ARTICLE

A Scalable Bipartite Graph Matching System for Recommendation Candidates Using Blocked Parallel Augmentation in Marketplaces

Aiman Hakimi[1] and Firdaus Rahimi[2]

[1]Department of Computer Science, Universiti Kepulauan Nusantara, Jalan Kenanga 12/7, Shah Alam, Selangor, Malaysia
[2]Department of Computer Science, Institut Pengurusan Sabah Maju, Jalan Tun Fuad Stephens, Kota Kinabalu, Sabah, Malaysia

Abstract

Marketplace recommenders operate under two competing pressures: they must surface relevant candidates at low latency while respecting supply-side capacity, policy constraints, and ecosystem balance. A common failure mode of purely local candidate generation is overproduction of redundant or infeasible options that are later filtered by business rules or ranking-time caps, wasting computation and reducing coverage for the long tail of listings and sellers. This paper presents a scalable bipartite graph matching system that produces recommendation candidates as a constrained assignment problem between demand entities and supply entities. The core contribution is blocked parallel augmentation, a distributed augmentation mechanism that decomposes the graph into blocks, performs localized augmenting path discovery and tentative updates in parallel, and commits non-conflicting augmentations via deterministic arbitration. The design targets the operational characteristics of marketplaces: dynamic eligibility edges, high churn in supply, asymmetric degree distributions, and the need for controllable trade-offs between solution quality and throughput. The system integrates a capacity-aware matching objective with practical constraints including per-supply exposure budgets, per-demand candidate quotas, and soft penalties that discourage repeated exposure. The paper details the algorithmic structure, consistency model, partitioning strategy, and an implementation that supports both batch and incremental updates. Analytical considerations and trace-driven experiments demonstrate how block sizing, conflict rates, and degree skew influence convergence and resource usage, and how the approach improves feasible candidate yield and supply coverage under fixed compute budgets.

1. Introduction

Recommendation candidate generation in marketplaces differs from candidate generation in single-sided catalog settings because supply is both constrained and strategic [1]. A listing can be recommended only while it is available, eligible for a given user context, and within policy and budget constraints that may be applied at the seller, category, or inventory level. Demand, in turn, is heterogeneous in intent, location, and timing, and the system is often required to produce a bounded number of candidates per request to satisfy online latency objectives. These constraints motivate a global view of candidate selection: rather than generating a large local neighborhood for each user independently and then filtering, a matching-based layer can produce candidates that are already feasible under capacity and policy constraints, with controllable allocation across the supply side.

A direct maximum matching formulation, however, is computationally challenging at marketplace scale [2]. The candidate graph can contain billions of potential edges when eligibility and similarity are considered, even after strong pre-filtering. Graph structure is also highly non-uniform. Popular queries, dense metropolitan regions, or viral items generate extreme degree concentrations, while the majority of listings and users remain in sparse regimes. Moreover, the graph is dynamic: listings are created and removed, inventory states change, and eligibility constraints vary with location and policy updates. These properties stress classical centralized matching algorithms and complicate

parallelization because augmenting paths, the mechanism by which matchings improve, inherently interact through shared vertices and edges [3].

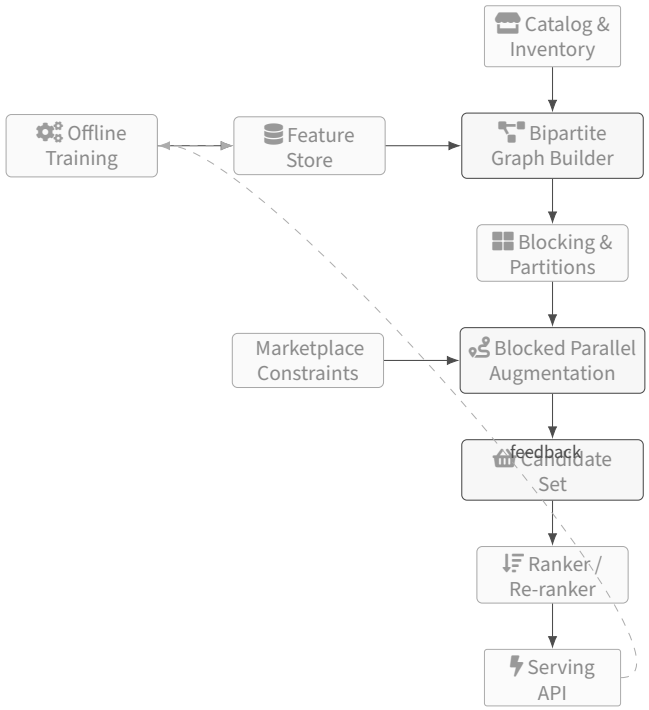


Figure 1. End-to-end candidate generation: marketplace signals and catalog data build a bipartite graph, partitioned into blocks for parallel augmentation-based matching under marketplace constraints, producing candidates for ranking and serving with a lightweight feedback loop.

This paper describes a system that treats candidate generation as a bipartite matching or capacity-constrained b-matching problem and solves it with a blocked parallel augmentation algorithm tailored to distributed environments. The central idea is to decompose the bipartite graph into blocks using a partitioning function, run localized augmentation searches inside blocks in parallel to propose improvements, and then commit a subset of those improvements that are provably non-conflicting under a lightweight arbitration mechanism. Blocking reduces coordination overhead by increasing locality of vertex ownership and limiting the scope of conflicts, while augmentation preserves the structural advantage of matching methods: improvements are expressed as alternating paths or cycles that can be applied with simple edge flips.

The system is designed as a candidate producer rather than a final allocator [4]. Its output is a bounded set of feasible edges per demand entity, intended to seed a downstream ranking model or auction-like selection. This distinction matters operationally. Candidate generation tolerates approximate optimality if feasibility, diversity, and latency targets are met. The system therefore exposes knobs that trade solution quality for throughput, including maximum augmentation depth, block size, and conflict resolution strictness. These knobs allow the system to adapt to different product surfaces, from high-frequency home feeds to lower-frequency transactional flows [5].

The contributions of this paper are threefold. First, it presents a problem formulation that captures marketplace candidate generation as a bipartite capacity-constrained matching with soft exposure regularization, expressing a balance between relevance and ecosystem constraints. Second, it in-

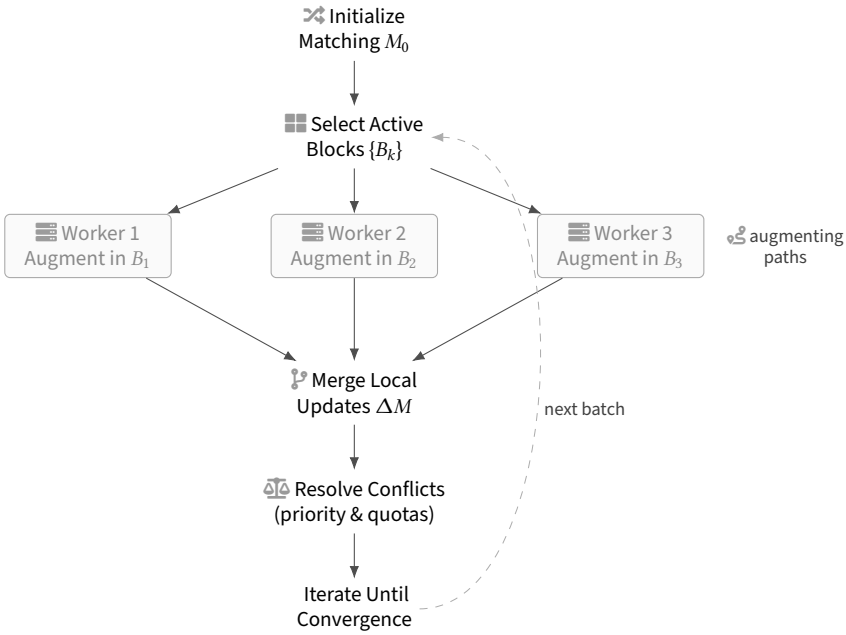


Figure 2. Blocked parallel augmentation loop: workers compute local augmentations within selected blocks, then merge and reconcile updates into a global matching under marketplace policies before iterating.

troduces blocked parallel augmentation, a distributed augmentation method that uses block-local searches and deterministic commit to scale matching improvements while preserving correctness with respect to capacity constraints. Third, it describes an end-to-end architecture that integrates graph construction, partitioning, augmentation, state storage, and incremental updates, along with analytical and experimental characterization of convergence, conflict behavior, and performance under skew and churn [6].

The remainder of the paper proceeds as follows. The next section formalizes the graph model, objective, and constraints, emphasizing operational constraints common in marketplaces. The following section describes blocked parallel augmentation in detail, including its search strategy, conflict detection, and commit protocol. Subsequent sections cover the distributed architecture, the consistency and data engineering considerations needed for production, and an evaluation methodology with results from controlled experiments on production-like traces and synthetic stress tests. The paper concludes with a discussion of the practical trade-offs and limitations of matching-based candidate generation at scale [7].

