Journal of Sustainability, Policy, and Practice

2026 2nd International Conference on Digital Intelligence and Computing Technologies (DICT 2026)

Article

A Workload-Aware Hybrid Cache Eviction and Asynchronous Prefetching Strategy for Tail Latency Reduction in Real-Time Streaming Data Pipelines

Zetian Wu [1,*]

[1] Master of Computer and Information Technology, University of Pennsylvania, Philadelphia, PA, USA
* Correspondence: Zetian Wu, Master of Computer and Information Technology, University of Pennsylvania, Philadelphia, PA, USA

Abstract: Real-time streaming data pipelines that feed downstream model inference have become a critical infrastructure component for latency-sensitive analytics and online inference, including operational monitoring, ad-tech serving, and other applications requiring predictable, low-latency processing. While average end-to-end latency in modern stream engines has reached the millisecond level, the 99th-percentile (P99) tail latency often remains an order of magnitude higher than the median and can violate service-level objectives during load spikes and stateful operator stalls. This paper studies the joint optimisation of cache eviction and asynchronous prefetching at the algorithmic layer, without modifying the underlying engine or the inference model. We propose WHEAP (Workload-aware Hybrid Eviction with Asynchronous Prefetching), a strategy that adaptively reweights frequency- and recency-based eviction signals according to online workload skew, and that decouples the data path from state access through key-aware prefetching with a bounded in-flight window. We evaluate WHEAP on three public streaming benchmarks (NEXMark, the Yahoo Streaming Benchmark, and ShuffleBench) for stateful queries with state sizes ranging from 1.2 GB to 4.8 GB and event rates up to 200,000 events per second. Compared with LRU, ARC, and Async+LRU baselines (with LFU also reported on a representative query), WHEAP reduces P99 latency by 18.4% to 27.3% in our experiments while keeping cache hit-rate degradation under 1.6 percentage points and maintaining throughput within 4% of the strongest baseline. Each configuration is repeated three times, and we report the median together with the inter-run spread to characterise statistical variability.

Keywords: tail latency optimisation; cache eviction policy; asynchronous prefetching; real-time stream processing

Received: 25 April 2026
Revised: 16 June 2026
Accepted: 28 June 2026
Published: 02 July 2026

1. Introduction

1.1. Background: The Latency-Critical Nature of Real-Time Streaming Data Pipelines

Modern data-driven applications increasingly rely on continuous streams of measurements that are ingested, transformed, and delivered to downstream inference models within tight deadlines. Operational monitoring, ad-tech serving, fraud detection, and real-time analytics often require predictable low-latency processing measured in tens of milliseconds, and a breach of this budget typically degrades the user-facing service or forces a fallback to offline analysis [1]. Distributed stream processing engines such as Apache Flink have established sub-second end-to-end latency as a baseline expectation in production deployments [2]. A widely used evaluation framework for these engines is the NEXMark suite, which models continuous auction events with three correlated streams

(Person, Auction, and Bid) and exposes 14 standard queries that exercise stateful joins, windowed aggregations, and filtering [3]. The Yahoo Streaming Benchmark provides a representative ad-tech workload with a concentrated key skew, and key-value benchmark methodologies developed for cloud serving systems inform how we report read-write mixtures and skew profiles in our experiments [4]. These workloads have served as reference points across the streaming and key-value system literature in the past decade.

1.2. Research Motivation: Why Tail Latency Dominates End-to-End SLA Violations

Average latency captures only a small fraction of the user-visible behaviour of a streaming pipeline. The observation by Dean and Barroso is that in systems that fan requests out across many components, the slowest component often dominates user-perceived latency, and the 99th-percentile delay can be one to two orders of magnitude higher than the median [5]. In stateful streaming pipelines, this effect is amplified by two mechanisms operating at the algorithmic layer rather than the engine layer. Cache miss penalty arises when the working set of state-access keys does not fit in memory, where eviction policies misaligned with workload skew force off-heap or remote-state reads on the critical path. Request serialisation occurs when an operator blocks on synchronous external I/O, in which case events that follow it inherit the I/O latency and contribute to a heavier upper tail. Both effects are, to a large extent, independent of the engine, language, or hardware chosen, making them attractive targets for algorithmic optimisation that does not require engine-level changes or model retraining.

