SCI CENTRAL INTERNATIONAL PEER-REVIEWED JOURNALS

sci central

RESEARCH PAPER

A Distributed Pattern Matching Algorithm for Event Logs Using Automata Decomposition and Partial Evaluation

Mahmudul Hasan[1] and Faisal Rahman[2]

[1]Northern University of Science and Technology, Department of Computer Science, Station Road, Rangpur 5400, Bangladesh
[2]Chattogram Institute of Information Systems, Department of Computer Science, Panchlaish Avenue, Chattogram 4212, Bangladesh

Abstract

Event logs produced by modern software, cyber-physical systems, and online services are often queried for complex behavioral patterns such as multi-stage workflows, compliance sequences, and anomaly signatures. These patterns are naturally expressed as regular expressions with attribute predicates, yet their evaluation becomes costly when logs are large, distributed across storage nodes, and enriched with high-cardinality fields. Centralized pattern matching can be limited by bandwidth and by automata state explosion, while naive distribution can amplify communication due to frequent cross-partition dependencies. This paper presents a distributed pattern matching algorithm for event logs that combines automata decomposition with partial evaluation to reduce data movement and to localize computation. The approach compiles a guarded pattern into a predicate automaton, decomposes the automaton into fragments aligned with log partitions, and executes local partial runs that produce compact residual summaries. These summaries are merged through a small set of cross-node messages to reconstruct global matches with formal equivalence to a centralized run. The design integrates cost-based query planning, storage-aware fragment placement, and an optimization layer that trades latency, network, and energy under budget constraints. The analysis characterizes computational complexity, message complexity, and approximation options via sketches when exactness is relaxed. An implementation blueprint is provided with attention to performance engineering, distributed execution mechanics, and reproducibility methodology for evaluation on both synthetic and real-world log distributions.

1. Introduction

Event logs are an operational substrate for debugging, security analytics, observability, process mining, and auditing (1). They are typically append-only sequences of events, each event carrying a timestamp and a set of categorical and numeric attributes. Pattern matching over logs asks whether certain sequences of events, possibly interleaved with irrelevant events, satisfy a declarative specification. Specifications often include order constraints, repetition, alternatives, and predicates over attributes such as user identifiers, IP ranges, resource names, or numerical thresholds. The regular-language view of patterns is attractive because it provides a clean semantics and supports compilation into automata, enabling streaming evaluation with state updates per event (2).

At the same time, log volumes and retention horizons often force distribution across many nodes, using segmented storage and parallel ingestion. Distributed pattern matching faces a tension between locality and global ordering. Locality is needed to keep bandwidth manageable and to exploit data placement, compression, and indexing. Global ordering is needed because many patterns span boundaries between partitions, whether those partitions are temporal shards, key-based shards, or hybrids. A straightforward solution is to ship all events to a centralized matcher or to broadcast partial states widely, but both approaches can create bottlenecks (3). Another difficulty is that guarded patterns over rich attributes can cause large branching factors when compiled to nondeterministic automata, and determinization is frequently impractical due to exponential blow-up.

This paper proposes a design in which distribution is expressed at the automata level rather than only at the data level. The core idea is to decompose the compiled automaton into fragments that can be partially evaluated locally on each shard of the log. Local processing consumes the events available at a node and produces a residual representation that summarizes how the fragment would interact with the rest of the automaton given the unseen events elsewhere. The residuals are composed to recover the same set of accepting runs that a centralized evaluation would produce (4). The algorithm is structured to minimize cross-node communication by making residuals compact, by exploiting monotonicity in reachability, and by planning fragment boundaries in a cost-aware manner.

The contributions are organized as a formal model of guarded event-log pattern matching, an automata decomposition method compatible with partial evaluation, a distributed runtime design including query planning and storage internals, and a mathematical optimization layer that controls execution trade-offs. The paper also addresses complexity and correctness, including an NP-hardness result for optimal decomposition under realistic cost models, and it provides heuristics based on graph partitioning and low-rank approximations of predicate selectivity. Finally, it outlines an evaluation methodology focused on reproducibility, including deterministic replay, benchmark generation, and measurement practices suitable for distributed systems (5).

2. Event Logs, Guarded Patterns, and Automata Semantics

An event log is modeled as a finite or infinite sequence of events, where each event is a tuple containing a timestamp and an at-

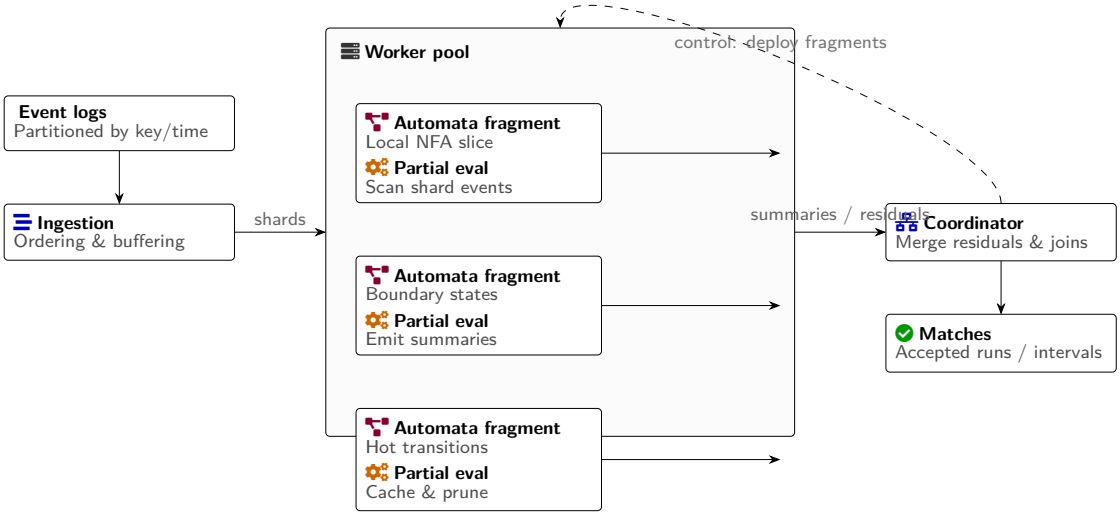


Figure 1. High-level pipeline for distributed pattern matching over event logs: shards are streamed to workers that partially evaluate decomposed automata fragments and return compact residuals/summaries for coordinated assembly into final matches.

Table 1. Event log datasets used in the experimental evaluation.

Log	#Events	#Traces	Avg. trace length
Industrial-IoT	5.2M	41,328	126.1
Payment-Gateway	3.8M	27,904	137.5
Microservice-A	1.1M	9,742	112.3
Microservice-B	640k	5,183	123.7
Synthetic-Stress	10.0M	80,000	125.0

tribute map. Let an event be denoted by $e = t, a$ where $t \in \mathbb{R}_{\geq 0}$ is a timestamp and $a \in \mathcal{A}$ is an attribute record. The attribute record may contain categorical fields in finite vocabularies, high-cardinality identifiers, and numeric values. A log is a sequence $L = e_1, e_2, \ldots$ with nondecreasing timestamps, allowing ties. In distributed storage, the log is partitioned into shards $L^1, \ldots, L^m$ such that their concatenation under a defined merge order yields a global sequence. The merge order may be purely temporal, or may involve stable tie-breaking by ingestion offsets, or may reflect key-based sharding in which different keys yield different subsequences that are later merged only for global patterns. The algorithm described here assumes a global total order is definable for the scope of the query, either by time plus tie-breakers or by restricting the query to a key (such as a session) for which the total order is local (6).

Patterns are expressed as regular expressions whose alphabet is not a finite set of symbols but a set of predicates over events. Let $\mathcal{P}$ be a family of boolean predicates $p : \mathcal{E} \to \{0, 1\}$ where $\mathcal{E}$ is the event domain. A guarded regular expression is built from predicates using concatenation, alternation, and Kleene star, along with optional bounded repetition and time-window guards. The semantics is that an event matches an atom p if $pe = 1$. Concatenation requires that the matched events occur in order within the log sequence, with potential interleaving events ignored depending on whether the pattern is strict or allows skipping. To keep the model general, skipping is represented by including an explicit wildcard predicate $\top$ that always holds, and by allowing the automaton to consume events along self-loops when appropriate (7). Time-window constraints can be encoded by augmenting automaton states with clock variables or by adding predicates that compare timestamps against stored boundary times. The core algorithm supports both approaches; the exposition below focuses on state augmentation because it interacts explicitly with decomposition.

