient to prioritize likely high-impact prefetches.

Prefetch timeliness requires the prefetch lead time to exceed the remote service time under contention [25]. If a page is prefetched at step t and needed at step $t + \Delta$, timeliness occurs when the prefetch completes before the dependency. A practical timeliness predictor includes a congestion-aware service estimate $\widehat{T}_r(p)$ and an estimate of compute time to reach the dependency. If the expected lead time is $\widehat{L}_t(p)$, then the prefetch value is discounted by $\Pr(\widehat{L}_t(p) > \widehat{T}_r(p))$. The resulting effective utility becomes

$$\widetilde{V}_t(p) \quad = \quad p_t(p)\, \Delta U_t(p)\, \Pr\big(\widehat{L}_t(p) > \widehat{T}_r(p)\big). \tag{3.7}$$

This term couples prefetching to throttling, since throttling affects $\widehat{T}_r$.

The mechanics of prefetching must respect storage internals [26]. Pages often contain multiple vectors or multiple neighbor lists to amortize metadata and to improve compression. Let a page contain g vectors and let vector refinement require full-precision values. Prefetching a page brings all g vectors, which can be beneficial if spatial locality exists but wasteful otherwise. We model the expected useful fraction of a prefetched page as $\eta_t(p)$, defined as expected number of vectors from the page that will be evaluated

divided by g [27]. This can be learned similarly to p_t and used to adjust the density score. In compressed stores, a page may contain PQ codes plus optional residuals; if reranking requires full vectors stored elsewhere, the prefetcher can stage codes first and defer full vectors until candidates are sufficiently promising.

Complexity analysis highlights the need for lightweight inference. Frontier size can be $O(W)$ for beam width W, and each frontier element can generate multiple candidate pages [28]. Scoring every candidate page with a high-dimensional model would be expensive. A practical design uses a low-rank feature map. If $\phi \in \mathbb{R}^m$ is large, we approximate $\theta^\top \phi$ by a low-rank factorization $\theta \approx Uv$ where $U \in \mathbb{R}^{m \times r}$ and $v \in \mathbb{R}^r$, or by random feature hashing. Feature hashing maps sparse categorical features into a fixed dimension with $O(1)$ updates, making scoring $O(\mathrm{nnz}(\phi))$. The prefetch selection itself is typically $O(M \log M)$ for M scored candidates if a heap is used, but M can be limited by only considering candidates derived from the top portion of the frontier under a rank cutoff. This induces an approximation, but empirically the marginal utility decays quickly with rank [29].

We now connect prefetching to approximate nearest neighbor graph dynamics. Consider a best-first traversal maintaining a visited set $\mathcal{V}$ and a priority queue of candidates keyed by distance estimate. Expanding a node requires reading its neighbor list page. If neighbor pages are remote, the traversal becomes a sequence of remote reads with limited parallelism. Prefetching can increase parallelism by issuing neighbor page reads for multiple likely expansions [30]. However, issuing too many reads increases congestion. The system therefore benefits from a prefetch window that is elastic: when the remote tier is lightly loaded, it prefetches deeper; when congestion rises, it becomes conservative. This elasticity is implemented by linking the prefetch budgets b_t and o_t to throttling prices, described in the next section [31].

4. Remote Memory Throttling and Multi-Objective Control

Remote memory throttling shapes the rate and concurrency of remote reads so that tail latency remains bounded under contention and so that the system enforces fairness across tenants and query classes. In a disaggregated vector database, throttling must consider two heterogeneous classes of remote reads. Metadata reads, such as neighbor pages or inverted list headers, often lie on the critical path. Payload reads, such as full vectors for reranking, can be deferred or sampled [32]. A throttling policy that treats all reads identically risks harming recall or wasting bandwidth.

We model the compute node as generating a stream of remote requests indexed by j with sizes b_j, class labels $c_j \in \{\text{meta}, \text{payload}\}$, and estimated marginal utilities u_j, interpreted as expected recall gain or expected reduction in uncertainty per byte. The storage tier provides a service capacity that varies with congestion. The throttling problem is to decide admission times and priorities. A convenient abstraction is token-based admission [33]. The compute node maintains tokens representing remote byte credits and remote outstanding credits. A request is admitted only if sufficient tokens exist. Tokens are replenished at rates set by a controller that observes congestion and tail latency signals.

Let $x(t)$ be the admitted byte rate at time t and let $y(t)$ be an observed tail signal such as the p-th percentile of remote completion time over a window [34]. We seek to maintain $y(t)$ below a target $\bar{y}$ while maximizing utility. A classical control-inspired formulation is

$$\max_{x(t)} \quad \int u(t)\, x(t)\, dt \;-\; \lambda \int x(t)\, dt \tag{4.1}$$

$$\text{s.t.} \quad y(t) \le \bar{y}, \qquad 0 \le x(t) \le x_{\max}, \tag{4.2}$$

where $u(t)$ is the average marginal utility per byte among currently pending requests and λ is a congestion price. The term with λ represents a penalty for consuming remote bandwidth. In a multi-tenant system, λ acts as a dual variable that adapts based on congestion; higher congestion increases λ and reduces admitted traffic [**?**].