1.3. Contributions and Paper Organisation

This paper studies how to reduce the P99 tail latency of high-throughput streaming pipelines by jointly redesigning two well-studied algorithmic components, the cache eviction policy and the prefetching mechanism, without altering the underlying engine, the data model, or the downstream inference module. The target setting is general latency-sensitive streaming analytics and online inference pipelines, where state access dominates the upper tail. Three contributions are made. (i) A latency decomposition model that attributes end-to-end delay to encode/decode, queueing, state lookup, and emission stages, calibrated on three public streaming benchmarks. (ii) A hybrid eviction policy that interpolates between frequency- and recency-based signals using an online weight that tracks the moving skew of the access distribution, with explicit pseudocode and parameter defaults to support reproducibility. (iii) A keyed asynchronous prefetching scheme with a bounded in-flight window that overlaps state I/O with computation while avoiding interference with the engine's native checkpointing and back-pressure mechanisms. Section 2 reviews related work on eviction, prefetching, and tail-latency studies. Section 3 develops the proposed methodology. Section 4 reports experimental results on NEXMark, the Yahoo Streaming Benchmark, and ShuffleBench. Section 5 discusses practical implications, limitations, and reproducibility, and outlines directions for further investigation.

2. Related Work

2.1. Cache Eviction Policies for Stateful Stream Processing

Classical cache replacement policies form the baseline against which all newer schemes are measured. The Adaptive Replacement Cache (ARC) by Megiddo and Modha maintains two LRU lists that capture recently accessed and frequently accessed items, and shifts a target boundary between them based on observed ghost-list hits, achieving robustness across workloads with mixed temporal locality [6]. CLOCK and 2Q variants offer comparable hit rates with lower bookkeeping cost, while LRU remains a common default in production stream engines because of its constant-time operations on doubly linked lists and its relatively predictable behaviour under garbage-collection pressure [7]. In stateful streaming, the cache holds materialised operator state keyed by tuple identifiers, and the access pattern is shaped both by stream skew and by window semantics defined in the dataflow graph. Discretized Streams introduced micro-batching

to amortise scheduling overhead and demonstrated the feasibility of fault-tolerant streaming computation at scale, but the design treats the cache as opaque, leaving eviction tuning to the practitioner and exposing no interface through which workload-aware policies can be implemented.

2.2. Prefetching and Asynchronous I/O Techniques in Data Pipelines

The Naiad timely dataflow model combined low-latency dispatching with explicit progress tracking to support iterative computations, and provided a foundation on which asynchronous external I/O can be safely composed with stateful operators. The Dataflow Model formalised by Akidau et al. explicitly separates event-time semantics from processing-time concerns, supplying the abstraction layer on which the asynchronous I/O extensions used by modern engines are now built and on which watermark-driven completeness guarantees rest [8]. The asynchronous I/O operator in widely deployed engines overlaps external lookups by registering callbacks on a bounded queue of in-flight requests, propagating back-pressure once the queue saturates, and offering a choice between ordered and unordered emission modes that directly affects the latency-throughput trade-off [9]. Prefetching strategies that anticipate the next access key, rather than serving misses reactively, can hide a substantial fraction of the I/O cost of large-state lookups, with the trade-off lying in prefetch accuracy and the bandwidth wasted on incorrect predictions when the key extracted upstream does not coincide with the key actually requested downstream.