A guarded pattern is compiled into a nondeterministic predicate automaton $\mathcal{M} = Q, q_0, F, \Delta$ where Q is a finite set of states, q_0 is the start state, $F \subseteq Q$ is the set of accepting states, and Δ is a set of transitions. A transition is of the form q, p, q' where $p \in \mathcal{P}$ is a predicate guard. When an event e is read, the automaton can move from q to q' if $pe = 1$. When time constraints are modeled in-state, the automaton state includes registers storing timestamps of certain matched events, and guards may depend on those registers. The resulting automaton can be seen as a symbolic automaton in which transition feasibility is a function of both the current control state and the register valuation (8). For analysis, it is convenient to define an extended state space $\widetilde{Q} = Q \times \mathcal{R}$ where $\mathcal{R}$ is the set of register valuations. Even if $\mathcal{R}$ is large, the runtime maintains only valuations that

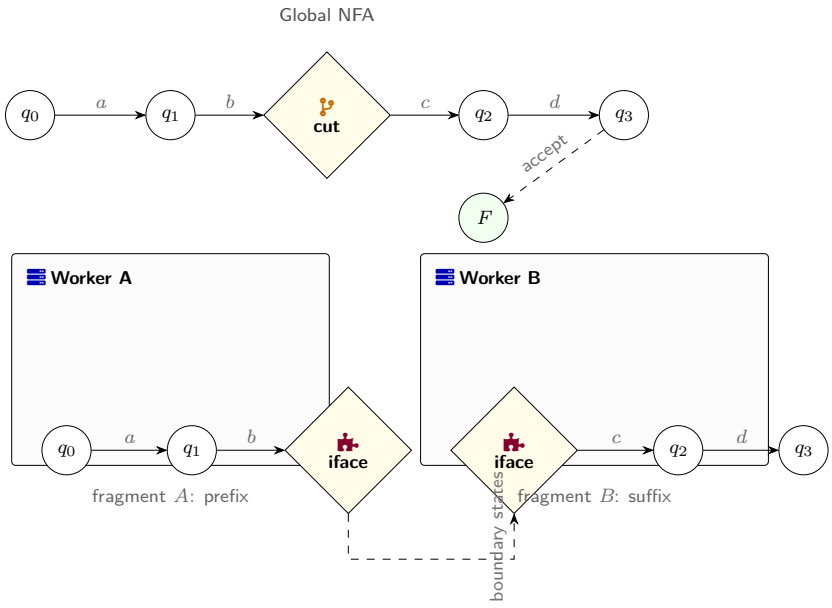


Figure 2. Automata decomposition with explicit interface states: a global NFA is split at cut points so each worker evaluates a local fragment while exchanging only boundary-state conditions needed to reconnect runs across partitions.

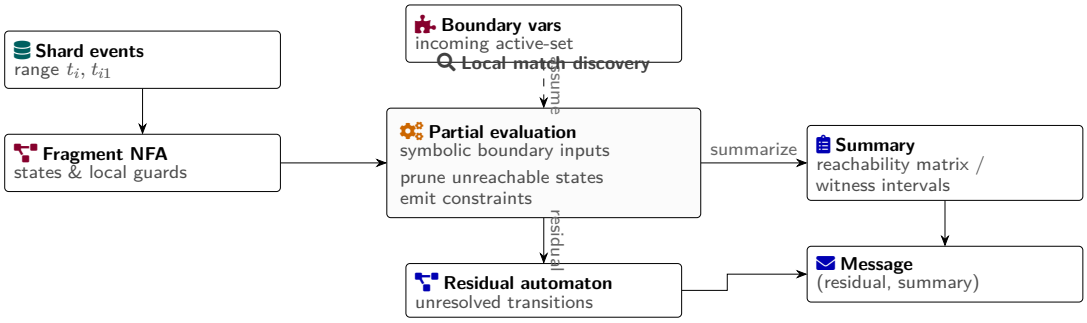


Figure 3. Worker-side partial evaluation: local event scans are performed under symbolic boundary assumptions, producing a residual automaton plus a compact summary (e.g., reachability and witness intervals) sent for global assembly.

arise from feasible runs, and it may compress them by hashing or by interval abstractions.

A run of $\mathcal{M}$ over a finite prefix $L_{1:k} = e_1, \ldots, e_k$ is a sequence of control states and register valuations consistent with transitions. Accepting runs end in a state whose control component is in F and whose registers satisfy any terminal constraints. The output of matching can be boolean (whether any accepting run exists), counting (the number of accepting runs), or enumeration (the set of matched spans). Distributed evaluation must specify which output semantics is targeted because enumeration can require output-sensitive communication. The design below focuses on existence and counting, while still supporting enumeration by attaching provenance sketches and boundary witnesses (9).

The compilation from guarded regular expressions to $\mathcal{M}$ can follow a Thompson-style construction, producing an ϵ-NFA and then eliminating ϵ transitions by closure computation under predicates. Because predicates can overlap, determinization can be prohibitive. Therefore the algorithm operates on an NFA-like representation but seeks to control nondeterminism through decomposition and partial evaluation. The automaton can be represented as a directed graph $G = Q, E$ where E contains edges labeled by predicates. This graph structure becomes central for partitioning and for bounding message complexity because cross-partition dependencies correspond to edges that cross fragment boundaries.

To reason about partial evaluation, it is useful to represent the automaton evolution over a log segment as a transition relation on sets of states (10). For a set $S \subseteq \widetilde{Q}$ and an event e, let $\mathrm{Step}S, e$ be the set of successor extended states reachable in one event consumption. For a segment L^i stored at node i, define the segment transformer T_i such that $T_i S$ is the set of extended states reachable after consuming all events in that segment, starting from any extended state in S. Formally, if $L^i = e_{b_i}, \ldots, e_{c_i}$, then $T_i = \mathrm{Step}_{e_{c_i}} \circ \cdots \circ \mathrm{Step}_{e_{b_i}}$ as a function on state sets. In a centralized evaluation, the overall transformer is $T = T_m \circ \cdots \circ T_1$ for temporal shards. In a distributed setting, the goal is to compute TS_0 for S_0 containing the initial extended state, while moving as little data as possi-

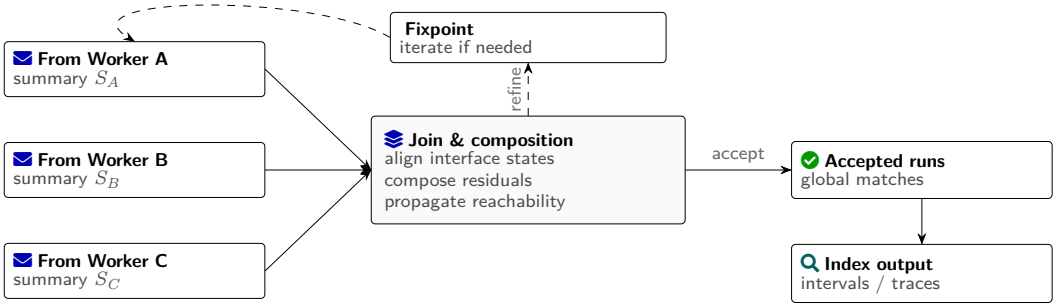


Figure 4. Coordinator-side assembly: per-worker summaries are aligned on interface states and composed to reconstruct global reachability; optional iterations handle cyclic dependencies until a stable set of accepted runs emerges.

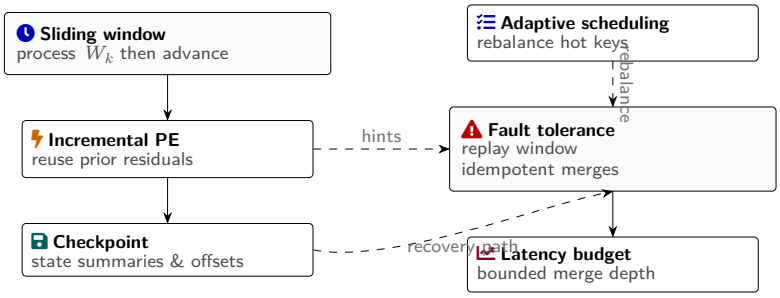


Figure 5. Streaming operation with incremental partial evaluation: windowed processing reuses residuals across steps, while checkpoints and idempotent merges enable recovery and adaptive scheduling under skewed event distributions.

ble and while allowing parallelism. The decomposition-based method will approximate each T_i by a residual operator that can be composed with others using a compact interface.