We implement this idea with a Lagrangian that also captures energy and fairness. Suppose tenant a has weight w_a and its admitted rate is $x_a(t)$. Let $E_a(t)$ be an energy proxy proportional to bytes transferred and CPU used for decompression [35]. The multi-objective Lagrangian is

$$\mathcal{L} \;=\; \sum_a \int w_a\, u_a(t)\, x_a(t)\, dt \;-\; \lambda \int \Big(\sum_a x_a(t)\Big)\, dt \;-\; \mu \sum_a \int E_a(t)\, dt \;-\; \sum_a \nu_a \int \big(x_a(t) - \bar{x}_a\big)\, dt. \tag{4.3}$$

The variables λ, μ, ν_a are dual prices for total bandwidth, energy, and per-tenant fairness constraints. The admission policy that greedily admits requests with highest net utility per byte,

$$\text{admit } j \text{ if } \quad w_a u_j \;-\; \lambda \;-\; \mu e_j \;-\; \nu_a \;>\; 0, \tag{4.4}$$

resembles weighted scheduling with dynamic prices [36]. This rule can be realized by maintaining separate queues for classes and tenants and by selecting from the queue with highest positive score.

The dual variables are updated by observing congestion and budget violations. A simple gradient ascent on the duals yields

$$\lambda_{t+1} \;=\; \Big[\lambda_t \;+\; \gamma_\lambda\big(\widehat{y}_t - \bar{y}\big)\Big]_+, \qquad \nu_{a,t+1} \;=\; \Big[\nu_{a,t} \;+\; \gamma_\nu\big(\widehat{x}_{a,t} - \bar{x}_a\big)\Big]_+, \tag{4.5}$$

where $\widehat{y}_t$ is the measured tail signal and $\widehat{x}_{a,t}$ is observed tenant rate over a window, γ are step sizes, and $[\cdot]_+$ denotes projection onto nonnegative reals. The update increases the congestion price when tail exceeds target and increases the fairness price when a tenant exceeds its share [37]. This is backprop-friendly and can be integrated with learned utility predictors.

A crucial aspect is distinguishing critical metadata reads from optional payload reads. We represent a request's urgency by a slack time Δ_j, the estimated time until the request becomes blocking on the compute path. A metadata read typically has low slack; payload reads for reranking have higher slack [38]. Throttling can prioritize low-slack reads even under congestion by reserving a portion of tokens for them. This is equivalent to a constraint

$$\sum_{j \in \text{meta}} x_j(t) \;\geq\; \kappa \sum_j x_j(t), \tag{4.6}$$

for some $\kappa \in (0, 1)$, implemented by separate token pools.

We now connect throttling to prefetching [39]. Prefetches increase demand earlier; they are valuable only if they do not violate tail constraints. We therefore assign prefetch requests an effective utility that already accounts for timeliness and overfetch waste, and we allow the throttler's price λ to reduce prefetch admission more aggressively than critical-path reads. Concretely, prefetch requests can be multiplied by a factor $\zeta(\lambda)$ decreasing in λ, yielding a policy that smoothly disables prefetching as congestion rises [40]. This avoids oscillatory behavior where prefetching and throttling fight.

Stability matters. If the controller reacts too strongly, it can underutilize the remote tier; if it reacts too weakly, queues build and tail explodes. A simplified linearized analysis assumes $y(t)$ increases with admitted rate beyond capacity [41]. Let $y \approx y_0 + a(x - x^\star)$ for x near the operating point $x^\star$. The dual update on λ acts like an integral controller. The closed-loop system is stable for step size γ_λ sufficiently small relative to a, a standard condition that can be tuned empirically by measuring the slope of tail with respect to rate.

Finally, throttling interacts with energy and compression [42]. When vectors are stored compressed, remote bytes per candidate decrease but compute for decompression increases. Energy per request can be approximated as $e_j = e_{\text{net}} b_j + e_{\text{cpu}} \cdot \text{cycles}_j$. The controller can incorporate energy constraints by increasing μ when an energy budget is exceeded, effectively preferring requests that yield more utility per joule. This becomes relevant in dense reranking phases where payload reads dominate.

5. Query Planning, Approximate Execution, and Complexity

Vector databases often support multiple index structures and execution strategies [43]. In a disaggregated setting, the planner must choose a strategy that is robust to remote latency and congestion. Planning includes selecting which partitions to probe in an inverted file, choosing beam widths and termination conditions for graph traversal, deciding whether to prefetch payloads, and choosing reranking depth. Each choice trades recall for remote I/O and latency.