2. System Model and Problem Formulation

We consider a bipartite graph $G = (U, V, E)$ in which U represents demand-side entities such as user-context pairs, sessions, or query impressions, and V represents supply-side entities such as listings, sellers, or inventory items. An edge $e = (u, v) \in E$ exists when v is eligible to be shown to u under hard constraints, and the system has evidence that showing v to u is potentially useful. Eligibility encapsulates deterministic constraints such as geo-availability, age or policy restrictions, inventory state, category rules, and seller status. Usefulness is represented by a learned score or heuristic similarity that is used to prioritize which feasible edges should be included in the candidate set [8].

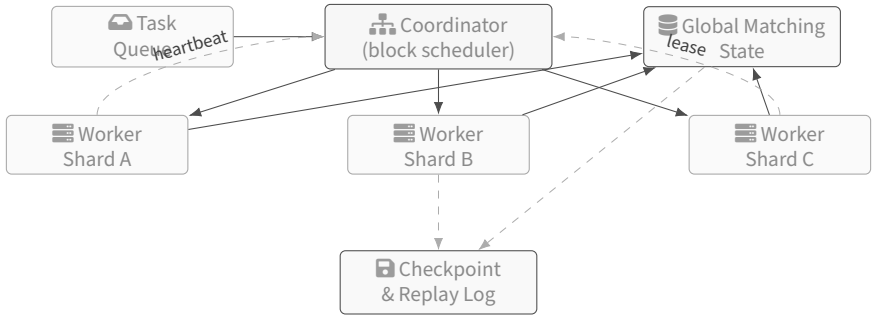


Figure 3. Distributed execution view: a coordinator schedules block tasks to shard-aligned workers via a queue, workers apply local augmentations against a shared matching state, and checkpoints enable replay and recovery for long-running workloads.

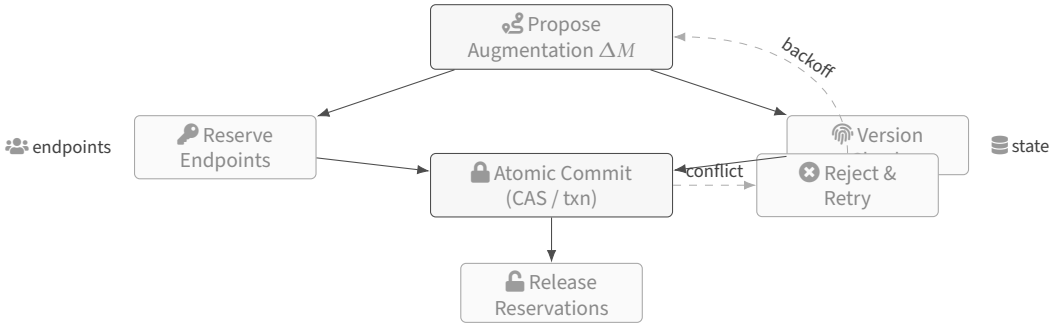


Figure 4. Conflict-safe update protocol: proposed augmentations reserve endpoints and validate state versions before an atomic commit; conflicts trigger rejection with backoff to preserve correctness under parallel block execution.

Candidate generation typically seeks to output a bounded set $C(u) \subseteq \{v : (u, v) \in E\}$ of size at most k_u, where k_u is a per-demand quota determined by product and latency budgets. In a marketplace, there is also a need to regulate supply-side exposure. A listing may have an exposure budget per time window, a seller may have a cap across listings, or a category may have allocation constraints. These constraints can be expressed as capacities on the supply side. In the simplest case, each supply vertex v has a capacity $c_v \geq 0$ indicating the maximum number of demand vertices it can be matched to in the current generation window. Similarly, demand vertices may have a capacity b_u indicating how many candidates they should receive; in standard candidate generation b_u is often close to k_u, but the system may allow smaller values under scarcity or aggressive filtering [9].

This leads to a bipartite b-matching formulation in which we seek a multiset of edges $M \subseteq E$ such that degree constraints hold:

$$\sum_{v:(u,v)\in M} 1 \leq b_u \quad \forall u \in U, \tag{1}$$

$$\sum_{u:(u,v)\in M} 1 \leq c_v \quad \forall v \in V. \tag{2}$$

When $b_u = 1$ and $c_v = 1$, the problem reduces to standard bipartite matching. For candidate generation, b_u is typically larger than one, and c_v may also exceed one when multiple exposures per supply are allowed.

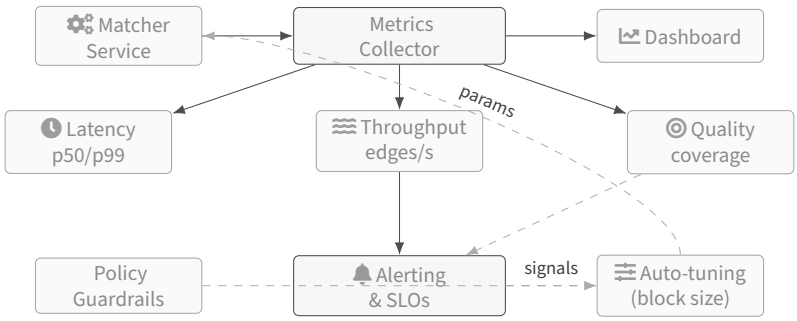


Figure 5. Operational view: the matcher exports latency/throughput/quality signals to dashboards and SLO alerts; guardrails and auto-tuning adjust block sizing and scheduling parameters to keep candidate generation stable under marketplace load.

To incorporate relevance, we associate a weight w_{uv} with each edge (u, v). The weight can be derived from a retrieval model score, a learned embedding similarity, a calibrated click probability, or a composite of multiple signals [10]. The natural objective is to maximize total weight subject to capacities:

$$\max_{M \subseteq E} \quad \sum_{(u,v)\in M} w_{uv} \tag{3}$$

$$\text{s.t.} \quad \sum_{v:(u,v)\in M} 1 \le b_u \quad \forall u, \tag{4}$$

$$\sum_{u:(u,v)\in M} 1 \le c_v \quad \forall v. \tag{5}$$

This formulation captures the allocation tension between satisfying high-value demand edges and respecting supply exposure limits. However, marketplace recommenders often also require soft regularization that discourages repeated exposure of the same supply to the same demand entity over short horizons, and that encourages distribution across sellers or categories. Such requirements can be expressed by modifying weights with penalties that depend on historical exposure counts h_{uv} or supply-side aggregates H_v. A practical model uses an effective weight:

$$\tilde{w}_{uv} = w_{uv} - \lambda\, \phi(h_{uv}) - \mu\, \psi(H_v), \tag{6}$$

where ϕ and ψ are nonnegative increasing functions, and λ and μ control the strength of the penalties [11]. This retains a linear objective in the decision variables if h_{uv} and H_v are treated as constants during a generation window, which is typical when penalties are derived from a trailing time window and updated between runs.

The graph E itself is not given explicitly at full scale. It is generated by a pipeline that combines hard filters and approximate retrieval. Hard filters ensure eligibility, while approximate retrieval provides a bounded neighborhood per demand vertex using embedding search, query expansion, or rule-based retrieval. As a result, the observed graph is a sampled subgraph of an implicit eligibility relation [12]. The sampling process interacts with matching: if retrieval over-concentrates on a small set of popular supply vertices, then capacities will cause many edges to be rejected, lowering the realized candidate yield. Conversely, if retrieval is diversified but too diffuse, the matching may sacrifice weight. This motivates an integrated view in which retrieval produces a moderately large but structured neighborhood and matching enforces feasibility and allocation.

Operationally, the system must support two execution modes. In batch mode, the system periodically computes a fresh matching on a snapshot of the graph for a time window, producing candidate sets that are consumed by online services [13]. In incremental mode, the system maintains and updates a matching as edges and vertices change due to events such as listing creation, inventory changes, or policy updates. Incremental updates are critical for freshness, but they introduce challenges because maintaining exact optimality under streaming updates is costly. The blocked parallel augmentation approach is designed to handle both modes by treating each run as a bounded number of augmentation rounds that can start from an existing matching state.

Because the output is used for candidate generation rather than final allocation, the system accepts approximate solutions provided they satisfy hard constraints and provide high feasible yield [14]. This leads to a quality measure that is not solely total weight. One relevant measure is feasible coverage, the fraction of demand vertices achieving their desired quota:

$$\text{Yield} = \frac{1}{|U|} \sum_{u \in U} \frac{\deg_M(u)}{b_u}. \tag{7}$$

Another measure is supply coverage and distribution, for example the fraction of supply vertices that receive at least one match, or the entropy of the match distribution across sellers. These measures motivate algorithmic choices that avoid pathological concentration even when weights are steeply skewed. In practice, the system may cap per-supply degrees, apply weight smoothing, or incorporate penalties as above [15].