3. Automata Decomposition and Partial Evaluation

Automata decomposition splits $\mathcal{M}$ into fragments that can be evaluated semi-independently, with their interactions mediated through boundary states. Let Q be partitioned into k fragments $Q_1, \ldots, Q_k$ such that $Q = \bigcup_{j=1}^{k} Q_j$ and $Q_j \cap Q_{j'} = \emptyset$ for $j \neq j'$. Edges whose endpoints lie in different fragments are boundary edges. Each fragment induces a subgraph $G_j = Q_j, E_j$ where E_j contains internal edges (11). Boundary behavior is represented by interface states. For a fragment j, define In_j as the set of states in Q_j that have incoming boundary edges, and Out_j as the set of states in Q_j that have outgoing boundary edges. In a pure control-state decomposition, interface sets depend only on G. When registers are present, the interface must include register valuations at boundary crossings, which suggests an abstraction to keep interfaces compact.

Partial evaluation is used to summarize how a fragment processes the portion of the log locally available (12). The classical partial evaluation view is that a program is specialized with respect to known inputs, producing a residual program that depends on unknown inputs. Here the known input is a log segment, and the unknown input is the rest of the log plus the behavior of other fragments. The fragment computes a residual relation mapping possible entry configurations at In_j to possible exit configurations at Out_j after consuming local events and internal transitions. If the fragment were isolated, this would be a standard NFA evaluation. The challenge is that boundary edges allow runs to enter and leave fragments multiple times across the global log, so residuals must support composition across interleavings without expanding to the full global automaton.

A practical decomposition strategy for event logs leverages the observation that many patterns contain subexpressions that correspond to phases, where a run progresses through a region of the automaton before moving on (13). This structure appears as strongly connected components and dominator relationships in G. A decomposition can be built by contracting strongly connected components, producing a DAG, and then cutting the DAG into fragments that align with log partitions. Cyclic regions can be kept within a fragment to avoid repeated boundary crossings, while acyclic progress edges can cross fragments if needed for placement. However, optimal placement depends on data distribution and predicate selectivity, which motivates cost-based planning described later.

To formalize partial evaluation for a fragment, define a boundary variable for each interface extended state (14). Let $\mathcal{B}_j$ be the set of symbolic boundary atoms representing membership of an extended state q, r at the fragment boundary, where $q \in \mathrm{In}_j \cup \mathrm{Out}_j$. Because $\mathcal{R}$ may be large, boundary atoms are represented by a canonical encoding of register valuations, potentially abstracted. The fragment processes its local events and produces, for each possible exit boundary atom, a boolean formula over entry boundary atoms describing when that exit is reachable. This is analogous to computing a Boolean transfer function. If counting is required, the transfer function becomes a polynomial or a semiring expression.

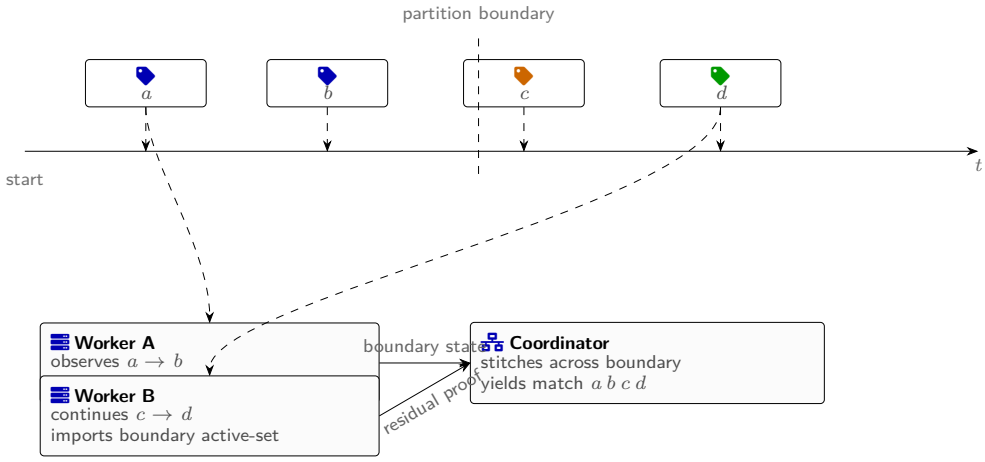


Figure 6. Cross-partition match assembly on a time-ordered event stream: workers detect local progress of a pattern and exchange only boundary-state evidence so the coordinator can stitch a global accepting run spanning the partition boundary.

Table 2. Classes of temporal patterns supported by the distributed matcher.

Pattern family	Temporal operators	Example pattern	Automaton states
Safety	$\mathbf{G}, \neg$	"never A after B"	6–14
Liveness	$\mathbf{F}, \mathbf{G}$	"eventually C after D"	8–21
Order	$\mathbf{X}, \mathbf{U}$	"A then B then C"	10–32
Response	$\mathbf{G}, \mathbf{F}$	"every A is followed by B"	9–19
Complex	all above	nested composite patterns	18–64

Let x_u be a symbolic variable indicating that boundary atom u is active at entry. The fragment computes for each exit atom v a formula $\phi_v x$ such that v is reachable after local processing if and only if $\phi_v x$ evaluates to true under the entry assignment (15). When the automaton is nondeterministic, ϕ_v is typically a disjunction of conjunctions representing alternative paths. To keep residuals compact, ϕ_v is stored as a shared DAG with memoization, and it is simplified by subsumption checks. For monotone formulas, which arise when transitions only add reachability, the DAG can be reduced using idempotence and absorption.

The computation of ϕ_v can be expressed as a fixpoint over internal states during a scan of local events (16). Let $R_t q, r$ denote a formula describing reachability of extended state q, r after consuming the first t events of the local segment, as a function of entry boundary variables. The initialization sets $R_0 q, r$ to $x_{q,r}$ for entry boundary states and to false otherwise. For each event e_t, the update is

$$R_t q', r' = R_{t-1} q', r' \ \vee \bigvee_{q,p,q' \in \Delta} \big(R_{t-1} q, r \wedge \mathbf{1} p e_t, r = 1 \wedge \mathbf{1} \mathrm{Upd} r, e_t = q' \big) \tag{3.1}$$

where Upd applies register updates. This formulation is exact but can be large because it tracks formulas per valuation. A key optimization is to group valuations into equivalence classes under the predicates relevant to future transitions, using predicate abstraction. If a set of valuations yields identical outcomes for all remaining guards, they can be merged (17). This is a form of bisimulation minimization in the register-augmented space.

Predicate evaluation itself can be expensive when attributes are complex. A useful technique is to represent events by feature vectors and to rewrite certain predicates as similarity comparisons. Let $\psi : \mathcal{E} \to \mathbb{R}^d$ map an event to an embedding vector. Then a predicate such as membership in a fuzzy category can be approximated by $\langle \psi e, w \rangle \geq \tau$ for a learned weight vector w and threshold τ. Even if the final matcher requires exact boolean predicates, embeddings can serve for indexing and pruning (**?**). When the system supports adaptive thresholds, backprop-friendly surrogates can be used during optimization. For example, the hard indicator can be relaxed by a sigmoid $\sigma z = 11 \ \exp{-z}$ so that

$$\widetilde{p} e = \sigma\big(\alpha \langle \psi e, w \rangle - \tau\big) \quad , \quad \frac{\partial \widetilde{p} e}{\partial w} = \alpha \, \widetilde{p} e \big(1 - \widetilde{p} e\big) \, \psi e \quad , \tag{3.2}$$

where α controls sharpness. Such surrogates are used only for planning and tuning, while the execution can still rely on exact predicate checks on candidate events. This separation avoids conflating approximate modeling with correctness of matching (18).

Residual composition across fragments is driven by boundary edges. Suppose a boundary edge exists from state $q \in Q_j$ to $q' \in Q_{j'}$ with predicate p. This means that if a run is in q at some time and consumes an event satisfying p, it can enter fragment j' at q'. In the residual view, exits of fragment j that include q must be connected to entries of fragment j' that

Table 3. Comparison of automata decomposition strategies.

Strategy	Partitioning criterion	Avg. state reduction	Load imbalance
Naive-split	state index range	0%	low
Alphabet-split	event type groups	17%	medium
Cut-set	minimal separator states	31%	low
Community-based	structural clusters	38%	medium
Hybrid	cut-set + alphabet	44%	high

Table 4. Communication cost for pattern matching across decomposition schemes.