We formalize planning as selecting an execution policy π that maps query features and system state to algorithm parameters [44]. Let θ denote parameters such as number of probed centroids n_{probe}, beam width W, maximum expansions $E_{\max}$, and rerank depth R. The planner chooses θ to optimize expected objective under budgets:

$$\min_{\theta} \quad \mathbb{E}\big[T(\theta)\big] \; + \; \alpha\, \mathbb{E}\big[D(\theta)\big] \tag{5.1}$$

$$\text{s.t.} \quad \mathbb{E}\big[\text{Rec@}k(\theta)\big] \geq r_0. \tag{5.2}$$

This resembles constrained optimization with nonconvex and instance-dependent response surfaces. A practical approach uses a mixture of analytic models and empirical calibration [45]. For inverted files, expected candidates scanned is roughly proportional to n_{probe} times average list length, while recall depends on how much probability mass of true neighbors lies in the probed centroids. For graph search, expansions depend on W and graph properties; recall depends on avoiding local minima.

Dynamic programming appears when the planner must allocate a limited remote budget across stages. Consider a two-stage pipeline: candidate generation stage produces M candidates with compressed codes, then reranking fetches full vectors for a subset of size R. The remote budget is split: $B = B_{\text{gen}} + B_{\text{rerank}}$. The expected recall can be written as a function $U(M, R)$ with diminishing returns in both arguments [46]. If M depends on B_{gen} and R depends on B_{rerank}, then choosing the split can be cast as

$$\max_{B_{\text{gen}}} \quad U\big(M(B_{\text{gen}}),\; R(B - B_{\text{gen}})\big). \tag{5.3}$$

When B is discretized, a dynamic program can compute the best split by evaluating a small grid of budgets. This is useful when budgets are small enough to allow coarse discretization and when the utility surface is smooth. The same idea extends to multi-stage pipelines, such as centroid probing, code evaluation, payload prefetch, and final reranking. In disaggregated systems, the planner can incorporate congestion prices by treating the effective budget as B measured in priced bytes, where each remote byte costs λ [47].

Approximate execution frequently uses compression and sketching. Product quantization represents a vector as a concatenation of codewords from learned codebooks. If x is approximated by $\hat{x}$, then similarity estimation error affects ranking. For inner product, the error is

$$|q^\top x - q^\top \hat{x}| \quad = \quad |q^\top (x - \hat{x})| \;\leq\; \|q\|_2 \, \|x - \hat{x}\|_2. \tag{5.4}$$

Thus, bounding reconstruction error bounds score error [48]. In practice, the database can maintain per-codebook distortion statistics and use them to estimate uncertainty in scores. This uncertainty can inform reranking depth: if the top candidates are separated by margins larger than estimated error, reranking can be reduced, saving remote payload reads.

Random projection sketches can further reduce remote I/O. Let $R \in \mathbb{R}^{m \times d}$ be a random projection matrix with $m \ll d$ and define sketches $\tilde{x} = Rx$, $\tilde{q} = Rq$. If R approximately preserves distances, then $\|\tilde{q} - \tilde{x}\|_2$ approximates $\|q - x\|_2$. The database can store $\tilde{x}$ in smaller pages and use it to prune candidates before fetching full payload. The error of this pruning can be analyzed via concentration [49]. If the projection approximately preserves squared distances within $(1 \pm \varepsilon)$, then a candidate that is truly far

is unlikely to appear near under the sketch. This yields a tunable trade-off between additional compute and reduced remote payload fetches.

Hashing-based pruning, such as locality-sensitive hashing, provides another lever [50]. A hash function family partitions vectors so that similar vectors collide with higher probability. In a disaggregated system, the hash tables can be stored remotely with compact buckets. Querying hashes yields a small set of candidate buckets whose pages can be prefetched. The planner can choose the number of hash tables and probes based on congestion [51]. Complexity is typically $O(L \cdot P)$ where L is number of tables and P is probes per table, but remote traffic scales with number of buckets retrieved. The planner can use throttling prices to reduce L or P under congestion.

We now address computational hardness. Even in simplified forms, deciding which pages to prefetch or which partitions to probe under a strict byte budget is combinatorial [52]. Consider the following decision problem. Given a set of pages with sizes s_i and utilities v_i representing expected recall gain, determine whether there exists a subset with total size at most B whose total utility exceeds V. This is the knapsack decision problem, which is NP-complete. Prefetch selection reduces to this when $p_t(p)$ and $\Delta U_t(p)$ are deterministic and additive [53]. More realistic formulations incorporate timeliness, overlap, and interference, making the problem at least as hard. This implies that optimal prefetch schedules are infeasible at runtime, and systems must rely on greedy heuristics or approximations.