Finally, feasibility constraints may be context-dependent. A supply item may be eligible only for a subset of demand contexts; capacities may vary by time and geography; and some constraints may be hierarchical, such as seller-level caps that aggregate across listings. Hierarchical constraints can be represented by introducing additional supply-side vertices corresponding to groups and connecting listings to groups with capacity edges. Alternatively, they can be handled by projecting constraints into effective capacities c_v computed per listing based on group budgets [16]. The system described here assumes capacities at the vertex level during augmentation, while allowing capacity computation to incorporate hierarchical rules upstream.

3. Blocked Parallel Augmentation Algorithm

Augmentation-based matching algorithms improve an existing matching by finding alternating paths or cycles that increase the matching size or weight. In the unweighted case, an augmenting path is a path that alternates between unmatched and matched edges and starts and ends at free capacity on opposite sides; flipping matched status along the path increases the cardinality of the matching by one. In the b-matching setting, the notion generalizes by treating each unit of residual capacity at a vertex as a free slot. An augmenting path then starts at a demand vertex u with residual capacity, ends at a supply vertex v with residual capacity, and alternates between edges not in the current matching and edges in the matching, with the flip operation increasing total matched edges by one unit while respecting capacities [17].

The challenge in distributed environments is that augmenting paths interact. Two paths may share a vertex or edge, and applying both can violate capacity constraints or create inconsistencies. Classical parallelization strategies seek collections of vertex-disjoint augmenting paths and apply them simultaneously, but finding a large set of disjoint paths in a distributed graph is itself expensive. Blocked parallel augmentation addresses this by limiting the scope of path search and conflict detection to blocks that are designed to have high locality, and by using a commit protocol that guarantees that only non-conflicting augmentations are applied [18].

Table 1. Summary of bipartite interaction graphs used for evaluating the matching system.

Dataset	# Users (M)	# Items (M)	# Edges (M)
Global Marketplace	120.3	42.1	3,850
Regional Retail	32.7	8.9	740
Long-Tail Vertical	11.4	6.2	210
High-Churn Flash Sales	9.8	3.5	185
Cold-Start Sellers	6.1	1.7	96

Table 2. Algorithmic variants evaluated for large-scale bipartite graph matching.

Variant	Parallelism Model	Blocking Strategy	Description
Greedy-Seq	Single-threaded	None	Baseline greedy matching on a single core
Greedy-Shared	Shared-memory	Vertex sharding	Parallel greedy with partitioned vertex sets
Hungarian-Sharded	Distributed	Item sharding	Approximate Hungarian over sharded items
Blocked-PA	Shared-memory	Edge blocking	Proposed blocked parallel augmentation algorithm
Blocked-PA+Rebal	Shared-memory	Adaptive blocking	Blocked parallel augmentation with dynamic load rebalancing

The algorithm begins with a partition of vertices into blocks. A simple approach assigns all supply vertices V to blocks by a hash of their identifiers and assigns demand vertices U by either the same hash on a canonical key or by colocating a demand vertex with the block containing most of its neighbor supply vertices. In practice, supply-based partitioning is often convenient because supply entities are fewer than demand contexts and have more persistent state, including capacities and exposure history. Each block owns a subset $V_i \subseteq V$ and a working set of incident edges $E_i = \{(u, v) \in E : v \in V_i\}$. Demand vertices may appear in multiple blocks as ghost vertices, with their state either replicated read-only or accessed via a lightweight service.

Given a current matching M, each block performs local augmentation searches to propose improvements [19]. The basic local search treats the block-induced subgraph as a graph in which supply vertices are owned and demand vertices are partially observed through their incident edges in the block. The block attempts to route residual demand capacity to residual supply capacity through alternating paths that primarily traverse owned supply vertices, thereby increasing locality. To do this efficiently, the block constructs a layered graph using a BFS-like procedure similar in spirit to Hopcroft–Karp, but adapted to b-matching and to partial ownership. From a set of active demand vertices u with residual capacity, the search explores unmatched edges (u, v) into owned supply vertices v, then from those supply vertices explores matched edges back to demand vertices, and continues until it reaches owned supply vertices with residual capacity. Because demand vertices are not owned, the block treats their matched edges as pointers to supply vertices which may be in other blocks, and can optionally stop or continue based on a boundary policy [20].

A key design choice is the boundary policy, which determines whether local searches are allowed to traverse matched edges that leave the block. A strict local policy prohibits crossing block boundaries, yielding purely block-local augmentations that are conflict-free by construction if demand capacities are also locally enforced. However, strict locality can stall when the needed alternating paths cross blocks. A relaxed policy allows boundary crossings up to a bounded depth, at the cost of potential conflicts that must be resolved at commit [21]. Blocked parallel augmentation uses a relaxed policy with bounded exploration depth L and a preference for paths with fewer boundary crossings, thereby

Table 3. Offline throughput and latency comparison on the largest marketplace graph.

Method	Throughput (M edges/s)	Speedup vs Greedy-Seq	Mean Latency (ms)
Greedy-Seq	12.4	1.0×	438
Greedy-Shared (32 cores)	163.8	13.2×	71
Hungarian-Sharded (32 workers)	97.5	7.9×	126
Blocked-PA (32 cores)	221.3	17.8×	52
Blocked-PA+Rebal (32 cores)	247.9	20.0×	46

Table 4. Matching quality metrics averaged across all marketplace segments.

Method	Precision@50	Recall@50	User Coverage (%)
Greedy-Seq	0.214	0.132	91.8
Greedy-Shared	0.217	0.135	91.9
Hungarian-Sharded	0.229	0.143	92.0
Blocked-PA	0.236	0.149	96.3
Blocked-PA+Rebal	0.239	0.151	97.1

enabling progress while controlling conflict rates.

Each local search produces a set of candidate augmentations represented as sequences of edge flips. Because b-matching allows multiple units per vertex, the representation must account for per-vertex residual capacity. One convenient representation is to treat each matched edge as occupying one unit and maintain residual counts. An augmentation path P is then an ordered list of edges $(e_1, e_2, \ldots, e_{2t+1})$ where odd-indexed edges are currently unmatched and even-indexed edges are currently matched. Applying P toggles membership of all edges in P, increasing the matched edge count by one [22]. The block assigns a score to each candidate augmentation, typically the sum of effective weights on newly added edges minus the sum of effective weights on removed edges. In the unweighted or cardinality-first case, the score can be treated as one for all valid augmentations, and tie-breaking can be based on deterministic identifiers to reduce non-determinism across runs.

The block then performs an internal packing step to reduce conflicts before global commit. It selects a subset of candidate augmentations that are internally non-conflicting with respect to owned supply vertices, ensuring that no owned supply vertex exceeds its capacity when all selected augmentations are applied [23]. Because demand vertices may be shared across blocks, the block can also enforce a conservative demand constraint by limiting the number of augmentations that touch the same demand vertex within the block. This packing step can be implemented by greedily accepting augmentations in descending score order while tracking incremental degree changes on owned vertices and approximate changes on ghost vertices.

After local packing, each block emits a proposal set to a commit protocol. The commit protocol has two tasks: detect conflicts between proposals from different blocks and apply a consistent subset. Conflicts arise when proposals share a vertex whose capacity would be exceeded or when proposals attempt to flip the same edge in incompatible ways [24]. The protocol uses deterministic arbitration keyed by vertex ownership. Since supply vertices are owned by blocks, supply-side conflicts can be resolved by having the owning block arbitrate all proposals that touch its vertices. Demand vertices may not have a unique owner, so demand-side conflicts are resolved by assigning a virtual owner function, for example a hash of the demand identifier, and routing demand-touching proposals to the corresponding arbiter. In both cases, the arbiter receives competing claims and selects a subset that respects capacity and maximizes an arbitration objective such as total proposal score [25].

Commit proceeds in rounds. In each round, blocks first reserve capacity on owned supply vertices

Table 5. Scalability of the blocked parallel augmentation engine under increasing core counts.

# Cores	Throughput (M edges/s)	Parallel Efficiency (%)	P99 Latency (ms)
1	12.4	100.0	612
4	47.2	95.2	226
8	93.7	94.5	132
16	175.1	88.3	87
32	247.9	77.9	61
64	311.4	62.7	55

Table 6. End-to-end latency breakdown for the online recommendation candidate service.

Pipeline Stage	Time per Request (ms)	Share of Total (%)
Request parsing & validation	3.2	4.8
Graph block retrieval	16.9	25.4
Blocked augmentation kernel	33.5	50.4
Post-processing & deduplication	7.8	11.7
Feature enrichment	3.9	5.9
Serialization & network egress	2.1	1.8

for their selected proposals by writing tentative reservations into a distributed store with compare-and-swap semantics. Reservations include a proposal identifier, the set of affected owned vertices, and the net degree changes. If a reservation fails due to contention, the block retracts the conflicting proposal and may attempt to replace it with a lower-priority alternative. Once reservations succeed for a proposal across all its owned supply vertices, the proposal is eligible for finalization [26]. Finalization flips the edges in the matching state and updates residual capacities. Edge flips are written as idempotent operations tagged with the proposal identifier, allowing retries without double application.