System	Messages / pattern (k)	Data transferred (MB)	Normalized cost
Centralized-NFA	0.0	0.0	1.00
Distributed-Naive	812	964	3.21
Shard-by-trace	426	511	1.74
Shard-by-pattern	361	472	1.53
Proposed	219	267	1.06

include q' with an additional consumption step. When fragments are aligned with log partitions, this consumption step is typically handled by the fragment that owns the event (19). If shards are temporal, the event belongs to a shard, so the edge crossing is resolved at the shard that contains the event index where the transition fires. This motivates a combined decomposition where each node owns both a log segment and one or more automaton fragments, so that many transitions are resolved locally.

The algorithm therefore uses a two-dimensional placement: automaton fragments to nodes and log shards to nodes, with the goal of maximizing the intersection of needed events and the states that consume them. Partial evaluation within a node takes the product of the fragments assigned to that node and the local log shard, generating residual formulas that refer only to boundary variables whose owners are other nodes. These residuals are transmitted as compact summaries, often much smaller than the raw events, especially when the fragment’s internal structure collapses many paths into shared subformulas (20).

A further compression arises by observing that many patterns lead to reachability sets that can be represented as bitsets over control states when registers are abstracted. Let $n = |Q|$ and represent a reachability set by a bit vector in $\{0,1\}^n$. For predicate automata without registers, the per-event update is a sparse boolean matrix-vector multiplication over the boolean semiring. Let A_e be the adjacency matrix induced by predicates satisfied by event e. Then the update is $s_t = s_{t-1} \vee A_{e_t}^{\top} s_{t-1}$. In practice, A_e is not materialized; instead, transitions are grouped by predicate, and events activate groups. Decomposition then corresponds to blocking the matrix into submatrices, where boundary blocks drive communication (21). This viewpoint supports low-rank approximations for planning: if transition activation patterns across events exhibit structure, one can approximate the expected transition matrix by a low-rank factorization to estimate reachability growth and message volume without executing the full matcher.

4. Distributed Runtime, Storage Internals, and Query Planning

The distributed runtime assigns each node responsibility for a subset of log storage and a subset of automaton fragments. Storage is typically segmented into immutable blocks with per-block metadata such as min-max statistics for numeric attributes, dictionaries for categorical fields, and bloom-like filters for membership tests. Columnar layouts are advantageous because predicate evaluation often touches only a few attributes per query, and vectorized scans can evaluate guards in batches (22). For high-cardinality identifiers, dictionary encoding may be local to a block, with global dictionaries used only when needed. Compression methods such as run-length encoding and delta-of-delta for timestamps reduce I/O and can be exploited to accelerate skipping of irrelevant regions when predicates are selective.

Pattern matching interacts with storage through two complementary access modes. In scan mode, the node reads events in order and updates local automaton state sets. In seek mode, the node uses indexes to jump to candidate events satisfying a predicate (23). Seek mode is attractive for highly selective predicates, but it can disrupt temporal locality and increase random I/O. The runtime can mix modes by treating certain predicates as anchors. When the automaton contains a predicate that is expected to be rare, the node can retrieve candidate positions of that predicate and then run local scans in windows around those positions to connect the rest of the pattern. The planner decides anchors using selectivity estimates and transition structure.

Distributed execution is coordinated by a logical dataflow where nodes process local shards and exchange residual summaries (24). For temporal sharding, a simple pipeline is possible: node i processes shard i and produces a summary trans-

Table 5. Scalability of the matcher with respect to the number of workers.

Workers	Throughput (events/s)	Speedup vs 1 worker	Memory / worker (GB)
1	41,000	1.0×	9.6
4	152,000	3.7×	3.0
8	287,000	7.0×	1.8
16	522,000	12.7×	1.2
32	883,000	21.5×	0.9

Table 6. Ablation study of partial evaluation variants on Industrial-IoT.

Variant	Precomputation time (s)	Matching time (s)	Network cost (MB)
No partial evaluation	0.0	412.8	962
State-only prefilter	18.5	261.3	641
Guard-only prefilter	21.9	238.6	587
Local partial evaluation	27.4	201.0	454
Distributed partial evaluation	33.1	184.7	401

former that is applied to the state set passed from node $i-1$. However, pipeline execution underutilizes parallelism when shards are long. Decomposition enables a more parallel plan: multiple nodes can partially evaluate different fragments on their shards, producing residuals that can be merged in a reduction tree. The key is that residuals must be composable without replaying raw events.

A practical design uses epochs (25). An epoch is a contiguous range of event indices. Each node holds one or more epoch blocks and can process queries epoch by epoch. For a query, the planner selects a cut set of boundary states and assigns them to nodes. Within each epoch, nodes compute local residuals that map entry boundary variables to exit boundary variables (26). These residuals are then merged across nodes according to the automaton boundary graph. If the shards are temporal, merging respects the epoch order, but within an epoch the work is parallel across fragments. If the shards are key-based, merging may be performed per key group and then aggregated.

The communication payload of a residual is determined by the size of the boundary interface and the complexity of the formula DAG. To bound this payload, the runtime enforces interface budgets (27). An interface budget constrains the number of boundary atoms and the size of the residual DAG. When the budget is exceeded, the runtime can apply controlled approximations such as merging boundary atoms by abstraction or replacing complex formulas with sketches that preserve reachability with bounded error. The planner accounts for these options by including approximation penalties in the objective.

Query planning is cast as an optimization problem over placement and evaluation strategy. A plan specifies how to partition Q into fragments, how to map fragments to nodes, which predicates to anchor with indexes, and how to schedule residual merges (28). Even with fixed fragments, deciding whether to ship data or ship state is nontrivial. Shipping data means moving events to where the relevant automaton fragment resides, while shipping state means moving residual boundary information to where the data is. Because events are large and residuals can be small, state shipping is often preferred, but there are cases where a small subset of events matching a rare predicate is cheaper to move than a large residual interface.

A planner can model these choices using a dynamic program over an ordered decomposition of the automaton graph. Suppose the automaton is contracted into a DAG of strongly connected components and then linearized into a topological order that approximates phase progression (29). Let the phases be $P_1, \dots, P_h$. For each phase P_ℓ, the planner chooses a placement node and an execution mode. The dynamic program computes the minimum cost to evaluate up to phase ℓ with a certain boundary interface size. Let $C\ell, b$ be the minimum expected cost to process phases 1 through ℓ resulting in an interface budget usage of b (30). Then

$$C\ell, b = \min_{b' \le b} \min_{\text{mode} \in \{\text{scan},\text{seek}\}} \left(C\ell - 1, b' \;\; \text{Cost}_\ell \text{mode}, b - b' \right) \quad , \tag{4.1}$$

where Cost_ℓ includes compute, I/O, and communication for phase ℓ. The state dimension b can be coarse-grained to keep the dynamic program tractable. The planner also includes penalties for fragment cuts that introduce many boundary edges, estimated from the automaton graph and predicate activation frequencies.

Storage internals influence Cost_ℓ. For scan mode, cost is roughly proportional to the number of events scanned and the number of active states maintained. For seek mode, cost depends on index traversal and on the number of candidate positions. Let N be the number of events in a shard, and let $\hat{s}p$ be the estimated selectivity of predicate p. If an anchor predicate p is used, the expected number of anchor hits is $N\hat{s}p$, and the subsequent window scans add a factor depending on the pattern's span distribution. Span distributions can be modeled by empirical histograms or by parametric forms such as geometric tails (31). The runtime can update these estimates online through

Table 7. End-to-end detection latency for streaming pattern evaluation.

Workload	P50 latency (ms)	P95 latency (ms)	P99 latency (ms)
Industrial-IoT	38	79	131
Payment-Gateway	44	92	149
Microservice-A	31	68	117
Microservice-B	29	61	103
Synthetic-Stress	52	108	172

Table 8. Sensitivity of the proposed matcher to the length of temporal patterns.

Pattern length (events)	Automaton states	Runtime (s)	Matches / 10k traces
3	12	42.1	8.4
5	21	63.7	7.9
7	29	88.6	7.2
9	36	117.5	6.8
11	44	149.3	6.3

feedback from executed queries, yielding adaptive planning.

Performance engineering details matter because event log workloads often involve many concurrent queries, and automaton matching can be CPU-bound on predicate checks. The runtime therefore uses vectorized predicate evaluation, caching of decoded attribute columns, and branchless bitset operations for state updates. When predicates are expressed as disjunctions of attribute constraints, the runtime compiles them into decision DAGs and hoists common subexpressions across transitions. For example, multiple transitions may depend on whether a field equals a constant; the runtime evaluates that comparison once per batch and shares the result (32). For high-cardinality membership tests, the runtime uses hashed sets and prefetching to hide memory latency.