Graph traversal introduces another hard aspect: deciding an expansion order to minimize remote reads for a target recall resembles searching with uncertain rewards on a graph, which can be related to budgeted maximum coverage. If each neighbor page reveals a set of candidate vectors and the goal is to cover the true nearest neighbors with limited reads, then selecting pages to read is analogous to selecting sets to cover items, again NP-hard [54]. These reductions motivate heuristics that exploit structure, such as diminishing returns and locality. Greedy selection has approximation guarantees for submodular utilities; while recall is not strictly submodular, it often behaves similarly due to redundancy among candidates.

Given hardness, we advocate practical heuristics grounded in marginal utility estimation and congestion-aware pricing [55]. A common heuristic is to expand the most promising candidates first, to prefetch only one or two steps ahead, and to stop when marginal utility per priced byte falls below a threshold. This threshold is connected to the dual price λ: when congestion rises, the system requires higher utility to justify remote I/O. This is analogous to an economic admission control and provides an interpretable tuning knob.

We also consider error analysis for approximate queries under early termination [56]. If the query stops after evaluating a candidate set C, then the returned top-k is restricted to C. The recall depends on whether the true top-k is contained in C. Let $\mathcal{N}_k(q)$ denote true top-k and define $Z = |\mathcal{N}_k(q) \cap C|/k$. Then $\mathrm{Rec@}k = Z$. The system cannot directly compute Z online, but it can estimate a lower confidence bound using score margins and uncertainty. Suppose the system has approximate scores $\hat{s}$ with bounded error ϵ for candidates in C. If the k-th best candidate in C has score $\hat{s}_{(k)}$ and the best unseen candidate has an upper bound $\hat{s}_{\text{unseen}} + \epsilon$, then if

$$\hat{s}_{(k)} - \epsilon \; > \; \hat{s}_{\text{unseen}} + \epsilon, \tag{5.5}$$

the system can conclude that unseen candidates cannot enter top-k under the bound, improving confidence in recall. Estimating $\hat{s}_{\text{unseen}}$ can use coarse quantizer bounds or sketch-based bounds. This supports adaptive termination that is sensitive to query difficulty.

6. Distributed Implementation, Storage Internals, and Evaluation Methodology

A disaggregated vector database must implement low-latency remote access, efficient caching, and concurrency-safe execution while maintaining predictable tail behavior [57]. We describe an implementation approach focused on the interactions among page layout, transport, prefetching, throttling, and query execution.

Storage internals are organized around page granularity. Neighbor lists, centroid metadata, PQ code blocks, and full vectors are stored in separate page types to allow differentiated caching and fetch policies. Page sizes are chosen to balance transfer efficiency and overfetch [58]. Larger pages amortize transport overhead and improve compression, but they increase waste when only a few items are needed. Compression is guided by entropy considerations. If a page contains PQ codes with b bits per subquantizer and m subquantizers, then each vector code is mb bits, and the page’s information content is roughly proportional to its entropy under the code distribution. Storing codes in a packed format reduces bytes, while storing frequently accessed neighbor pages uncompressed may reduce CPU overhead for decoding [59]. The optimal choice depends on whether the remote tier is bandwidth- or latency-limited. In disaggregated execution, bandwidth pressure is often the limiting factor during high concurrency, suggesting that compression is valuable, but decompression must be engineered carefully to avoid shifting the bottleneck to CPU.

Cache management at compute nodes uses a segmented policy that separates metadata pages from payload pages [60]. Metadata pages have high reuse due to shared traversal paths in popular regions of the graph or popular centroids. Payload pages may have lower reuse depending on query diversity. The cache records both recency and frequency, and it exposes features to the prefetcher such as reuse distance estimates and admission decisions. A lightweight feature store maintains per-page statistics updated with approximate counting to avoid contention, using techniques akin to count-min sketches for frequency [61].

Remote access is implemented over a low-overhead transport such as an RDMA-like one-sided read, or an RPC over a kernel-bypass stack. The system batches small reads when possible. Prefetching issues reads asynchronously and stores completions in a completion queue. The query executor consumes pages from local cache or completion buffers [62]. To reduce synchronization, the executor treats missing pages as futures and continues with other expansions when available, which increases overlap. This requires that the traversal algorithm be expressed in a way that tolerates out-of-order expansions. Best-first search naturally uses a priority queue; the executor can pop the next available candidate whose dependencies are satisfied, while keeping a bounded set of in-flight expansions.

The throttling controller runs per compute node with periodic feedback from storage nodes [63]. Storage nodes export congestion signals such as queue depths, service time percentiles, and byte rates. Compute nodes aggregate these into $\widehat{y}_t$ and update prices. To avoid global synchronization, each compute node maintains local estimates of dual prices, and storage nodes can broadcast coarse-grained price hints. This decentralization is important for scalability. The correctness requirement is not exact fairness at every moment but bounded unfairness over windows [64]. Token replenishment rates are derived from the prices, and the request admission path remains lock-free by using atomic counters.