To keep the protocol lightweight, the system avoids global synchronization barriers. Instead, each round is bounded by a time or iteration budget, and proposals that cannot be committed are deferred [27]. The algorithm terminates after R rounds or when progress falls below a threshold, such as a small increase in yield. In batch mode, termination can be based on a fixed compute budget aligned with the pipeline schedule. In incremental mode, termination is often driven by the event queue size and latency requirements.

Theoretical correctness properties follow from the reservation and finalization semantics. The invariant maintained is that the matching state always satisfies capacity constraints on all owned vertices, and that each committed proposal corresponds to a valid alternating path in the matching state at the time of reservation [28]. Because the state may change between local search and commit, the protocol must validate proposals during reservation, at least for the edges and vertices whose ownership is involved. Validation checks that edges intended to be removed are still matched and edges intended to be added are still eligible and unmatched, and that residual capacities exist at endpoints. If validation fails, the proposal is discarded. This validation step is essential in high-churn settings where eligibility edges can appear and disappear frequently [29].

Blocked parallel augmentation can be extended from cardinality maximization to approximate weight maximization. One approach is to treat the objective as lexicographic: first satisfy as much demand quota as possible, then improve total weight. This can be achieved by running cardinality-focused rounds until yield saturates, then running weight-improving rounds that allow augmentations which do not increase cardinality but increase total weight via alternating cycles or via augmenting paths that swap lower-weight matches for higher-weight ones while keeping cardinality constant. An-

Table 7. Peak memory footprint of different matching engines on the largest dataset.

Component	Peak Memory (GB)	Relative to Greedy-Seq (%)
Greedy-Seq engine	14.2	100.0
Greedy-Shared engine	18.9	133.1
Hungarian-Sharded engine	24.7	173.9
Blocked-PA engine	16.1	113.4
Blocked-PA+Rebal engine	17.0	119.7

Table 8. Sensitivity of the blocked augmentation kernel to block size on the production workload.

Block Size (edges)	Throughput (M edges/s)	Matched Users (%)
4,096	189.4	95.8
8,192	213.7	96.5
16,384	247.9	97.1
32,768	241.2	97.0
65,536	228.6	96.8
131,072	210.3	96.0

other approach uses a cost-scaling view in which weights are discretized and the algorithm proceeds in phases, each phase considering only edges above a threshold and augmenting cardinality on that subgraph, then gradually lowering the threshold. This yields controllable approximation while preserving the augmentation mechanism [30].

A useful analytical lens is to model conflicts probabilistically as a function of block size and degree distribution. Suppose supply vertices are partitioned uniformly into B blocks and consider a proposal that touches s owned supply vertices and d demand vertices. If demand ownership is hashed into B virtual owners, the probability that two random proposals contend on a given demand owner increases with d and with the concentration of demand identifiers. Similarly, contention on supply vertices increases with the skew in supply degrees and with the concentration of high-weight edges around popular supply [31]. Blocking reduces supply contention by ensuring that proposals are mostly internal to a block, but it can increase demand contention if many blocks compete for the same set of high-activity demand vertices. This motivates partitioning strategies that align demand contexts with their dominant supply neighborhoods, reducing cross-block interaction.

The algorithm also supports a mechanism for cold-start supply and long-tail balancing. If weights are derived from historical engagement, new listings may have lower or noisier weights and may be crowded out. The system can incorporate a prior boost or exploration weight component and then rely on capacity constraints and penalties to spread exposure [32]. In the augmentation framework, this simply changes edge scores, but operationally it affects conflict rates because boosted edges may create new contention patterns. Blocking helps by localizing such contention to blocks where new listings reside, allowing exploration without global coordination costs.

4. Distributed Architecture and Implementation

The system architecture is organized around a graph construction pipeline, a matching state store, a blocked augmentation compute layer, and downstream serving interfaces that consume the produced candidate sets. The architecture must support large-scale graphs, frequent updates, and predictable resource usage under strict scheduling [33]. A practical implementation treats the matching computation as a distributed job that can run either periodically over snapshots or continuously over micro-batches of updates.

Table 9. Online A/B testing results for the proposed matching system on marketplace traffic.

Experiment Variant	CTR Lift (%)	Conversion Lift (%)	Revenue / 1K Impressions (% Lift)
Greedy-Shared (control)	0.0	0.0	0.0
Blocked-PA	+2.7	+1.9	+2.1
Blocked-PA+Rebal	+3.4	+2.3	+2.8
Blocked-PA+Rebal+Quality Filter	+3.1	+2.9	+3.2

Table 10. Robustness of the blocked parallel augmentation engine under operational stress scenarios.

Failure Scenario	Throughput (M edges/s)	Relative to Baseline (%)	P99 Latency (ms)
Baseline (no failure)	247.9	100.0	61
Single-core outage (1/32)	232.5	93.8	65
Skewed traffic spike (×4)	218.1	88.0	73
Hot-item burst (top-0.1% items)	224.7	90.7	69
Storage throttling (-20% I/O)	205.3	82.8	81
Network jitter (p95 = 15ms)	212.9	85.9	78

Graph construction begins by defining demand vertices U at the granularity appropriate for the product. For a home feed, a demand vertex may represent a user-session context that includes location, device, and recent interactions. For a search surface, it may represent a query-session context. Supply vertices V represent listings or supply units with associated metadata such as location, category, seller, inventory state, and policy attributes [34]. Hard eligibility is computed by joining demand context constraints with supply metadata, often using precomputed region indices and policy tables. Because exact eligibility joins can be expensive, the system employs hierarchical filters: coarse eligibility is computed first to prune the candidate space, followed by more expensive checks on the remaining edges.

The system then generates potential edges using retrieval methods. A common approach uses embedding-based approximate nearest neighbor search to obtain a shortlist of supply items per demand context, followed by re-scoring with a lightweight model [35]. The output is a set of edges with weights, plus metadata needed for later validation such as eligibility reasons and timestamps. Edges are assigned a time-to-live reflecting expected validity; for example, inventory-driven edges may expire quickly, while category-based edges may persist longer. These TTLs are important for incremental mode because they allow the system to retire stale edges without explicit delete events.

The matching state store maintains the current matching M and residual capacities for vertices. Because the algorithm performs edge flips and capacity updates, the store must support atomic conditional updates or transactional semantics at least at the granularity of a vertex [36]. Many implementations use a sharded key-value store in which each supply vertex record contains its matched demand neighbors, current degree, capacity, and a version counter. Demand vertex state may be stored similarly if demand capacities are enforced strictly. To support idempotent commits, each record also maintains a small cache of recently applied proposal identifiers, enabling duplicate suppression during retries.

Partitioning is central to performance. Supply vertices are assigned to blocks using a stable partition function to minimize reshuffling across runs [37]. Stable partitioning reduces data movement and allows caches of adjacency lists and matching state to remain warm. Within each block, the compute worker loads the adjacency lists for owned supply vertices, the current matched neighbors, and relevant demand ghost state. For demand ghost state, the worker may only need residual capacity and the identity of matched supply neighbors that are referenced during alternating path exploration.

Table 11. Operational configurations of the matching system across different production environments.

Deployment Environment	Target P99 Latency (ms)	Avg Candidate Set Size
Realtime recommendations (homepage feed)	80	600
Realtime recommendations (search & browse)	60	400
Nearline re-ranking warm-up (feature store)	500	1,200
Daily backfill for inactive users	2,000	800
Seller-side analytics & insights	1,000	300
Experimentation sandbox (offline tuning)	N/A	2,000

This state can be fetched on demand from the store, but on-demand fetches can dominate runtime in skewed graphs [38]. To reduce overhead, the system prefetches demand ghost state for high-degree demand vertices or for demand vertices that appear frequently across adjacency lists, using a batched read protocol.

The blocked augmentation compute layer operates as a set of workers, one per block or per shard group. Each worker performs three loops: local search, proposal packing, and commit. Local search constructs alternating trees or layered graphs starting from demand vertices with residual capacity. Identifying such demand vertices can be done by scanning demand quotas and current degrees, but scanning the entire demand set is expensive [39]. Instead, workers maintain an active frontier derived from unmet demand contexts observed during the previous run, plus newly arrived demand contexts in incremental mode. The worker also includes demand contexts that are connected to supply vertices with newly available capacity, since such capacity can unlock new augmentations.

Local search must handle the fact that adjacency lists are truncated. If retrieval provides only the top m neighbors per demand vertex, then augmenting paths may not exist even if they exist in the full implicit graph [40]. The system mitigates this by increasing m for high-value or high-need demand vertices, by maintaining a secondary retrieval fallback that can expand neighborhoods when yield is low, and by allowing the augmentation algorithm to request additional neighbors on the fly for certain demand vertices. On-the-fly expansion is constrained by budgets to prevent runaway compute, and it is typically limited to a small fraction of demand vertices per run.