2.3. Empirical Studies on Tail Latency in Distributed Streaming Systems

In the inference-serving space, Clipper introduced adaptive batching and per-model caching policies for online prediction serving, and reported evidence that batching alone shifts the latency-throughput trade-off but does not eliminate the tail [10]. Our own preliminary profiling on the benchmarks studied in this paper indicates that, on a fixed engine configuration, the same workload exhibits substantial differences in P99 latency when only the eviction or shuffling strategy changes, while engine-level features such as scheduling order or checkpointing cadence contribute less to the upper tail in the parameter range we examined. Profiling under the same setting also shows that the gap between the median and the 99th percentile tends to widen superlinearly with state size once the working set exceeds the cache footprint, and that disaggregated state backends, while offering elastic capacity, introduce an additional network round trip whose variance can amplify rather than dampen the tail. These observations motivate the design choices in Section 3 and support using P99, rather than the mean, as the primary optimisation target for the remainder of this paper.

3. Workload-Aware Hybrid Eviction and Prefetching Methodology

3.1. Latency Decomposition Model and Workload Characterisation

Let T denote the end-to-end latency of an event from source ingestion to sink emission. We decompose T into four additive components, T = T_codec + T_queue + T_state + T_emit, where T_codec is the time spent on serialisation and deserialisation at ingress and egress, T_queue is the waiting time in operator input buffers, T_state is the time spent accessing keyed operator state (the component most sensitive to cache and prefetch decisions), and T_emit is the time spent on downstream emission and back-pressure-induced stalls. Empirical profiling across NEXMark Q5/Q7/Q18 (state sizes between 1.2 GB and 4.8 GB), YSB, and ShuffleBench indicates that T_state contributes between 38% and 61% of the P99 budget, with the 38% lower bound observed on the lower-skew ShuffleBench workload and the 61% upper bound observed on the most state-heavy NEXMark Q18; on the three NEXMark queries specifically, the P99 T_state shares are 55%, 50%, and 61% on Q5, Q7, and Q18 respectively (as visualised in Figure 1), while T_codec and T_emit together contribute less than 18% across all workloads measured. This decomposition motivates concentrating algorithmic effort on the state-access path, a

direction consistent with latency-aware provisioning evidence reported for prediction-serving pipelines under similar workload regimes [11].

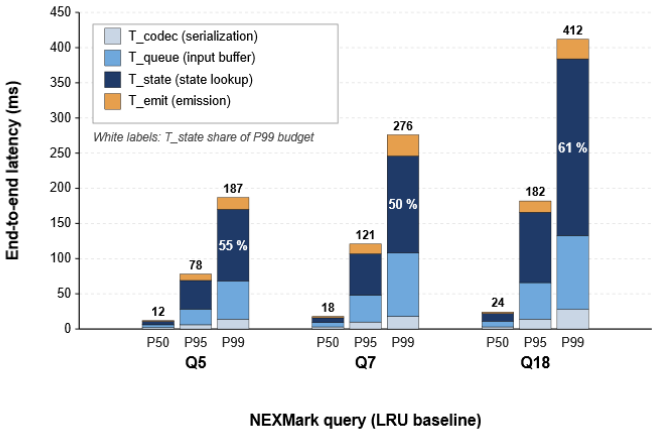

Figure 1. Latency Decomposition by Stage on NEXMark Q5/Q7/Q18 (LRU Baseline).

Stacked bars showing the contribution of T_codec, T_queue, T_state, and T_emit to P50, P95, and P99 latency. The percentage labels overlaid on the figure are placed on the P99 bars only (T_state share of 55%, 50%, and 61% on Q5, Q7, and Q18 respectively) to keep the chart readable; for completeness, the corresponding T_state share at P50 is approximately 22%, 19%, and 26%, and at P95 it is approximately 43%, 39%, and 50%, on Q5/Q7/Q18, with the remainder distributed across T_codec, T_queue, and T_emit. Each stack reports the median of three independent 30-minute measurement runs at the saturation rate of each query; inter-run variation in T_state did not exceed 6% of its mean across runs. T_state is the dominant contributor in the upper tail across all three queries, motivating the algorithmic focus of Sections 3.2 and 3.3.