Performance engineering emphasizes tail latency. The executor isolates critical-path metadata reads by giving them higher priority and reserving tokens [65]. It also limits head-of-line blocking by splitting large payload reads into smaller chunks only when it is beneficial, noting that too fine a granularity increases overhead and can harm throughput. The prefetcher reduces compute overhead via feature hashing and by scoring only a limited set of candidates. It also memoizes common feature computations such as projected query embeddings. Decompression uses vectorized instructions when available and is scheduled on dedicated threads to avoid interfering with latency-sensitive traversal logic [66].

Distributed execution across multiple storage nodes introduces placement and routing. Pages are partitioned by consistent hashing on page identifiers, so that each page has a primary storage node and optional replicas. A query may therefore contact multiple storage nodes. The executor coalesces requests by storage node to reduce connection overhead and to improve batching [67]. Congestion is also per storage node, so the throttler maintains per-destination prices λ_s and admits requests based on the destination’s current price. This prevents a hot shard from destabilizing the entire cluster.

Evaluation methodology focuses on metrics that reveal the benefits and risks of prefetching and throttling. Latency is reported not only as average but also as tail percentiles, since remote congestion primarily affects tails [**?**]. Recall is measured at k and at multiple k values. Remote bytes, remote request counts, and outstanding concurrency are measured per query and per tenant. The evaluation includes

interference experiments where background workloads generate remote traffic, and it measures how the controller maintains tail targets. It also includes ablations that disable prefetching, disable throttling, or replace adaptive scoring with static heuristics [68]. To assess generality, experiments vary embedding dimension d, index type, compression settings, and page sizes.

Reproducibility requires controlling nondeterminism. Graph search can be sensitive to tie-breaking, concurrency, and floating point differences [69]. The system fixes random seeds for index construction, uses deterministic tie-breaking rules in priority queues, and records the exact parameter settings used by the planner and controller. Traces of queries and system signals are logged with timestamps to enable replay. The training of the prefetch probability model records the feature hashing seed, learning rates, and the window definitions for labels. Evaluation scripts pin CPU frequency settings where possible, isolate network paths, and report the hardware and kernel-bypass configuration [70]. These practices do not remove all variance, especially under shared fabrics, but they narrow it and enable meaningful comparisons.

7. Conclusion

Storage-compute disaggregation can make vector databases more elastic and operationally simpler, but it elevates remote memory traffic to a first-class performance constraint. Approximate nearest neighbor search, whether graph-based or centroid-based, produces access patterns that are difficult for remote tiers because they are random, bursty, and query-dependent. This paper presented a system model and mechanisms for making disaggregated vector search robust under these conditions [71]. Adaptive prefetching was framed as a state-conditioned prediction and selection problem over a dynamic frontier, using lightweight feature embeddings, low-rank projections, and congestion-aware value discounting. Remote memory throttling was formulated as a multi-objective control problem with Lagrangian prices that internalize congestion, energy, and fairness, enabling decentralized token-based admission that prioritizes critical-path reads while scaling back speculative prefetches during contention.

We analyzed the complexity of optimal prefetch and probing decisions, highlighting NP-hardness in simplified reductions and motivating greedy, price-guided heuristics. We discussed approximate execution error sources from compression and sketches and connected uncertainty estimates to adaptive termination and reranking [72]. Implementation considerations emphasized page layout, transport batching, cache segmentation, and tail-focused performance engineering. Evaluation methodology centered on tail latency, interference, and reproducibility practices suitable for shared clusters.

The combined perspective suggests that disaggregated vector databases benefit from treating remote memory as a priced resource whose consumption is optimized jointly with recall, rather than as a passive extension of DRAM. Prefetching and throttling are most effective when coupled through shared estimates of marginal utility, timeliness, and congestion prices, allowing the system to adapt continuously across workload shifts and contention regimes [73].

References

[1] K. Erciyes, *Distributed Graph Algorithms*, pp. 117–136. Springer International Publishing, 4 2018.

[2] C. B. Jones, G. Fragkos, and R. Darbali-Zamora, “Exploring the interplay between distributed wind generators and solar photovoltaic systems,” *Journal of Renewable and Sustainable Energy*, vol. 17, 1 2025.

[3] N. Suri, “Distributed systems security,” 10 2019.

[4] A. Polyakov, “On global existence of strong and classical solutions of navier-stokes equations,” 4 2019.

[5] R. Malik, S. Kim, X. Jin, C. Ramachandran, J. Han, I. Gupta, and K. Nahrstedt, “Mlr-index: An index structure for fast and scalable similarity search in high dimensions,” in *International Conference on Scientific and Statistical Database Management*, pp. 167–184, Springer, 2009.