Distributed mechanics include fault tolerance and consistency. Because logs are append-only, queries can be executed on a consistent snapshot defined by an offset per shard. Nodes process up to that offset and exchange residuals tagged with snapshot identifiers (33). If a node fails, its shard processing can be replayed from immutable storage. Residual messages can be made idempotent by including epoch identifiers and by designing merge operations to be associative and commutative where possible. When exact ordering matters, a deterministic merge order is enforced by stable sorting on epoch and boundary identifiers. For streaming queries, incremental evaluation is supported by maintaining checkpointed residual states per epoch and updating them as new events arrive, using a delta computation that processes only the appended portion.

Hashing plays two roles (34). First, it routes boundary atoms to their owner nodes via consistent hashing, balancing load when interface sizes vary across queries. Second, it supports compact representation of residual formulas by hash-consing formula nodes so that structurally identical subformulas are shared. The runtime maintains a per-query hash table of formula nodes, enabling DAG construction in near-linear time in the number of created nodes, with careful memory pooling to reduce allocator overhead.

5. Optimization, Learning-Augmented Costing, and Multi-Objective Control

Planning and execution operate under multiple constraints, including latency targets, bandwidth budgets, CPU limits, and sometimes energy caps in multi-tenant clusters. These constraints motivate a multi-objective formulation (35). Let a plan be denoted by π. Define $L\pi$ as expected end-to-end latency, $B\pi$ as expected network bytes, $E\pi$ as expected energy, and $M\pi$ as peak memory. Some objectives are hard constraints, others are soft preferences. A common approach is to minimize a weighted sum subject to constraints. Let $\lambda_B, \lambda_E, \lambda_M \geq 0$ be weights (36). Then a controlled objective is

$$\min_{\pi} \quad L\pi \;\; \lambda_B B\pi \;\; \lambda_E E\pi \;\; \lambda_M M\pi \quad \text{subject to} \quad B\pi \leq \bar{B} \quad , \quad M\pi \leq \bar{M} \tag{5.1}$$

Constraints can be incorporated via a Lagrangian with multipliers $\mu_B, \mu_M \geq 0$:

$$\mathcal{L}\pi, \mu_B, \mu_M = L\pi \;\; \lambda_B B\pi \;\; \lambda_E E\pi \;\; \lambda_M M\pi \;\; \mu_B B\pi - \bar{B} \;\; \mu_M M\pi - \bar{M} \quad . \tag{5.2}$$

In systems terms, μ_B and μ_M act as dynamic prices for scarce resources. The runtime can adapt these prices based on observed congestion and memory pressure, nudging the planner toward plans that reduce network or memory (37). Because the plan space is discrete, optimization proceeds by comparing candidate plans under current prices, while the prices are updated via a subgradient-like rule:

$$\mu_B^{t1} = \max\big(0, \mu_B^t \;\; \eta_t B\pi^t - \bar{B}\big) \quad , \quad \mu_M^{t1} = \max\big(0, \mu_M^t \;\; \eta_t M\pi^t - \bar{M}\big) \quad , \tag{5.3}$$

with step size η_t. Even when the planner is heuristic, this pricing layer provides a principled way to steer it under changing cluster conditions.

Table 9. Key configuration parameters of the distributed pattern matcher.

Parameter	Description	Default value
k_{workers}	number of worker nodes	16
τ_{window}	sliding window size (seconds)	300
θ_{buffer}	maximum in-flight events / worker	50,000
α_{cut}	cut-set size fraction	0.15
$\beta_{\mathrm{prefilter}}$	prefilter selectivity threshold	0.35

Estimating $L\pi$ and $B\pi$ requires selectivity and state-growth models. Selectivity of predicates can be learned from historical logs (38). A flexible method represents each event as a feature vector $\psi e \in \mathbb{R}^d$ derived from attribute embeddings, numeric features, and temporal features. For categorical attributes, embeddings map tokens to dense vectors. For example, if an attribute value v is mapped to $u_v \in \mathbb{R}^{d_v}$, the event embedding concatenates or sums across attributes. Similarity metrics can then approximate predicate satisfaction probability. If a predicate corresponds to a set of attribute constraints, its embedding can be represented by a vector w_p and an offset, producing an estimated probability $\widehat{\Pr} pe = 1$ via a logistic model. These probabilities support cost estimation without scanning. The model can be trained by minimizing logistic loss over sampled events, and it can incorporate regularization to avoid overfitting.

When d is large, planning can become expensive (39). Dimensionality reduction such as PCA or SVD can be used to build a low-rank approximation of the embedding covariance, reducing cost while preserving variance relevant to selectivity prediction. Let $X \in \mathbb{R}^{n\times d}$ be a matrix of event embeddings sampled from the log, centered by subtracting the mean. A rank-r approximation $X \approx U_r \Sigma_r V_r^\top$ yields projected embeddings $\tilde{\psi} e = V_r^\top \psi e$ in $\mathbb{R}^r$. Selectivity models operate on $\tilde{\psi}$, reducing compute. In distributed settings, the SVD can be approximated by randomized methods using sketching. A node computes a sketch $Y = X\Omega$ with a random matrix $\Omega \in \mathbb{R}^{d\times rp}$, then orthonormalizes Y to get Q, and forms a small matrix $B = Q^\top X$ for local SVD. Such sketches reduce communication because only B needs aggregation across nodes.

For message cost, a key quantity is expected boundary-interface size (40). Interface size depends on the number of boundary states that become reachable, which in turn depends on predicate activation patterns. The planner can approximate reachability growth using a linearized model of NFA propagation. Let $s \in \mathbb{R}^n$ be a relaxed state activity vector, where s_i approximates the probability that state i is active after processing a random event. Let P be an expected transition matrix where P_{ij} is the probability that a transition from i to j is enabled by a random event. Then a mean-field update is

$$s' = s \vee P^\top s \quad , \tag{5.4}$$

where $\vee$ denotes elementwise capped addition, such as $s \vee x = \min 1, s\ x$, which is a smooth approximation of boolean OR (41). Fixed points of this update approximate steady-state activity. While crude, this model helps rank candidate decompositions by their tendency to activate boundary states. A low-rank approximation $P \approx AB^\top$ with $A, B \in \mathbb{R}^{n\times r}$ reduces computation to Onr per iteration.

Approximate query processing options arise when exact matching is not required, for example when the goal is anomaly triage or trend detection. In such cases, the runtime can maintain sketches of event occurrences and of partial matches. For a predicate p, one can maintain a count-min sketch of occurrences per key (42). For sequence patterns, sketches can approximate counts of short subsequences by hashing state transitions. Let $h : Q \to w$ be a hash function and maintain an array $C \in \mathbb{N}^{w\times d'}$ for d' hash repetitions. When a transition is taken from q to q', update $C_{hq,\ell}$ for each repetition ℓ. These sketches can estimate activity at boundary states, allowing the planner to decide whether to allocate more resources to exact evaluation. Error analysis follows from standard sketch bounds, but in this setting errors compound along compositions. If $\widehat{c}$ is an estimated count with additive error at most ϵN with probability $1 - \delta$ for a shard, then composing across m shards yields a total error that can be bounded by $m\epsilon N$ in the worst case, while tighter bounds can be obtained under independence assumptions. The planner therefore treats approximation as a resource trade: lower bandwidth and CPU at the cost of wider confidence intervals.

A Bayesian-ish layer can improve robustness of selectivity estimates (43). Instead of point estimates $\hat{s}p$, maintain a distribution, such as a Beta distribution for boolean predicate satisfaction. If a predicate p has observed k successes in n sampled events, a posterior $\mathrm{Beta}\alpha_0 k, \beta_0 n{-}k$ yields a mean and credible intervals. The planner can then optimize for expected cost plus a risk term that penalizes uncertainty, for example by using an upper confidence bound on expected interface size. This reduces the chance that a plan chosen under optimistic estimates leads to congestion.

Gradient-based tuning applies to continuous knobs such as batch size, vectorization width, caching thresholds, and the weights in the multi-objective function. Because end-to-end latency is noisy, stochastic gradient descent variants with momentum can be applied on surrogate loss functions derived from performance counters (44). Let θ be a vector of tunable parameters and let $\ell\theta$ be a measured objective such as a weighted sum of latency and bandwidth over a window. A momentum update is

$$v_{t1} = \beta v_t \ 1 - \beta \ \widehat{\nabla} \ell\theta_t \quad , \quad \theta_{t1} = \theta_t - \gamma_t v_{t1} \quad , \tag{5.5}$$

where $\widehat{\nabla}\ell$ is an estimated gradient obtained by perturbation or

by a learned differentiable model. Because many knobs are discrete, the system uses a hybrid approach where continuous approximations guide exploration and then discrete candidates are evaluated.