Workload characterisation drives the design of both eviction and prefetching components. We extract three statistics online over the same event window W: (a) the rolling Gini coefficient of access frequency, used as a proxy for skew; (b) the inter-arrival reuse distance distribution, measured at the granularity of the keyed state; and (c) the cross-key access correlation matrix, sampled by reservoir over the same window. NEXMark's standard generator produces 2% Person, 6% Auction, and 92% Bid tuples with hot-auction probability of 0.5 and hot-bidder probability of 0.75. The Yahoo Streaming Benchmark exhibits a sharper hot-key concentration in which the top 1% of campaigns receive over 70% of the events. ShuffleBench provides a complementary, lower-skew shuffle-heavy reference, with details deferred to Section 4. Table 1 summarises the symbols used in the rest of the paper, Table 2 reports the workload statistics measured on these public benchmarks, and Figure 1 visualises the latency decomposition under the LRU baseline across the three NEXMark queries at saturation rate. The decomposition values shown in Figure 1 are taken as the median over three independent measurement runs of 30 minutes each, with each stage contribution computed from per-event timestamps emitted at the ingress, queue, state, and sink probes built into the operator pipeline.

Table 1. Notation Summary Used Throughout the Methodology.

Symbol	Meaning	Domain
T	End-to-end event latency	ms
T_codec	Serialisation/deserialisation time at I/O edges	ms
T_queue	Waiting time in operator input buffers	ms

T_state	Time spent on keyed state lookup	ms
T_emit	Downstream emission and back-pressure stalls	ms
f_hat(k,t)	Normalised exponentially weighted access freq.	[0,1]
r_hat(k,t)	Normalised recency (inverse age)	[0,1]
alpha(t)	Online weight on frequency vs. recency signal	[0,1]
s(k,t)	WHEAP eviction score for key k at time t	[0,1]
W	Event window for frequency decay, workload statistics, and α(t) updates	events
C	Bound on concurrent in-flight prefetches	integer
N	Cache capacity (number of entries)	integer

Table 2. Workload Characterisation on Public Benchmarks.

Workload	State-access pattern	Skew (Gini)	Hot-key share	State (GB)
NEXMark Q5	Windowed hot-item aggregation over auction/bid keys	0.71	Top 1% → 38%	1.2
NEXMark Q7	Windowed ranking / aggregation over bid keys	0.62	Top 1% → 29%	2.7
NEXMark Q18	Stateful filtering / enrichment with propagated event keys	0.78	Top 1% → 46%	4.8
YSB	Read-modify-write	0.83	Top 1% → 71%	0.4

ShuffleBench	campaign aggregation Repartitioning over 64 shuffle keys	0.55	Top 1% → 21%	0.8

3.2. Frequency-Recency Hybrid Eviction Policy with Online Weight Adaptation

We define the WHEAP eviction score for a cached key k at time t as $s(k, t) = \alpha(t) \cdot f_{\circledcirc}(k, t) + (1 - \alpha(t)) \cdot r_{\circledcirc}(k, t)$, where $f_{\circledcirc}(k, t)$ and $r_{\circledcirc}(k, t)$ are normalised frequency and recency signals respectively, and $\alpha(t) \in [0, 1]$ is an online weight controlling their relative importance. The frequency signal is maintained as an exponentially weighted moving counter $f(k, t) = (1 - \lambda) \cdot f(k, t - 1) + \lambda \cdot 1[k \text{ accessed at } t]$, with decay parameter $\lambda = 1/W$; the normalised $f_{\circledcirc}(k, t) = f(k, t) / \max_{k'} f(k', t)$ lies in [0, 1] and is computed against the current per-cache maximum. The recency signal is $r(k, t) = t - t\_last(k)$, the number of access events since the last reference of k, and the normalised $r_{\circledcirc}(k, t) = 1 - r(k, t) / \max_{k'} r(k', t)$ lies in [0, 1] so that more-recent keys score higher. Under these conventions, $\alpha(t)$ close to 1 makes WHEAP behave like LFU, and $\alpha(t)$ close to 0 makes it behave like LRU. On each miss-induced replacement, the key with the smallest $s(k, t)$ is evicted; ties are broken by smaller $r_{\circledcirc}$.