[6] E. E. Mohammed, R. Y. S. Naji, A. A. Hussein, M. A. Saeed, and R. A. M. A. Selwi, "Anomaly detection system for secure cloud computing environment using machine learning," in *2025 5th International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, pp. 1–9, IEEE, 8 2025.

[7] M. Zaccarini, F. Poltronieri, D. Borsatti, W. Cerroni, L. Foschini, G. Y. Grabarnik, D. Scotece, L. Shwartz, C. Stefanelli, and M. Tortonesi, "Chaos engineering based kubernetes pod rescheduling through deep sets and reinforcement learning," in *NOMS 2025-2025 IEEE Network Operations and Management Symposium*, pp. 1–7, IEEE, 5 2025.

[8] D. Basin, M. Gras, S. Krstic, and J. Schneider, *RV - Scalable Online Monitoring of Distributed Systems*, pp. 197–220. Germany: Springer International Publishing, 10 2020.

[9] S. Akbari and M. Hauswirth, "Universal workers: A vision for eliminating cold starts in serverless computing," in *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*, pp. 442–444, IEEE, 7 2025.

[10] L. Vishwakarma, D. Das, S. K. Das, and C. Becker, "Smartgrid-ng: Blockchain protocol for secure transaction processing in next generation smart grid," in *Proceedings of the 25th International Conference on Distributed Computing and Networking*, pp. 174–185, ACM, 1 2024.

[11] A. Pratinav and S. Mishra, "Outbox pattern for reliable distributed systems," *ISCSITR-INTERNATIONAL JOURNAL OF CLOUD COMPUTING*, vol. 6, pp. 18–25, 12 2025.

[12] L. Tirpitz, I. Kunze, P. Niemietz, A. K. Gerhardus, T. Bergs, K. Wehrle, and S. Geisler, "Reducio: Data aggregation and stability detection for industrial processes using in-network computing," in *Proceedings of the 19th ACM International Conference on Distributed and Event-based Systems*, pp. 98–109, ACM, 6 2025.

[13] S. Holzer, P. Frangoudis, C. Tsigkanos, and S. Dustdar, "Smt-as-a-service for fog-supported cyber-physical systems," in *Proceedings of the 25th International Conference on Distributed Computing and Networking*, pp. 154–163, ACM, 1 2024.

[14] D. D. Gaudio, B. Ariguib, A. Bartenbach, and G. Solakis, "A live context model for semantic reasoning in iot applications," in *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 322–327, IEEE, 3 2022.

[15] B. Full, J. Manner, S. Böhm, and G. Wirtz, *MicroStream vs. JPA: An Empirical Investigation*, pp. 99–118. Germany: Springer International Publishing, 10 2022.

[16] S. Priyadarshini and S. Rodda, "Frequent subgraph mining by giraph distributed system," *International Journal of Engineering and Advanced Technology*, vol. 9, pp. 1267–1275, 6 2020.

[17] A. Poshtkohi and M. B. Ghaznavi-Ghoushchi, *IoT and Distributed Systems*, pp. 15–22. Auerbach Publications, 2 2023.

[18] M. Jančič, J. Slak, and G. Kosec, "p-refined rbf-fd solution of a poisson problem," in *2021 6th International Conference on Smart and Sustainable Technologies (SpliTech)*, pp. 1–6, IEEE, 9 2021.

[19] L. F. H. Duarte, G. B. Nardes, W. Grignani, D. R. Melo, and C. A. Zeferino, "Deep nibble: A 4-bit number format for efficient dnn training and inference in fpga," in *2024 37th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)*, pp. 1–5, IEEE, 9 2024.

[20] C. Avasalcai, B. Zarrin, and S. Dustdar, "Edgeflow—developing and deploying latency-sensitive iot edge applications," *IEEE Internet of Things Journal*, vol. 9, pp. 3877–3888, 3 2022.

[21] J. D. Herath, P. Yang, and G. Yan, "Codaspy - real-time evasion attacks against deep learning-based anomaly detection from distributed system logs," in *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, pp. 29–40, ACM, 4 2021.

[22] R. Chandrasekar, R. Suresh, and S. Ponnambalam, "Evaluating an obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 International Conference on Advanced Computing and Communications*, pp. 628–629, IEEE, 2006.

[23] R. Fronteddu, U. Ardinghi, L. Colombi, S. Dahdal, A. Morelli, M. Tortonesi, C. Stefanelli, and N. Suri, "Semantic information management systems," in *2025 International Conference on Military Communication and Information Systems (ICMCIS)*, pp. 1–9, IEEE, 5 2025.

[24] A. Pfleiderer and B. Bauer, "Fused hybrid training samples through synthetic anomaly generation for optimized model training," in *2025 International Conference on Advanced Machine Learning and Data Science (AMLDS)*, pp. 248–256, IEEE, 7 2025.