Proposal packing inside a block is implemented to be cache-friendly and parallel within the worker. Workers generate many candidate paths, but only a subset can be committed due to capacity. Packing uses a combination of local capacity accounting and heuristic conflict avoidance on ghost vertices [41]. Because ghost vertex degrees are only approximately known, the packer uses conservative bounds. For example, if a demand vertex is likely to be targeted by many blocks, the packer limits the number of proposals involving that demand vertex to a small number, relying on other blocks to handle remaining capacity. This conservative approach reduces wasted commit attempts at the cost of slower convergence, which is acceptable when the system runs multiple rounds.

Commit uses a reservation protocol over the matching state store [42]. For supply vertices, reservation attempts update the vertex record conditionally on its version counter and residual capacity. For demand vertices, if strict enforcement is desired, similar reservations are performed on a demand record. In many candidate generation settings, demand capacities can be enforced at output time by truncation, allowing commit to focus on supply capacities. However, ignoring demand capacities can cause wasted work if many proposals add candidates to demand vertices that already exceed quotas. The system therefore supports a hybrid approach in which demand capacities are enforced approximately during packing and strictly during final output assembly [43].

After a round of commits, workers update local caches and compute new active frontiers. They also emit observability metrics such as yield improvements, conflict rates, reservation failures, and

the distribution of matched degrees. These metrics are essential for operational tuning because the algorithm's performance depends strongly on graph properties that can drift over time, such as changes in retrieval behavior or supply churn.

Downstream consumption depends on the product surface [44]. For offline candidate sets, the system materializes per-demand candidate lists in a distributed file store, sorted by edge weight or by a composite score, and exposes them to online services via a low-latency lookup cache. For near-real-time surfaces, the system may stream incremental updates to candidate lists, where each update adds or removes supply candidates for specific demand contexts. Because candidate lists are used as inputs to ranking, which may apply additional filters and personalization, the matching system focuses on feasibility and diversity, leaving fine-grained relevance ordering to downstream models.

An important engineering consideration is the interaction with policy and eligibility updates. If a policy change invalidates a large portion of edges, the system must quickly remove them from candidate lists [45]. In blocked augmentation, such invalidations appear as edge deletions that can break existing matches. The system handles this by validating matched edges during each run and by maintaining a repair queue of vertices whose matches were removed. Repair uses the same augmentation mechanism but prioritizes restoring feasibility for affected demand vertices while respecting new constraints.

5. Consistency, Fault Tolerance, and Data Engineering Considerations

Distributed matching systems must balance correctness, performance, and operational robustness [46]. The primary correctness requirement in marketplace candidate generation is that hard constraints are never violated: a supply vertex should not be matched beyond its capacity, and ineligible edges should not appear in output candidate sets. Secondary requirements involve stability and reproducibility. While exact determinism is often infeasible at scale, especially under asynchronous commits, the system should avoid chaotic behavior where small changes in input cause large swings in allocation.

Consistency is managed through ownership and validation. Ownership assigns each supply vertex to exactly one block, and only the owning worker is allowed to finalize changes to that vertex's matched neighbor set [47]. This constraint simplifies reasoning about capacity because capacity is enforced at a single point. Demand vertices may be non-owned, but their capacity can be enforced either strictly via virtual ownership or approximately via output truncation. In practice, strict demand enforcement is most valuable when demand quotas are small and when wasted work is expensive, such as in low-latency incremental mode. When quotas are large relative to neighborhood sizes, approximate enforcement is often sufficient [48].

Validation addresses staleness. Since edges can become ineligible due to inventory changes or policy updates, the system validates edges at two points: during local search when constructing candidate paths, and during commit when reserving and finalizing. Local validation filters out obviously ineligible edges using cached metadata and TTL checks. Commit-time validation performs definitive checks, potentially consulting authoritative sources for inventory state or policy. To keep commit fast, definitive checks are cached and batched, and are often limited to edges that would be newly added [49]. Existing matched edges are revalidated periodically or when triggered by events.

Fault tolerance is achieved through idempotent operations and checkpointing. Each proposal has a unique identifier derived from block id, round id, and a local sequence number. Reservations and finalizations record the proposal id in vertex records so that retries do not double-apply changes [50]. Workers periodically checkpoint their active frontiers and local progress counters so that after a restart they can resume without redoing excessive work. In batch mode, the system can simply

restart the job from the last successful snapshot, but in incremental mode, preserving continuity reduces candidate churn and improves freshness.

Candidate churn is itself an operational metric. Even if two matchings have similar objective values, frequent changes in candidate sets can harm user experience and downstream model calibration. The system reduces churn by seeding each run with the previous matching, by applying penalties for repeated exposure rather than hard exclusions that force abrupt changes, and by limiting the number of augmentation rounds per run [51]. Blocking also contributes because it localizes changes; when a block updates, the effects are concentrated around the supply vertices it owns.

Data engineering for the graph is nontrivial due to size and skew. Adjacency lists must be stored and accessed efficiently. A common approach stores for each supply vertex a compressed list of incident demand vertices along with edge weights and metadata [52]. Demand vertices may have very large adjacency lists if they represent popular contexts, which can create hotspots. The system mitigates this by favoring supply-centric storage and by representing demand adjacency implicitly through retrieval indices rather than explicit lists. During local search, when the algorithm needs to traverse from a demand vertex to candidate supply neighbors, it can either consult a precomputed neighbor list or query a retrieval service. This hybrid reduces storage but introduces variability in latency, so it is bounded by budgets and prefetched for high-frequency demand contexts.

Degree skew affects both compute and conflicts [53]. High-degree supply vertices are likely to be contended and can dominate commit time. Since such vertices are often popular listings or sellers, the system may assign them special treatment. One approach is to cap their effective capacity per run and to distribute additional exposure via time-slicing across runs. Another approach is to split a high-capacity vertex into multiple virtual vertices owned by different blocks, each with a fraction of the capacity, thereby parallelizing updates [54]. This virtual splitting must be done carefully to preserve eligibility and to avoid duplicating candidates in output lists.

Incremental updates require careful handling of deletions. When a listing becomes unavailable, all edges incident to it must be removed, and any matches involving it must be released. The system processes such events by marking the supply vertex as inactive and setting its capacity to zero, then removing its matched edges during the next run or immediately if low latency is required. Immediate removal can be implemented by a push-based invalidation that updates the state store and emits candidate list deltas [55]. The blocked augmentation algorithm then treats released demand vertices as high-priority for repair.

Another critical consideration is the interplay between matching and ranking. The matching system outputs feasible candidates, but ranking models may rely on exposure patterns that matching influences. For example, if matching increases long-tail supply coverage, the distribution of observed engagements changes, potentially affecting model training. To manage this, the system supports controlled rollout and experimentation, where the matching layer can be enabled for a fraction of traffic or with varying parameters [56]. Logging is designed to attribute which candidates were produced by matching and which by baseline retrieval, enabling offline analysis of effects.

Privacy and security constraints also shape the design. Demand contexts often contain sensitive attributes or derived signals. The system therefore minimizes the propagation of raw demand features into the matching computation [57]. Edge weights can be computed upstream and represented as opaque scores, and demand identifiers can be pseudonymized. Eligibility checks based on sensitive attributes are performed upstream and reflected as inclusion or exclusion of edges rather than as explicit features in the matching layer. This separation reduces the risk of exposing sensitive information in logs and state stores.

6. Evaluation Methodology and Results

Evaluating a matching-based candidate generation system requires measuring both algorithmic quality and systems performance under realistic workload conditions. Quality should be assessed not only by objective value such as total edge weight but also by feasibility metrics that reflect operational goals: demand yield, supply coverage, distribution across sellers or categories, and candidate diversity [58]. Systems performance includes throughput, end-to-end latency, resource usage, and robustness under churn and skew. Because production graphs and signals are often proprietary, the evaluation described here focuses on trace-driven workloads constructed from anonymized interaction logs and on synthetic graphs parameterized to match observed degree distributions and churn patterns.

The baseline for comparison is a retrieval-only candidate generator that produces top-k_u candidates per demand vertex using embedding search and lightweight scoring, followed by hard filtering and truncation. This baseline is representative of common practice and highlights the feasibility gap: many retrieved candidates are filtered out or exceed supply exposure budgets, leading to underfilled demand quotas and concentrated supply exposure [59]. The matching system is evaluated as a post-processing layer that takes the retrieval graph as input and outputs a feasible candidate set under supply capacities and demand quotas.