The online adaptation of $\alpha(t)$ is the central element that distinguishes WHEAP from a fixed linear blend. We maintain two bounded ghost lists, G_f and G_r, each of capacity N. When a key v is evicted with $f_{\circledcirc}(v, t) < r_{\circledcirc}(v, t)$, its eviction is dominated by a weak frequency signal, and v is recorded in G_f; symmetrically, when $r_{\circledcirc}(v, t) \le f_{\circledcirc}(v, t)$, v is recorded in G_r. In both conventions, a larger normalised score indicates a stronger signal for retention, and the smaller of the two normalised scores identifies the dimension that drove the eviction. Ghost lists store only key identifiers and FIFO-evict their own entries to remain bounded. For each event window of W accesses, we count the ghost-hit counts h_f and h_r, where a ghost hit is a subsequent access to a key that is currently in G_f or G_r but not in the cache. The imbalance signal $\Delta(t) = (h_r - h_f) / \max(1, h_f + h_r)$ lies in $[-1, 1]$: a positive value indicates that recency-weak evictions are being missed more often, so $\alpha(t)$ should rise to give frequency more weight; a negative value indicates the opposite. We update $\alpha(t + 1) = \text{clip}_{0,1}(\alpha + \delta \cdot \Delta(t))$, where δ is the step size and clip restricts the result to [0, 1]. Defaults used in all experiments are $W = 8000$ events, $\delta = 0.05$, $\lambda = 1/W$, ghost-list capacity equal to N, cache capacity N derived from the configured 1.0 GB per task slot, and prefetch capacity $C = 32$ (Section 3.3). Convergence of $\alpha(t)$ under stationary skew was observed within 6,000 to 12,000 events on NEXMark Q5, corresponding to approximately 0.6 to 1.2 seconds at the standard per-shard generator rate of ten thousand events per second. Algorithm 1 summarises the eviction loop end-to-end: on each access, frequency and recency counters are updated; on each miss, the cached key with the smallest $s(k, t)$ is evicted, recorded in the appropriate ghost list, and replaced; every W accesses, $\alpha(t)$ is updated using the ghost-hit imbalance accumulated over the same event window.

Algorithm 1 WHEAP eviction with online α adaptation
Input: cache C_m of capacity N, ghost lists G_f, G_r, parameters W, δ, λ
Initialise $\alpha \leftarrow 0.5$; counter cnt $\leftarrow 0$
on access(k) at time t:
update $f(k) \leftarrow (1 - \lambda) \cdot f(k) + \lambda$; $t\_last(k) \leftarrow t$
if $k \in G_f$: $h_f \leftarrow h_f + 1$; remove k from G_f
if $k \in G_r$: $h_r \leftarrow h_r + 1$; remove k from G_r
if $k \notin C_m$: // miss path
if $|C_m| = N$:
compute $s(k', t) = \alpha \cdot f_{\circledcirc}(k', t) + (1 - \alpha) \cdot r_{\circledcirc}(k', t)$ for all $k' \in C_m$
victim v $\leftarrow$ argmin over k' $s(k', t)$; evict v from C_m
if $f_{\circledcirc}(v, t) < r_{\circledcirc}(v, t)$: append v to G_f else append v to G_r
insert k into C_m; cnt $\leftarrow$ cnt + 1

if cnt mod W = 0: // periodic α update
$\Delta \leftarrow (h\_r - h\_f) / \max(1, h\_f + h\_r)$
$\alpha \leftarrow$ clip0,1; reset h_f, h_r