[25] A. Ramesh, T. Huang, E. Ruppel, D. Dasari, B. Pourmohseni, F. Smirnov, M. Giani, P. Pazzaglia, C. Shelton, N. Pereira, A. Hamann, D. Ziegenbein, and A. Rowe, "Silverline: Lightweight virtualization and orchestration of distributed systems," in *2025 IEEE 31st Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 375–388, IEEE, 5 2025.

[26] P. Raith, T. Rausch, A. Furutanpey, and S. Dustdar, "faas-sim: A trace-driven simulation framework for serverless edge computing platforms," *Software: Practice and Experience*, vol. 53, pp. 2327–2361, 10 2023.

[27] C. Imianosky, P. R. O. Valim, C. A. Zeferino, and F. Viel, *SBESC - Evaluating the CCSDS 123 Compressor Running on RISC-V and ARM Architectures*. IEEE, 11 2020.

[28] S. H. S. A. UBAIDILLAH, N. AHMAD, and N. A. Sahabudin, "A survey on potential reactive fault tolerance approach for distributed systems in big data," in *Third International Conference on Computer Vision and Information Technology (CVIT 2022)*, pp. 21–21, SPIE, 2 2023.

[29] T. Kharkovskaia, "Design of interval observers for uncertain distributed systems," 12 2019.

[30] U. Matter, *Distributed Systems*, pp. 81–96. Chapman and Hall/CRC, 6 2023.

[31] J. Tournier, F. Lesueur, F. L. Mouël, L. Guyon, and H. Ben-Hassine, "Iotmap, a modelling system for heterogeneous iot networks," 6 2020.

[32] H. Geppert, F. Dürr, and K. Rothermel, "Efficient conflict graph creation for time-sensitive networks with dynamically changing communication demands," *IEEE Transactions on Network and Service Management*, pp. 1–1, 1 2025.

[33] *Proceedings of the 2022 Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*, ACM, 7 2022.

[34] M. Jančič, J. Slak, and G. Kosec, "Mipro - gpu accelerated rbf-fd solution of poisson's equation," in *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pp. 214–218, IEEE, 9 2020.

[35] Y. Jia and X. Shu, "Visual model checking distributed system," in *Proceedings of the 2022 5th International Conference on Artificial Intelligence and Pattern Recognition*, pp. 285–291, ACM, 9 2022.

[36] M. Zakarya, L. Gillam, K. Salah, O. F. Rana, S. Tirunagari, and R. BUYYA, "Colocateme: Aggregation-based, energy, performance and cost aware vm placement and consolidation in heterogeneous iaas clouds," 6 2021.

[37] R. Chandrasekar and T. Srinivasan, "An improved probabilistic ant based clustering for distributed databases," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, pp. 2701–2706, 2007.

[38] E. Azketa, X. Mendialdua, I. Ibarguren, and A. Solís, "Método de sincronización para sistemas distribuidos con seguridad funcional," *Revista Iberoamericana de Automática e Informática industrial*, vol. 18, pp. 113–118, 4 2021.

[39] K. Grining, "Privacy-preserving protocols in unreliable distributed systems," 10 2020.

[40] J. Pennekamp, R. Matzutt, S. S. Kanhere, J. Hiller, and K. Wehrle, "The road to accountable and dependable manufacturing," *Automation*, vol. 2, pp. 202–219, 9 2021.

[41] M. Josten and T. Weis, "Large language models as a cyber threat: Towards countering llm-based spam attacks," in *2025 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 606–607, IEEE, 3 2025.

[42] A. Arman, P. Bellini, D. Bologna, P. Nesi, G. Pantaleo, and M. Paolucci, "Automating iot data ingestion enabling visual representation.," *Sensors (Basel, Switzerland)*, vol. 21, pp. 8429–8429, 12 2021.

[43] R. KREIFELDT and M. BAILEY, "Integrated control of distributed systems," in *Reproduced Sound 22*, Institute of Acoustics, 12 2023.

[44] G. B. C, A. J. S, S. S, V. M, and R. M, "Water theft and leakage identification in distributed system," in *2022 Smart Technologies, Communication and Robotics (STCR)*, pp. 1–4, IEEE, 12 2022.

[45] W. Yin, B. B. Zhu, Y. Xie, P. Zhou, and D. Feng, "Backdoor attacks on bimodal salient object detection with rgb-thermal data," in *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 4110–4119, ACM, 10 2024.

[46] E. Bazgir, T. T. Tanha, A. A. Bhuiyan, E. Haque, and M. S. Uddin, "Analysis of distributed systems," *International Journal of Computer Applications*, vol. 186, pp. 16–21, 11 2024.

[47] "Big data and distributed systems," 10 2025.

[48] C. Chan and B. Cooper, "Debugging incidents in google's distributed systems: How experts debug production issues in complex distributed systems," *Queue*, vol. 18, pp. 47–66, 4 2020.