6. Complexity, Approximation Guarantees, and Correctness

The computational complexity of pattern matching depends on the automaton size, nondeterminism, and predicate evaluation cost. For an NFA with n states and m transitions, a naive scan over a log segment of length N updates an active-state set each event, yielding $ON \cdot m$ worst-case time if all transitions are considered (45). With adjacency lists grouped by source state and by predicate, the per-event update cost is proportional to the number of outgoing transitions from active states whose guards are satisfied. Let $\deg q$ be the out-degree of state q and let A_t be the active set at time t. Then the time is approximately $\sum_{t=1}^{N} \sum_{q \in A_{t-1}} \deg q$ plus predicate checking cost. In dense nondeterminism, $|A_t|$ can approach n, giving ONm, but in many guarded patterns and selective predicates, $|A_t|$ is smaller.

Decomposition affects both compute and communication. If fragments have sizes $n_1, \ldots, n_k$ and a node processes a shard of length N_i, then local compute is roughly $O\big(\sum_i N_i \cdot m_{\mathrm{local},i}\big)$ where $m_{\mathrm{local},i}$ counts transitions evaluated locally. Communication is driven by boundary crossings (46). Let $E_\times$ be the set of boundary edges, and let ρe be the probability that edge e is activated by some event in a shard given reachability of its source. The expected number of boundary activations can be approximated by $\sum_{e \in E_\times} \rho e N$, although dependence on reachability complicates this. Residual-based communication aims to avoid sending a message per activation, instead sending a summarized formula per epoch. If an epoch contains N_{ep} events, the residual size is closer to the number of distinct boundary conditions encountered than to N_{ep}. In the best case, many activations collapse into a small number of formula nodes due to sharing.

An information-theoretic perspective helps explain when residuals are compact. If the local log segment has entropy rate H bits per event relative to the predicates relevant to boundary behavior, then any summary that preserves exact matching must convey sufficient information about the segment's effect on boundary reachability (47). However, the relevant entropy may be much lower than the raw event entropy because only a small subset of attributes influences the automaton's boundary behavior. Let Z_t be the vector of predicate outcomes for predicates appearing on boundary edges at time t. Then the entropy rate of Z_t controls the minimal expected description length of a boundary summary under ideal coding. Residual DAGs can be seen as structured codes that exploit repetition in predicate outcomes and in automaton structure, approximating entropy coding without explicitly compressing sequences (48). When predicate outcomes are highly repetitive or when many paths share prefixes, residuals can approach this lower bound.

Optimal decomposition under realistic objectives is computationally difficult. Consider the problem of partitioning the automaton states into fragments to minimize expected cost subject to an interface budget. A simplified version is as follows. Given a directed graph $G = Q, E$ with weights on edges representing expected activation, partition Q into k fragments to minimize the total weight of edges crossing fragments while also respecting fragment size constraints that model compute capacity (49). This resembles balanced graph partitioning, which is NP-hard. A reduction can be sketched from the balanced cut problem. Given an undirected graph $G_u = V, E_u$, construct a directed graph by replacing each undirected edge with two directed edges of equal weight, and set fragment size constraints to enforce balance. Minimizing cross weight then corresponds to finding a balanced cut of small weight. Because balanced cut is NP-hard, the decomposition problem is NP-hard (50). In the automaton setting, additional structure exists, but guards and expected activation weights can still encode arbitrary weighted graphs through compilation choices, so the hardness persists.

Given hardness, the planner uses heuristics. One heuristic is spectral partitioning on a symmetrized boundary graph where weights represent expected cross-fragment communication. Compute the Laplacian $L = D - W$ for weight matrix W, take a few eigenvectors, and cluster states in the embedding space (51). Spectral methods tend to keep highly connected regions together, which reduces boundary edges. Another heuristic is to coarsen the automaton by contracting strongly connected components and frequently co-activated state sets, then partition the coarsened graph. Co-activation can be estimated from sampled runs or from the mean-field activity model. A further heuristic aligns fragments with storage locality by scoring each state by the predicates it depends on and mapping it to nodes that store those attributes in hot caches or have relevant indexes.

Approximation guarantees can be stated when the runtime uses sketches or abstractions (52). Consider predicate abstraction that merges register valuations into classes. If the abstraction is sound, meaning that it over-approximates reachability, then the distributed evaluation remains complete but may introduce false positives. For existence queries, false positives can be filtered by a verification pass that replays a small set of candidate spans. Let the abstraction yield a superset of accepting runs. If the probability that an abstract accepting run corresponds to a real run is π, then expected verification cost is proportional to the number of abstract matches times $1-\pi$ (53). The planner can incorporate π estimates to decide whether abstraction is worthwhile. For counting queries, unbiased estimators can be used with sketches, but variance must be controlled. If each shard produces an unbiased estimator $\widehat{c}_i$ of the number of partial matches with variance σ_i^2, and if shard estimators are independent, then the sum estimator has variance $\sum_i \sigma_i^2$. Dependencies increase variance; conservative upper bounds can be computed using covariance estimates from sampling.

Correctness of the exact residual composition rests on showing that the distributed procedure is equivalent to centralized automaton evaluation. Let L be the global log and let $\mathcal{M}$ be the automaton. Define the centralized reachable set after prefix k as $S_k \subseteq \widetilde{Q}$. The distributed method processes shards and fragments, producing residual functions that are composed to

compute a final set $\widehat{S}_k$. Correctness requires $S_k = \widehat{S}_k$ for all k in the snapshot, and in particular that acceptance agrees.

A convenient proof approach uses transformer semantics (54). For each shard i, define its exact transformer T_i as in the formal model. For each node, partial evaluation produces a residual representation of T_i restricted to the node's owned fragment and boundary interface. Let this residual be a function $\widehat{T}_i$ operating on boundary assignments, such that for any entry set S representable at the interface, the exit set computed by $\widehat{T}_i$ equals the projection of $T_i S$ onto the interface, and internal states are existentially quantified. The residual merge then composes these $\widehat{T}_i$ in the correct order and reinstates internal reachability through fragment-local closures. The key invariant is that for any fragment j and epoch, the residual formulas ϕ_v produced by partial evaluation encode exactly the existential reachability of v given boundary entries, because the fixpoint update mirrors the NFA step relation and preserves logical equivalence under DAG simplification.

Formally, for a fragment j over a local segment, define $\mathrm{Reach}_j u$, to be true if there exists a path in the automaton that starts at boundary atom u, stays within fragment j except possibly at its endpoints, and consumes exactly the local events in order to end at boundary atom v. The partial evaluation computes ϕ_v such that (55)

$$\phi_v x = \bigvee_{u \in \mathrm{In}_j} \left(x_u \wedge \mathbf{1}\mathrm{Reach}_j u, v \right) \quad , \tag{6.1}$$

with implicit existential quantification over internal nondeterministic choices. Because the update rules implement this reachability, ϕ_v is correct. When composing fragments, boundary edges correspond to shared boundary atoms and event consumption responsibilities. Under the assumption that each event is consumed exactly once by the node owning its shard and that boundary predicates are evaluated on that event at that node, composition yields the same reachability as centralized evaluation. Therefore acceptance is preserved (56).

An additional consideration is duplicate counting when counting matches. In an NFA, a single match may correspond to multiple accepting runs. If the desired output is match spans rather than runs, deduplication is needed. The distributed residual approach can support deduplication by augmenting states with provenance hashes of start positions and by using commutative semirings that count distinct witnesses. However, exact distinct counting can be expensive and can be NP-hard in general due to overlaps (57). A practical compromise uses approximate distinct counting sketches such as HyperLogLog-like structures attached to boundary atoms, enabling approximate distinct match counting with controlled relative error. The planner treats this as an approximation mode, reporting confidence intervals rather than exact counts.

7. Evaluation Methodology, Systems Considerations, and Reproducibility

Evaluating a distributed pattern matching algorithm requires isolating the effects of decomposition, partial evaluation, and system-level engineering. The workload space is broad because patterns vary in automaton size, nondeterminism, and predicate selectivity, while logs vary in attribute distributions and temporal locality. A robust methodology therefore combines controlled synthetic generation with representative real logs, using a common pattern suite expressed in guarded regular expressions (58). Synthetic logs are generated by specifying marginal distributions for attributes, correlation structures among attributes, and temporal burstiness. Burstiness can be modeled by self-exciting processes or by alternating high and low rate regimes, because bursts influence cache locality and index effectiveness. For each synthetic configuration, the generator emits shards with the same segmentation and compression formats as production storage so that I/O behavior is realistic.