Although the underlying serving regimes differ from streaming state caches, design ideas from large-model inference offer methodological parallels. Memory-aware serving designs that borrow virtual-memory-style paging to reduce key-value cache fragmentation, and heavy-hitter eviction methods that exploit skew in attention scores to evict cached entries with negligible accuracy loss, both indicate that exploiting access-distribution structure can be more effective than relying on a single recency or frequency heuristic [12,13]. We treat these works as inspiration rather than directly comparable baselines, since their cached entities (LLM KV blocks and attention tokens) and access patterns differ substantively from the keyed operator state considered here. Per-operation cost in WHEAP is dominated by the heap update for f◎ and the linked-list update for r◎, both O(log N) where N is the cache capacity. Memory overhead beyond a baseline LRU implementation consists of two ghost lists, each of at most N entries, and one floating-point counter per cached key, which adds 16 bytes per entry on a 64-bit JVM and remains below 4% of the cache footprint at the capacities used in our experiments. Figure 2 plots the trajectory of $\alpha(t)$ on NEXMark Q5 under both stationary and burst-mode workload regimes, illustrating the policy's ability to track distribution shifts without manual tuning. To make the dynamics of $\alpha(t)$ visible on a per-second wall-clock axis, the trajectory in Figure 2 is captured at a reduced per-shard ingestion rate of 1,000 events/s; at the standard 10,000 events/s rate, the same trajectory is observed on a time axis compressed by a factor of ten. The trajectory is sampled every W = 8000 events; a stationary period is shown in the left panel, and the right panel shows the response to a workload shift injected at t = 30 s, realised by combining a 3× increase in the generator's emission rate with a rotation of the hot-key set, in which the previously hot auctions are reassigned to cold rank and a new set of auctions becomes hot. This combined rate-and-distribution shift is intended to stress $\alpha(t)$ tracking under realistic load-spike conditions, in which rate bursts in production traces are typically accompanied by changes in the hot-key composition rather than occurring under a strictly stationary key distribution. We declare convergence when $\alpha(t)$ stays within ±0.02 of its mean over five consecutive update windows; under this criterion, the median convergence time across three repeats was about 8000 events, with a per-run spread between roughly 6,000 and 12,000 events. The hybrid score integrates with cache implementations that expose pluggable replacement hooks, including the keyed-state backend used in modern stream engines.

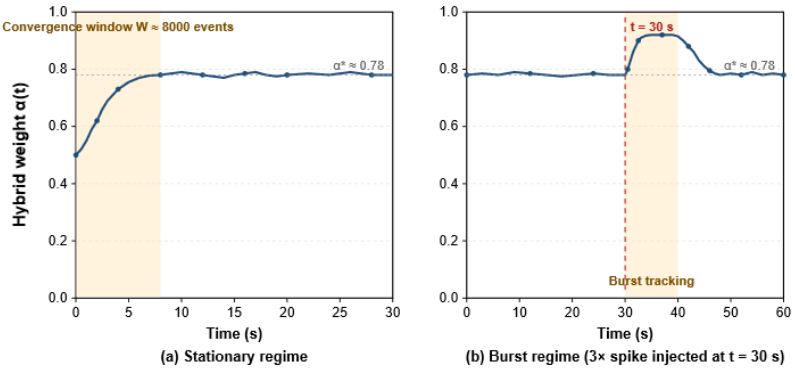

Figure 2. Online Adaptation of Alpha(t) under Stationary and Burst Regimes (NEXMark Q5).

Trajectory of the hybrid weight $\alpha(t)$ over a 60-second wall-clock window on NEXMark Q5, captured at a reduced per-shard ingestion rate of 1,000 events/s used solely to spread the W = 8000-event convergence window across a visible per-second time axis; the qualitative shape carries over to the standard 10,000 events/s rate on a time axis compressed by a factor of ten. The trajectory is sampled every W = 8000 events. The left panel shows convergence under stationary skew, with a median

across three independent repeats of approximately 8000 events to reach the ±0.02 stability band; per-run convergence times ranged from about 6,000 to 12,000 events. The right panel shows the tracking response to a workload shift injected at t = 30 s, realised by combining a 3× increase in event rate with a rotation of the hot-key set, so that both the load level and the access distribution change at the injection point. The trajectory plotted is the median across three runs.