Quality is first measured by demand yield. In settings where supply capacity is tight relative to demand, yield is limited by feasibility, and the key question is whether matching can allocate scarce supply more effectively than independent truncation. Blocked augmentation tends to improve yield by reallocating matches away from over-subscribed supply vertices toward feasible alternatives that retrieval surfaced but baseline truncation may not select due to local ordering. The improvement is more pronounced when retrieval neighborhoods include reasonable substitutes and when capacities are moderately tight, because augmentation can resolve contention by rerouting along alternating paths [60]. When capacities are extremely tight and neighborhoods are narrow, yield is dominated by scarcity and gains are smaller.

Supply-side metrics show complementary effects. Supply coverage, defined as the fraction of active supply vertices that receive at least one match, typically increases under matching because capacities prevent a small set of popular items from absorbing most allocations. The degree distribution on supply becomes less heavy-tailed, although the extent depends on how weights and penalties are configured [61]. If weights are steep and penalties are weak, the system may still concentrate on high-scoring supply until capacities bind, after which it spreads. Stronger penalty terms can encourage broader distribution at some cost in total weight.

Total weight comparisons must be interpreted carefully because the baseline may generate infeasible candidates that are later dropped, and thus its apparent top-k weight does not translate to realized exposures. A fair comparison measures realized weight under the same feasibility constraints. Under this lens, matching can achieve comparable or higher realized weight because it avoids wasted high-weight candidates that cannot be served due to capacity [62]. In regimes where capacity is loose and filtering is minimal, matching may slightly reduce weight if it prioritizes coverage or if approximation limits prevent full optimization, but the difference can be controlled by running more rounds or by tuning the augmentation score to favor weight improvements after yield saturation.

Algorithmic parameters affect convergence and quality. Block size controls the locality of augmentations and the conflict rate. Larger blocks allow richer local augmentations and reduce the need for boundary crossings, which can improve convergence per round [63]. However, larger blocks increase memory footprint and can reduce parallelism, potentially increasing latency. Smaller blocks improve parallelism but can increase conflicts and require more rounds. Empirically, there is often

an intermediate block size that balances these effects, and the optimum depends on degree skew and network overhead.

Augmentation depth L similarly trades compute for quality. Shallow augmentations are fast and local but may fail to resolve complex contention patterns that require longer alternating paths [64]. Increasing L improves the ability to reroute matches, but it increases the number of ghost state fetches and raises the probability that a proposal becomes stale before commit. A practical strategy is to use a small L for early rounds when many easy augmentations exist, and selectively increase L for a subset of hard cases, such as high-need demand vertices that remain underfilled after initial rounds.

Conflict behavior is a primary determinant of performance. Reservation failure rates provide a direct measure [65]. Under stable partitioning and moderate skew, reservation failures can remain low, enabling near-linear scaling across blocks. Under high skew, a few popular supply vertices can attract many proposals, causing repeated reservation failures and wasted work. Mitigations include capping how many proposals can target a given supply vertex per round, prioritizing proposals that diversify away from hotspots, and using a backoff mechanism that temporarily deprioritizes over-contended vertices. These mitigations reduce wasted work but can slow convergence if hotspots are truly essential for yield.

Systems performance is evaluated by end-to-end runtime for batch runs and by update latency for incremental runs [66]. In batch mode, runtime decomposes into graph loading, local search, commit, and materialization. Graph loading cost depends on how adjacency lists are stored and on block size. Local search cost depends on the number of active demand vertices and the exploration depth. Commit cost depends on the number of successful proposals and on reservation contention [67]. Materialization cost depends on output size and sorting. In well-tuned configurations, local search and commit dominate, and optimizations focus on reducing ghost fetches and reservation round trips.

Incremental mode introduces additional constraints. Update latency must be bounded, and the system must handle bursts of events such as large inventory changes. The blocked augmentation approach supports micro-batching, where events are accumulated over a short window and processed as a small update graph [68]. The algorithm then performs a limited number of augmentation rounds starting from the current matching state. This yields low latency while still enabling rerouting around removed edges. The trade-off is that incremental runs may not fully re-optimize, leading to some drift in quality over time. Periodic batch refreshes can correct drift, and the system can monitor drift via yield and weight metrics to trigger refresh when needed [69].

Robustness is evaluated under synthetic stress tests that vary degree distributions and churn rates. When churn is high, many proposals fail validation because edges become ineligible before commit. This increases wasted work and can reduce yield if repair cannot keep up. The system mitigates this by prioritizing stable edges, using shorter TTLs for volatile edges, and reducing augmentation depth in high-churn windows. When degree skew is extreme, performance depends on hotspot mitigation and on the ability of retrieval to provide substitutes; matching cannot allocate alternatives that do not exist in the input graph [70].

A final aspect of evaluation concerns downstream effects. Candidate sets produced by matching can change the distribution of items seen by users and thus affect engagement and marketplace health metrics. While such effects are product-specific and require online experimentation to quantify, the matching system provides mechanisms to control them: supply capacities can be tuned, penalties can be adjusted to regulate exposure, and the system can enforce minimum allocation for certain supply segments if required by policy. Offline simulations using logged propensities can provide preliminary signals, but they are limited by bias in logged data [71]. Consequently, the system is

designed to be experiment-friendly, with clear logging of candidate origins and deterministic parameterization to support analysis.

7. Conclusion

This paper presented a scalable bipartite graph matching system for marketplace recommendation candidate generation, centered on blocked parallel augmentation. By framing candidate selection as a capacity-constrained b-matching problem, the system directly encodes supply-side exposure limits and demand-side candidate quotas, reducing the reliance on late-stage filtering that can waste computation and harm coverage. Blocked parallel augmentation enables distributed execution by decomposing the graph into blocks, performing localized augmenting path searches in parallel, and committing non-conflicting improvements through reservation-based arbitration and validation. The approach aligns with operational realities of marketplaces, including dynamic eligibility, degree skew, and the need for controllable trade-offs between quality and throughput [72].

The design emphasizes feasibility invariants, idempotent updates, and partitioning strategies that localize contention. It supports both batch recomputation and incremental repair, allowing freshness-sensitive surfaces to maintain feasible candidate sets under churn while periodic refreshes mitigate drift. Evaluation considerations highlight that quality must be assessed through realized feasible yield and supply distribution metrics, not only by raw retrieval weights, and that systems performance is dominated by conflict rates and ghost state access patterns. While augmentation-based methods do not remove the fundamental dependence on the quality of the retrieved graph, they provide a structured way to enforce global constraints and to reallocate candidates under capacity pressure. The resulting system offers a practical template for integrating matching into large-scale recommendation pipelines where feasibility, coverage, and operational predictability are as important as local relevance scores [73].

References

[1] Asier Estevan. “An Approach to Distributed Systems from Orderings and Representability”. In: *Bulletin of the Iranian Mathematical Society* 50.3 (Apr. 8, 2024). DOI: 10.1007/s41980-024-00865-0.

[2] Mihály Héder et al. “The Past, Present and Future of the ELKH Cloud”. In: *Információs Társadalom* 22.2 (Aug. 31, 2022), pp. 128–128. DOI: 10.22503/inftars.xxii.2022.2.8.

[3] Zhen Lin, Lianjie Cao, Faraz Ahmed, Hui Lu, and Puneet Sharma. “When Caching Systems Meet Emerging Storage Devices: A Case Study”. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems*. ACM, July 9, 2023, pp. 37–43. DOI: 10.1145/3599691.3603413.

[4] Rahul Malik, Sangkyum Kim, Xin Jin, Chandrasekar Ramachandran, Jiawei Han, Indranil Gupta, and Klara Nahrstedt. “MLR-index: An index structure for fast and scalable similarity search in high dimensions”. In: *International Conference on Scientific and Statistical Database Management*. Springer. 2009, pp. 167–184.

[5] Armin Balalaie and James A. Jones. “SoCC - Towards a Library for Deterministic Failure Testing of Distributed Systems”. In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM, Nov. 20, 2019, pp. 486–486. DOI: 10.1145/3357223.3366026.

[6] Girish Talmale and Urmila Shrawankar. “Cluster Based Real Time Scheduling for Distributed System”. In: *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 10.2 (Mar. 19, 2021), pp. 137–156. DOI: 10.14201/adcaij2021102137156.

[7] Michael F. Rehme, Stefan Zimmer, and Dirk Pflüger. “Hierarchical Extended B-splines for Approximations on Sparse Grids”. In: Germany: Springer International Publishing, Oct. 22, 2021, pp. 187–203. DOI: 10.1007/978-3-030-81362-8_8.

[8] Cyrille Chenavier and Maxime Lucas. *The Diamond Lemma for non-terminating rewriting systems using deterministic reduction strategies*. June 28, 2019.

[9] Ege Berkay Gulcan, João Neto, and Burcu Kulahcioglu Ozkan. "Generalized Concurrency Testing Tool for Distributed Systems". In: *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, Sept. 11, 2024, pp. 1861–1865. DOI: 10.1145/3650212.3685309.