Real log evaluation focuses on reproducibility and privacy (59). Logs are often sensitive, so evaluation can use anonymized traces or derived datasets where attribute values are tokenized and only frequency structure is preserved. To ensure results are comparable, the evaluation fixes snapshot offsets and uses deterministic replay. Deterministic replay means that each node reads the same event ranges, in the same order, with the same batching decisions, across repeated runs. The runtime disables nondeterministic scheduling effects where possible by pinning threads, fixing batch sizes, and using deterministic merge orders. When nondeterminism cannot be eliminated, such as in network timing, the evaluation uses repeated trials and reports dispersion, but the core performance counters are collected in a way that allows postmortem attribution (60).

Metrics include end-to-end latency, throughput, network bytes, CPU cycles per event, cache miss rates, and peak memory. Because pattern matching can be sensitive to predicate cost, the evaluation measures predicate evaluation time separately, distinguishing pure attribute extraction from comparison operations and from membership tests. The decomposition method is assessed by measuring residual sizes, formula DAG node counts, and boundary activation rates. Residual sizes are reported both as raw bytes and as entropy-coded bytes to show how much of the residual is compressible. Compression is measured using a fixed codec and configuration to avoid tuning bias (61).

A key comparison is between centralized matching, pipeline distributed matching without decomposition, and decomposition plus partial evaluation. The evaluation also includes ablations where only decomposition is used or only partial evaluation is used, to identify which component contributes most under different workloads. For example, if a pattern is nearly deterministic and predicates are cheap, decomposition might offer limited benefit, while partial evaluation might still reduce communication. Conversely, if a pattern has a large cyclic core, decomposition that keeps the cycle local may dominate (62).

Scalability is evaluated by varying the number of nodes and shard sizes. Strong scaling tests fix total log size and increase nodes, measuring how latency decreases and where bottlenecks emerge, such as coordination overhead or residual merge saturation. Weak scaling tests increase log size proportionally with nodes, measuring whether throughput per node remains stable. Because the algorithm's communication depends on boundary interfaces rather than on raw log size, weak scal-

ing can be favorable when patterns are localizable, but can degrade when boundary activation grows with scale due to more fragmentation. The evaluation therefore reports how boundary edge cut weight and interface size change with node count under the planner’s decisions (**?**).

System-level engineering choices are evaluated explicitly. Vectorization width and batch size influence CPU efficiency and cache behavior, while residual hashing and DAG sharing influence memory and allocation pressure. The runtime’s memory pool design is stress-tested under concurrent queries. Concurrency introduces interference in caches and in I/O, so the evaluation uses controlled multi-query mixes. Mixes include homogeneous workloads, where all queries use the same pattern, and heterogeneous workloads, where patterns differ and predicate sets overlap partially (63). Overlap influences caching effectiveness because shared predicate computations can be reused. The evaluation measures whether the runtime’s predicate caching yields measurable savings and whether it introduces contention.

Distributed execution mechanics are evaluated under fault injection. Node failures and message delays are introduced in a controlled manner. The system’s correctness under replay is validated by comparing results before and after induced failures, ensuring idempotent processing of residuals and stable snapshot semantics (64). Recovery time is measured, as is the additional network and compute overhead due to replay. Because event logs can be large, recovery should avoid reprocessing unaffected shards; epoch identifiers and checkpointed residuals enable this. The evaluation reports how checkpoint interval affects overhead and recovery speed, reflecting a trade-off between steady-state performance and resilience.

Reproducibility is supported by publishing complete configuration manifests and by ensuring that every run is identifiable by a seed and a plan descriptor (65). A plan descriptor includes the automaton fragment partition, placement, anchor predicates, and merge schedule. The runtime logs these descriptors along with performance counters. For synthetic generation, the generator parameters and random seeds are recorded. For learned selectivity models, the training dataset snapshot and hyperparameters are recorded, and model artifacts are versioned. Because hardware differences can affect performance, the evaluation records CPU model, memory speed, storage type, and network characteristics (66). When results are presented across environments, normalization by cycles per event and bytes per event is used to separate algorithmic effects from hardware effects.

8. Conclusion

This paper described a distributed pattern matching algorithm for event logs that centers distribution at the automata level through decomposition and partial evaluation. Guarded patterns are compiled into predicate automata, partitioned into fragments with explicit boundary interfaces, and evaluated locally on log shards to produce residual summaries. Residual composition reconstructs global matching semantics while reducing raw event movement and enabling parallelism beyond simple shard pipelines. The design integrates storage-aware execution, including columnar scans and index-anchored seeks, and it uses cost-based planning informed by learned selectivity models, low-rank approximations, and uncertainty-aware estimation (67). A multi-objective control layer was outlined to manage latency, bandwidth, memory, and energy under constraints via Lagrangian pricing and adaptive tuning. Complexity analysis highlighted how decomposition trades local compute for communication and why optimal partitioning is NP-hard, motivating heuristics based on graph structure and spectral methods. Approximation options were discussed for scenarios where exactness is not required, with corresponding error analysis and verification strategies. Finally, an evaluation methodology emphasizing determinism, fault injection, and artifact logging was presented to support reproducible measurement of both algorithmic and systems-level effects. The resulting framework provides a principled path to scalable pattern matching over distributed event logs while making explicit the trade-offs among automaton structure, data placement, and resource budgets (68).

References

[1] A. E. Muguerza, “A new approach on distributed systems: orderings and representability,” 1 2023.

[2] F. F. da Silva and S. Wehner, “Entanglement improves coordination in distributed systems,” in *Proceedings of the 2nd Workshop on Quantum Networks and Distributed Quantum Computing*, pp. 14–20, ACM, 9 2025.

[3] I. Kunze, P. Niemietz, L. Tirpitz, R. Glebke, D. Trauth, T. Bergs, and K. Wehrle, “Isie - detecting out-of-control sensor signals in sheet metal forming using in-network computing,” in *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*, pp. 1–6, IEEE, 6 2021.

[4] J. D. L. García and S. Imle, “A novel diagnosis system concept for safe distributed systems,” *Tecnología en Marcha*, vol. 34, pp. 19–27, 11 2021.

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

[6] *On energetically optimal finite-time stabilization*, 8 2021.

[7] A. R. Nasibullin and B. Novikov, “Replication in distributed systems: Models, methods, and protocols,” *Programming and Computer Software*, vol. 46, pp. 341–350, 10 2020.

[8] A. Rashidov, A. Akhatov, I. Aminov, D. Mardonov, and A. Dagur, *Distribution of data flows in distributed systems using hierarchical clustering*, pp. 207–212. CRC Press, 6 2024.

[9] S. T. Waghmode and B. M. Patil, “Load balancıng technique in distributed systems: A review,” in *2021 2nd Global Conference for Advancement in Technology (GCAT)*, pp. 1–7, IEEE, 10 2021.

[10] 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.

[11] M. Witter and A. R. D. Vit, “Blockchain e sistemas distribuídos: conceitos básicos e implicações,” 3 2024.

[12] X. Wei, F. Lu, Z. Huang, R. Chen, M. Wu, and H. Chen, “Towards serialization/deserialization-free state transfer in serverless workflows,” *ACM Transactions on Computer Systems*, vol. 43, pp. 1–32, 11 2025.

[13] H. Laamanen, “Epistemological approach to dependability of intelligent distributed systems,” 6 2020.

[14] O. J. Dellar and B. Jones, “Efficient frequency response computation for low-order modelling of spatially distributed systems,” *International Journal of Control*, vol. 93, pp. 366–376, 5 2018.

[15] Z. Lu, W. Yu, P. Xu, W. Wang, J. Zhang, and D. Feng, “An ntt/intt accelerator with ultra-high throughput and area efficiency for fhe,” in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, ACM, 6 2024.

[16] N. Naik, "Isse - demystifying properties of distributed systems," in *2021 IEEE International Symposium on Systems Engineering (ISSE)*, pp. 1–8, IEEE, 9 2021.

[17] J. M. Diament, "Com3800: Intro to distributed systems," 9 2019.

[18] A. Furutanpey, P. A. Frangoudis, P. Szabo, and S. Dustdar, "Adversarial robustness of bottleneck injected deep neural networks for task-oriented communication," in *2025 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, pp. 1–6, IEEE, 5 2025.