[49] A. Thakur, S. Verma, N. Sindhwani*, and R. Vashisth, "Applications of optimized distributed systems in healthcare," 3 2024.

[50] K. Umapathy, S. Prabakaran, T. Dineshkumar, M. A. Archana, D. K. Sri, and A. N. Aledaily, *Theories of blockchain and distributed systems*, pp. 52–68. CRC Press, 9 2023.

[51] I. Colonnelli and M. Aldinucci, "Workflow models for heterogeneous distributed systems," 5 2022.

[52] H. Bazille, E. Fabre, and B. Genest, "Certification formelle des réseaux neuronaux profonds : un état de l'art en 2019," 11 2019.

[53] M. K. Aguilera, *ApPLIED@PODC - Apply or Perish*. ACM, 7 2018.

[54] V. Vijaykumar, R. Chandrasekar, and T. Srinivasan, "An obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 IEEE Conference on Cybernetics and Intelligent Systems*, pp. 1–6, IEEE, 2006.

[55] K. K. Horvath, D. Kimovski, B. Spiess, O. Hohlfeld, and R. Prodan, "Seal-cc: Scalable latency evaluation methodology for internet-of-things services," in *Proceedings of the 14th International Conference on the Internet of Things*, pp. 117–126, ACM, 11 2024.

[56] N. Nischal, "Automated evaluation for distributed system assignments," 1 2023.

[57] S. Bagchi, "Modeling slicer and control algorithm for distributed computation in metrized weaker topological spaces," *International Journal of Control and Automation*, vol. 10, pp. 209–220, 12 2017.

[58] J. Yuan, A. Le-Tuan, M. Nguyen-Duc, T.-K. Tran, M. Hauswirth, and D. Le-Phuoc, *VisionKG: Unleashing the Power of Visual Datasets via Knowledge Graph*, pp. 75–93. Germany: Springer Nature Switzerland, 5 2024.

[59] L. Nenov, "Reinforcement learning for key management in distributed systems," in *2024 32nd National Conference with International Participation (TELECOM)*, pp. 1–5, IEEE, 11 2024.

[60] D. Efimov, A. Polyakov, and A. Aleksandrov, "Proofs for "discretization of homogeneous systems using euler method with a state-dependent step"," 6 2019.

[61] I. Ivanisenko, "Dynamic method of distributed system load balancing evaluate," , ' . , vol. 2, pp. 74–77, 5 2021.

[62] F. Mora, A. Desai, E. Polgreen, and S. A. Seshia, "Message chains for distributed system verification," *Proceedings of the ACM on Programming Languages*, vol. 7, pp. 2224–2250, 10 2023.

[63] T. Srinivasan, R. Chandrasekar, V. Vijaykumar, V. Mahadevan, A. Meyyappan, and A. Manikandan, "Localized tree change multicast protocol for mobile ad hoc networks," in *2006 International Conference on Wireless and Mobile Communications (ICWMC'06)*, pp. 44–44, IEEE, 2006.

[64] K. Cui, S. Liu, W. Feng, X. Deng, L. Gao, M. Cheng, H. Lu, and L. T. Yang, "Correlation-aware cross-modal attention network for fashion compatibility modeling in ugc systems," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 21, pp. 1–24, 11 2025.

[65] G. B. King, S. McCarthy, P. Pattabhiraman, J. Luciani, and M. Fleming, "Fallout: Distributed systems testing as a service," 10 2021.

[66] J. Flak, T. Skowron, R. Cupek, M. Fojcik, D. Caban, and A. Domański, "Zigbee network for agv communication in industrial environments," in *2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–9, IEEE, 10 2023.

[67] T. Pollinger, A. V. Craen, C. Niethammer, M. Breyer, and D. Pflüger, "Leveraging the compute power of two hpc systems for higher-dimensional grid-based simulations with the widely-distributed sparse grid combination technique," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, ACM, 11 2023.

[68] J. Han, X. Wei, R. Chen, and H. Chen, "Seraph: A performance-cost aware tuner for training reinforcement learning model on serverless computing," in *Proceedings of the 15th ACM SIGOPS Asia-Pacific Workshop on Systems*, pp. 95–101, ACM, 9 2024.

[69] A. Saleh, "Privacy-protection in cooperative distributed systems," 8 2018.

[70] R. Malik, C. Ramachandran, I. Gupta, and K. Nahrstedt, "Samera: a scalable and memory-efficient feature extraction algorithm for short 3d video segments.," in *IMMERSCOM*, p. 18, 2009.

[71] "Distributed systems," 12 2020.

[72] L. Mariani, M. Pezzè, O. Riganelli, and R. Xin, "Predicting failures in multi-tier distributed systems," 11 2019.

[73] J. Barreiro-Gomez, *Distributed System Partitioning and DMPC*, pp. 179–192. Springer International Publishing, 8 2018.