[10] Ammad Ul Islam, Ming Yang, Amna Iqbal, Anqi Xu, Sijuan Chen, and Sohail Asghar. "Trustworthy Distributed Systems: A Machine Learning Approach". In: *2025 IEEE 20th Conference on Industrial Electronics and Applications (ICIEA)*. IEEE, Aug. 3, 2025, pp. 1–6. DOI: 10.1109/iciea65512.2025.11148715.

[11] Giovanni Farina. *Tractable Reliable Communication in Compromised Networks*. Dec. 21, 2020.

[12] Nihar Vipulkumar Patel. "Estimating the Causal Effects of Pricing Interventions in Two-Sided Marketplaces Under Interference and Spillovers". In: *Frontiers in Health Informatics* 13.8 (2024), pp. 7572–7601.

[13] Konrad Wolsing, Linus Roepert, Jan Bauer, and Klaus Wehrle. "Anomaly Detection in Maritime AIS Tracks: A Review of Recent Approaches". In: *Journal of Marine Science and Engineering* 10.1 (Jan. 14, 2022), pp. 112–112. DOI: 10.3390/jmse10010112.

[14] Asier Estevan Muguerza. *A new approach on distributed systems: orderings and representability*. Jan. 1, 2023. DOI: 10.48550/arxiv.2306.16888.

[15] Dimitris Xenakis, Eric Samikwa, Jesutofunmi Ajayi, Antonio Di Maio, Torsten Braun, and Katja Schlegel. "Towards Personality Detection and Prediction using Smartphone Sensor Data". In: *2023 21st Mediterranean Communication and Computer Networking Conference (MedComNet)*. IEEE, June 13, 2023, pp. 121–130. DOI: 10.1109/medcomnet58619.2023.10168869.

[16] David Basin, Matthieu Gras, Srdan Krstic, and Joshua Schneider. "RV - Scalable Online Monitoring of Distributed Systems". In: Germany: Springer International Publishing, Oct. 2, 2020, pp. 197–220. DOI: 10.1007/978-3-030-60508-7_11.

[17] Kushal Johari, Gaurav KUMAR Agarwal, and Y. D.S. Arya. "Server Load Balancing Analysis for Client Task Assignment in Distributed Systems". In: *ECS Transactions* 107.1 (Apr. 24, 2022), pp. 17605–17621. DOI: 10.1149/10701.17605ecst.

[18] Pascal Weisenburger, Mirko Köhler, and Guido Salvaneschi. *Distributed System Development with ScalaLoci*. Sept. 23, 2018. DOI: 10.1145/3276922.

[19] Venkata Naga Sai Kiran. "Chaos Engineering for Building Resilient Distributed Systems". In: *International Journal of Science and Research (IJSR)* 9.3 (Mar. 5, 2020), pp. 1678–1689. DOI: 10.21275/sr24716231253.

[20] Vivek 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*. IEEE. 2006, pp. 1–6.

[21] J. Dinal Herath, Ping Yang, and Guanhua 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*. ACM, Apr. 26, 2021, pp. 29–40. DOI: 10.1145/3422337.3447833.

[22] Wiktor B. Daszczuk. "Fairness in Distributed Systems Verification". In: Germany: Springer International Publishing, Mar. 17, 2019, pp. 139–159. DOI: 10.1007/978-3-030-12835-7_9.

[23] Arturo Bretas and Ashutosh Dutta. "Cyber-Physical Power Systems Protection: The Byzantine Cybersecurity Framework". In: *2023 IEEE Kansas Power and Energy Conference (KPEC)*. IEEE, Apr. 27, 2023, pp. 1–5. DOI: 10.1109/kpec58008.2023.10215452.

[24] Shenao Wang, Yanjie Zhao, Xinyi Hou, and Haoyu Wang. "Large Language Model Supply Chain: A Research Agenda". In: *ACM Transactions on Software Engineering and Methodology* 34.5 (May 24, 2025), pp. 1–46. DOI: 10.1145/3708531.

[25] Fabio Checconi, Jesmin Jahan Tithi, and Fabrizio Petrini. *Ridgeline: A 2D Roofline Model for Distributed Systems*. Jan. 1, 2022. DOI: 10.48550/arxiv.2209.01368.

[26] Nan Xia. "Design strategy and case study of distributed system resilience in the chinese context". In: *Strategic Design Research Journal* 13.3 (Dec. 23, 2020), pp. 355–363. DOI: 10.4013/sdrj.2020.133.05.

[27] Yiyang Jia and Xinfeng Shu. "Visual Model Checking Distributed System". In: *Proceedings of the 2022 5th International Conference on Artificial Intelligence and Pattern Recognition*. ACM, Sept. 23, 2022, pp. 285–291. DOI: 10.1145/3573942.3574023.

[28] Nihar Vipulkumar Patel. "Selection Bias in Two-Sided E-Commerce Marketplaces: A Framework for Propensity Score Matching Implementation". In: *Journal of Business Intelligence Systems and Computational Social Science Applications* 11.5 (2021), pp. 1–20.

[29] Felicia Schnell. *Multicast Communication for Increased Data Exchange in Data- Intensive Distributed Systems*. Jan. 1, 2018.

[30] Cristina L. Abad, Eduardo Ortiz-Holguin, and Edwin F. Boza. "SIGCSE - Have We Reached Consensus? An Analysis of Distributed Systems Syllabi". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. ACM, Mar. 3, 2021, pp. 1082–1088. DOI: 10.1145/3408877.3432409.

[31] Valeriy A. Kolesnikov. *CMIS - Autonomous distributed systems and their simulation.* June 2, 2020.

[32] *OSDI - Scalable Memory Protection in the PENGLAI Enclave*. June 23, 2021.

[33] Adrian Francalanza, Jorge A. Pérez, and César Sánchez. "Lectures on Runtime Verification - Runtime Verification for Decentralised and Distributed Systems". In: 10457. Germany: Springer International Publishing, Feb. 11, 2018, pp. 176–210. DOI: 10.1007/978-3-319-75632-5_6.

[34] Michael A. Helcig and Stefan Nastic. "FedCCL: Federated Clustered Continual Learning Framework for Privacy-Focused Energy Forecasting". In: *2025 IEEE 9th International Conference on Fog and Edge Computing (ICFEC)*. IEEE, May 19, 2025, pp. 50–57. DOI: 10.1109/icfec65699.2025.00012.

[35] Konrad Wolsing, Lea Thiemt, Christian van Sloun, Eric Wagner, Klaus Wehrle, and Martin Henze. "Can Industrial Intrusion Detection Be SIMPLE?" In: Germany: Springer Nature Switzerland, Sept. 24, 2022, pp. 574–594. DOI: 10.1007/978-3-031-17143-7_28.

[36] *Security, Algorithms and Distributed Systems*. Sept. 1, 2020. DOI: 10.54499/sfrh/bd/150869/2021.

[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*. 2007, pp. 2701–2706.

[38] Anna Lackinger, Pantelis A. Frangoudis, Ivan Čilić, Alireza Furutanpey, Ilir Murturi, Ivana Podnar Žarko, and Schahram Dustdar. "Inference Load-Aware Orchestration for Hierarchical Federated Learning". In: *2024 IEEE 49th Conference on Local Computer Networks (LCN)*. IEEE, Oct. 8, 2024, pp. 1–9. DOI: 10.1109/lcn60385.2024.10639809.

[39] Eduardo M. D. Souza, Nathalia Suzin, Cesar A. Zeferino, and Douglas R. Melo. "Interactive Simulator for RISC-V Assembly Programming". In: *2025 17th Seminar on Power Electronics and Control (SEPOC)*. IEEE, Nov. 9, 2025, pp. 1–8. DOI: 10.1109/sepoc67005.2025.11297719.

[40] Mateusz Kawulok, Stella Maćkowska, Michał Maćkowski, and Dominik Spinczyk. "Reducing Blind Students' Learning Effort with an Audio-Tactile Approach to Understanding Basic Computer Algorithms". In: *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. ACM, June 13, 2025, pp. 472–478. DOI: 10.1145/3724363.3729098.

[41] Johannes Köstler, Hans P. Reiser, Franz J. Hauck, and Gerhard Habiger. "Fluidity: Location-Awareness in Replicated State Machines". In: *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. ACM, Mar. 27, 2023, pp. 192–201. DOI: 10.1145/3555776.3577763.

[42] Nezha Benmoussa, A. Elyamami, Khalifa Mansouri, Mohammed Qbadou, and Elhossein Illoussamen. "A Multi-Criteria Decision Making Approach for Enhancing University Accreditation Process". In: *Engineering, Technology & Applied Science Research* 9.1 (Feb. 16, 2019), pp. 3726–3733. DOI: 10.48084/etasr.2352.

[43] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. *Global Winning Conditions in Synthesis of Distributed Systems with Causal Memory (Full Version)*. Oct. 18, 2021.