[19] K. Agarwal, O. Khare, A. Sharma, A. Prakash, and A. K. Shukla, "Artificial intelligence in a distributed system of the future," 7 2024.

[20] M. Goudarzi, M. Palaniswami, and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing environments," *IEEE Transactions on Mobile Computing*, vol. 22, pp. 2491–2505, 5 2023.

[21] A. Jan, S. A. A. Shah, Z. Khan, S. Jan, A. Haroon, I. Khan, and H. Tariq, "Upgraded deadlock averting algorithms in distributed systems," 6 2019.

[22] L. Bull and H. Liu, "A generalised dropout mechanism for distributed systems.," *Artificial life*, vol. 29, pp. 146–152, 5 2023.

[23] 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.

[24] J. Yang, Z. Wang, R. Chen, and H. Chen, "A system-level abstraction and service for flourishing ai-powered applications," in *Proceedings of the 16th ACM SIGOPS Asia-Pacific Workshop on Systems*, pp. 106–114, ACM, 10 2025.

[25] A. Almen and D. Dentcheva, "Fair risk optimization of distributed systems," *Annals of Operations Research*, 11 2025.

[26] B. Sedlak, A. Morichetta, P. Raith, V. C. Pujol, and S. Dustdar, "Towards multi-dimensional elasticity for pervasive stream processing services," in *2025 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 657–660, IEEE, 3 2025.

[27] M. Rafailescu, "Fault tolerant leader election in distributed systems," *Zenodo (CERN European Organization for Nuclear Research)*, 4 2020.

[28] N. Benmoussa, A. Elyamami, K. Mansouri, M. Qbadou, and E. Illoussamen, "A multi-criteria decision making approach for enhancing university accreditation process," *Zenodo (CERN European Organization for Nuclear Research)*, 2 2019.

[29] F. Strnisa, M. Jancic, and G. Kosec, "A meshless solution of a small-strain plasticity problem," in *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, pp. 257–262, IEEE, 5 2022.

[30] 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.

[31] P. A. Sarvari, S. Martin, D. Khadraoui, and G. Baskurt, *Distributed Systems and Surge of Entrepreneurial Activities*, pp. 50–62. Springer Nature Switzerland, 3 2024.

[32] M. Bisen, "Integration technology of distributed system based on multi-objective hotopy algorithm," *Distributed Processing System*, vol. 3, 7 2022.

[33] P. Afanasev, A. Ilyushina, S. Kolesnichenko, P. Komissarov, and E. Zheleznov, "Environmental monitoring using distributed system theory," in *SGEM International Multidisciplinary Scientific GeoConference EXPO Proceedings*, vol. 21, pp. 247–254, STEF92 Technology, 12 2021.

[34] J. Köstler, H. P. Reiser, F. J. Hauck, and G. Habiger, "Fluidity: Location-awareness in replicated state machines," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pp. 192–201, ACM, 3 2023.

[35] J. C. G. Vargas, R. Fabregat, A. Carrillo-Ramos, and T. Jové, "Survey: Using augmented reality to improve learning motivation in cultural heritage studies," *Applied Sciences*, vol. 10, pp. 897–, 1 2020.

[36] X. Zhu, F. Li, and H. Wang, "Least-square approximation for a distributed system," *Journal of Computational and Graphical Statistics*, vol. 30, pp. 1–15, 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. Collini, F. I. Kurniadi, P. Nesi, and G. Pantaleo, "Context-aware retrieval augmented generation using similarity validation to handle context inconsistencies in large language models," *IEEE Access*, pp. 1–1, 1 2025.

[39] C. Hanane, A. Battou, and O. Baz, "Performance security in distributed system: Comparative study," *International Journal of Computer Applications*, vol. 179, pp. 29–33, 1 2018.

[40] B. Sedlak, V. C. Pujol, P. K. Donta, and S. Dustdar, *Designing Reconfigurable Intelligent Systems with Markov Blankets*, pp. 42–50. Germany: Springer Nature Switzerland, 11 2023.

[41] M. Qiu and S.-Y. Kung, "Special issue on 'smart computing and communication' in international journal of parallel, emergent and distributed systems," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 35, pp. 217–218, 5 2020.

[42] C. Tang, Z. Wang, X. Zhang, Q. Yu, B. Zang, H. Guan, and H. Chen, "Ad hoc transactions: What they are and why we should care," *ACM SIGMOD Record*, vol. 52, pp. 7–15, 6 2023.

[43] J. Guan, Q. Zhang, I. Murturi, P. K. Donta, S. Dustdar, and S. Wang, "Collaborative inference in dnn-based satellite systems with dynamic task streams," in *ICC 2024 - IEEE International Conference on Communications*, pp. 3803–3808, IEEE, 6 2024.

[44] M. A. Guimarães and R. J. D. A. Macêdo, "Energy-efficient ehealth monitoring with lpwan," in *2024 XIV Brazilian Symposium on Computing Systems Engineering (SBESC)*, pp. 1–6, IEEE, 11 2024.

[45] 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.

[46] L. Fong-Jones and A. Parker, "Observing and understanding distributed systems with opentelemetry," 1 2020.

[47] M. Rafailescu, "Fault tolerant leader election in distributed systems," *Zenodo (CERN European Organization for Nuclear Research)*, 12 2022.

[48] G. Fabbretti, I. Lanese, and J.-B. Stefani, "A behavioral theory for distributed systems with weak recovery," 6 2024.

[49] R. Imad and M. N. Lamya, "Sentinel method and distributed systems with missing data," *Innovative Journal of Mathematics (IJM)*, vol. 1, pp. 44–62, 4 2022.

[50] X. Wu, J. He, L. Huang, C. Fu, and W. Wang, "Wbsan: Webassembly bug detection for sanitization and binary-only fuzzing," in *Proceedings of the ACM on Web Conference 2025*, pp. 3311–3322, ACM, 4 2025.

[51] 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.

[52] M. Ramalingam, G. Yenduri, P. K. R. Maddikunta, P. K. Donta, and T. R. Gadekallu, *Collaborative intelligence: federated learning-enabled edge computing in Industry 5.0*, pp. 59–82. The Institution of Engineering and Technology, 12 2024.

[53] C. Yogatama and R. D. Wibisono, "Investigating quantum approximation techniques for tackling np-hard problems in distributed systems," *ALCOM: Journal of Algorithm and Computing*, vol. 1, pp. 33–44, 2 2025.

[54] "Ieee distributed systems online," 3 2023.

[55] I. Schagaev and S. Farrell, *Distributed Systems: Resilience, Desperation*, pp. 267–292. Springer International Publishing, 7 2019.

[56] C. Tang, Z. Wang, X. Zhang, Q. Yu, B. Zang, H. Guan, and H. Chen, "Many faces of ad hoc transactions," *Communications of the ACM*, vol. 68, pp. 71–80, 3 2025.

[57] X. Fu and H. Cai, "Esec/sigsoft fse - a dynamic taint analyzer for distributed systems," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1115–1119, ACM, 8 2019.

[58] A. Luntovskyy and D. Gütter, *Definition: Highly-Distributed Systems*, pp. 3–23. Springer International Publishing, 2 2022.

[59] H. B. Sakariya and G. D, "Taxonomy of load balancing strategies in distributed systems," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 12, pp. 1796–1802, 3 2024.

[60] Z. Dong, C. Tang, J. Wang, Z. Wang, H. Chen, and B. Zang, "Optimistic transaction processing in deterministic database," *Journal of Computer Science and Technology*, vol. 35, pp. 382–394, 3 2020.

[61] N. Suri, "Distributed systems security," 10 2019.

[62] K. Davis, M. Schulte, and B. Uekermann, "Enhancing quasi-newton acceleration for fluid-structure interaction," *Mathematical and Computational Applications*, vol. 27, pp. 40–40, 5 2022.

[63] "International journal of distributed systems and technologies," 8 2024.

[64] K. He, "Research on verifiable access technology of mixed database in distributed system under big data," in *2021 International Conference on Aviation Safety and Information Technology*, pp. 636–640, ACM, 12 2021.

[65] 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.

[66] B. W. Watson, E. Badouel, and O. Niang, "Foreword of proceedings of cari 2020," 8 2020.

[67] 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.

[68] Y. Braidiz, D. Efimov, A. Polyakov, and W. Perruquetti, "On robustness of finite-time stability of homogeneous affine nonlinear systems and cascade interconnections," *International Journal of Control*, pp. 1–11, 9 2020.