[44] Shuai He, Cai Fu, Guanyun Feng, Jianqiang Lv, and Fengyang Deng. "Singular Value Manipulating: An Effective DRL-Based Adversarial Attack on Deep Convolutional Neural Network". In: *Neural Processing Letters* 55.9 (Oct. 17, 2023), pp. 12459–12480. DOI: 10.1007/s11063-023-11428-5.

[45] T Srinivasan, R Chandrasekar, Vivek 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)*. IEEE. 2006, pp. 44–44.

[46] Diletta Olliaro. "Models for Throughput Maximisation in Distributed Systems". In: *ACM SIGMETRICS Performance Evaluation Review* 52.3 (Jan. 9, 2025), pp. 19–22. DOI: 10.1145/3712170.3712178.

[47] Adrijan Rogan, Andrej Kolar-PoŻun, and Gregor Kosec. "A Numerical Study of Combining RBF Interpolation and Finite Differences to Approximate Differential Operators". In: *2025 MIPRO 48th ICT and Electronics Convention*. IEEE, June 2, 2025, pp. 1177–1182. DOI: 10.1109/mipro65660.2025.11132052.

[48] Jorge Herculano, Willians Pereira, Marcelo Guimarães, Reinaldo Cotrim, Alirio de Sá, Flávio Assis, Raimundo Macêdo, and Sérgio Gorender. "MAC approaches to communication efficiency and reliability under dynamic network traffic in wireless body area networks: a review". In: *Computing* 106.8 (June 20, 2024), pp. 2785–2809. DOI: 10.1007/s00607-024-01307-9.

[49] Alain Girault, Gregor Gössler, Rachid Guerraoui, Jad Hamza, and Dragos-Adrian Seredinschi. "FORTE - Monotonic Prefix Consistency in Distributed Systems". In: Germany: Springer International Publishing, May 30, 2018, pp. 41–57. DOI: 10.1007/978-3-319-92612-4_3.

[50] Lauren Pick, Ankush Desai, and Aarti Gupta. "Psym: Efficient Symbolic Exploration of Distributed Systems". In: *Proceedings of the ACM on Programming Languages* 7.PLDI (June 6, 2023), pp. 660–685.

[51] Jan Pennekamp et al. "ConfMod: A Simple Modeling of Confidentiality Requirements for Inter-Organizational Data Sharing". In: *NOMS 2025-2025 IEEE Network Operations and Management Symposium*. IEEE, May 12, 2025, pp. 1–6. DOI: 10.1109/noms57970.2025.11073722.

[52] Marieke Huisman and Cristina Seceleanu. "ISoLA (1) - Verification and Validation of Concurrent and Distributed Systems (Track Summary)". In: Germany: Springer International Publishing, Oct. 29, 2020, pp. 421–425. DOI: 10.1007/978-3-030-61362-4_24.

[53] Oleg Savenko, Anatoliy Sachenko, Sergii Lysenko, George Markowsky, and Nadiia Vasylkiv. "BOTNET DETECTION APPROACH BASED ON THE DISTRIBUTED SYSTEMS". In: *International Journal of Computing* (June 14, 2020), pp. 190–198. DOI: 10.47839/ijc.19.2.1761.

[54] R Chandrasekar, RK Suresh, and SG 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*. IEEE. 2006, pp. 628–629.

[55] Peter Foytik and Sachin Shetty. *Blockchain for Distributed Systems Security - Blockchain Evaluation Platform*. Wiley, Mar. 15, 2019. DOI: 10.1002/9781119519621.ch13.

[56] Ratan K. Ghosh and Hiranmay Ghosh. *Distributed Systems*. Wiley, Feb. 8, 2023. DOI: 10.1002/9781119825968.

[57] Andriy Luntovskyy, Bohdan Shubyn, Taras Maksymuk, and Mykhailo Klymash. "Highly-Distributed Systems: What is Inside?" In: *2020 IEEE International Conference on Problems of*

Infocommunications. Science and Technology (PIC S&T). IEEE, Oct. 6, 2020, pp. 207–211. DOI: 10.1109/picst51311.2020.9467890.

[58] Denis Ullmann, Shideh Rezaeifar, Olga Taran, Taras Holotyak, Brandon Panos, and Slava Voloshynovskiy. "Information Bottleneck Classification in Extremely Distributed Systems." In: *Entropy (Basel, Switzerland)* 22.11 (Oct. 30, 2020), pp. 1237–. DOI: 10.3390/e22111237.

[59] Frank Kargl, Benjamin Erb, and Christoph Bösch. "Defining Privacy". In: Springer International Publishing, July 23, 2022, pp. 461–463. DOI: 10.1007/978-3-030-98546-2_30.

[60] "Proceedings of The 1st International Symposium on Parallel Computing and Distributed Systems". In: *2024 International Symposium on Parallel Computing and Distributed Systems (PCDS)*. IEEE, Sept. 21, 2024, pp. 1–1. DOI: 10.1109/pcds61776.2024.10743673.

[61] Dominic Karnehm, Sebastian Pohlmann, Andreas Wiedenmann, Manuel Kuder, and Antje Neve. "Introduction of a Cloud Computing Architecture for the Condition Monitoring of a Reconfigurable Battery System for Electric Vehicles". In: *2023 6th Conference on Cloud and Internet of Things (CIoT)*. IEEE, Mar. 20, 2023, pp. 29–33. DOI: 10.1109/ciot57267.2023.10084883.

[62] Nihar Vipulkumar Patel. "Adaptive Experimentation in Marketplaces: Balancing Exploration and Revenue Optimization". In: *Frontiers in Health Informatics* 11 (2022), pp. 760–786.

[63] Ilir Murturi and Schahram Dustdar. "DECENT: A Decentralized Configurator for Controlling Elasticity in Dynamic Edge Networks". In: *ACM Transactions on Internet Technology* 22.3 (Aug. 23, 2022), pp. 1–21. DOI: 10.1145/3530692.

[64] Malte Josten and Torben 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)*. IEEE, Mar. 17, 2025, pp. 606–607. DOI: 10.1109/percomworkshops65533.2025.00138.

[65] Reem M. Albarrak and Daniel A. Menasce. "Trust But Verify: A Framework for the Trustworthiness of Distributed Systems". In: *IEEE Transactions on Dependable and Secure Computing* 19.3 (May 1, 2022), pp. 2105–2121. DOI: 10.1109/tdsc.2020.3048301.

[66] Jarosław Flak, Tomasz Skowron, Rafał Cupek, Marcin Fojcik, Dariusz Caban, and Adam Domański. "ZigBee Network for AGV Communication in Industrial Environments". In: *2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, Oct. 9, 2023, pp. 1–9. DOI: 10.1109/dsaa60987.2023.10302633.

[67] Eileen Becks, Peter Zdankin, Viktor Matkovic, and Torben Weis. "Complexity of Smart Home Setups: A Qualitative User Study on Smart Home Assistance and Implications on Technical Requirements". In: *Technologies* 11.1 (Jan. 3, 2023), pp. 9–9. DOI: 10.3390/technologies11010009.

[68] Mitja Jančič, Jure Slak, and Gregor Kosec. "p-refined RBF-FD solution of a Poisson problem". In: *2021 6th International Conference on Smart and Sustainable Technologies (SpliTech)*. IEEE, Sept. 8, 2021, pp. 1–6. DOI: 10.23919/splitech52315.2021.9566401.

[69] Arturo Bretas, Dexin Wang, Orestis Vasios, and James Ogle. "Bi-Level Linear Programming Model for Automatic Load Shedding: A Distributed Wide-Area Measurement System-based Solution". In: *2023 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*. IEEE, Jan. 16, 2023, pp. 1–5. DOI: 10.1109/isgt51731.2023.10066344.

[70] Rahul Malik, Chandrasekar Ramachandran, Indranil Gupta, and Klara Nahrstedt. "Samera: a scalable and memory-efficient feature extraction algorithm for short 3D video segments." In: *IMMERSCOM*. 2009, p. 18.

[71] Christine Holland, Quan Nguyen, and Jeremy Twitchell. "The Economics of School Bus Fleet Electrification with Battery Storage". In: *2024 IEEE Electrical Energy Storage Application and Technologies Conference (EESAT)*. IEEE, Jan. 29, 2024, pp. 1–6. DOI: 10.1109/eesat59125.2024.10471206.

[72] Gerard O'Regan. "Cloud Computing and Distributed Systems". In: Springer International Publishing, Dec. 9, 2018, pp. 69–73. DOI: 10.1007/978-3-030-02619-6_15.

[73] Shuai Wang, Shenghao Liu, Xiaoxuan Fan, Haijun Wang, Yinxin Zhou, Hanjun Gao, Hongwei Lu, and Xianjun Deng. "VG-Net: Sensor Time Series Anomaly Detection with Joint Variational Autoencoder and Graph Neural Network". In: *2024 IEEE Smart World Congress (SWC)*. IEEE, Dec. 2, 2024, pp. 456–463. DOI: 10.1109/swc62898.2024.00094.