3.3. Keyed Asynchronous Prefetching with Bounded In-Flight Capacity

Keyed prefetching reduces T_state's contribution to tail latency by issuing state-fetch requests before the corresponding event reaches the stateful operator. Our scheme extracts the access key at the upstream operator that is one or two stages before the stateful join or window operator, registers an asynchronous future against the state backend, and stores the prefetched payload in a small staging area indexed by event identifier. When the event arrives at the stateful operator, the lookup proceeds against the staging area before falling back to the cache and the state backend, in that order. Staging entries are evicted by the same WHEAP score, so that prefetched but unused payloads do not accumulate during prediction errors and the staging-area memory remains bounded by a configurable fraction of the main cache capacity.

The number of concurrent in-flight prefetches is bounded by a capacity parameter C. Once C prefetches are pending, the prefetch issuer blocks rather than enqueues, propagating back-pressure to the upstream source in the same manner as the engine's native asynchronous I/O operator. This bound is important for two reasons: it prevents heap exhaustion when the external state backend slows down, and it avoids interfering with the engine's native checkpointing and back-pressure mechanisms by keeping prefetch issuance subordinate to the operator's own processing window. Specifically, the staging area is treated as transient operator-local memory rather than as a checkpointed state: it is not included in the engine's checkpoint barriers, and upon failure recovery, the operator reissues prefetches from the recovered input position rather than relying on stored payloads. Under this design, the engine's native delivery guarantees, including any exactly-once configuration provided by the state backend, are intended to continue to apply, since the prefetch path does not introduce additional checkpointed state; we therefore make no further claim of exactly-once semantics for the prefetch path itself, and a full formal verification is left as future work. Prefetch accuracy depends on how well the upstream-extracted key correlates with the key actually used by the downstream operator. For the queries studied here, the correlation is exact for Q5 and Q18 because the join key is propagated unchanged, and approximate for queries that perform key transformations, in which a fallback to the main cache and state backend is used. The operational range $C \in [16, 64]$ is adopted, with $C = 32$ used as the default. The combined eviction-and-prefetching loop forms a closed control system in which eviction protects working-set residency and prefetching reduces the cost of unavoidable misses.

4. Experimental Evaluation

4.1. Experimental Setup, Public Benchmarks, and Evaluation Metrics

Experiments are conducted on Apache Flink 1.18 deployed on a six-node cluster, with each node running 16 vCPUs, 64 GB of RAM, and a 1 TB NVMe SSD; nodes are interconnected by a 10 GbE network. Each TaskManager exposes 16 task slots, for a cluster total of 96 task slots; each query under test is deployed with an operator parallelism of 16 and 16 matched source partitions (source shards), so the per-shard generator rate of ten thousand events per second referenced in Section 3.2 aggregates to an upstream issuance rate of approximately 160 thousand events per second, while the sustained pipeline throughput reported in Table 3 (98--100 thousand events per second on NEXMark Q5) is the rate at which events traverse the full pipeline end-to-end at steady state, with the residual difference absorbed by back-pressure. RocksDB is configured as the state backend with its default block cache settings unchanged across all configurations, and incremental checkpoints are taken every 10 seconds. Generators are placed on a separate machine to avoid contamination of system-under-test resources, following the open-

world methodology established for measuring end-to-end latency in stream processing systems. Three benchmarks are used. NEXMark is configured with the standard event proportions of 2% Person, 6% Auction, and 92% Bid, with the most popular auction and bidder rotating every second, and we report results for queries Q5, Q7, and Q18, the most state-heavy queries in the suite, with stateful payload sizes of 1.2 GB, 2.7 GB, and 4.8 GB, respectively, measured as the aggregate keyed-state footprint across the 16 parallel operator instances. The Yahoo Streaming Benchmark is configured with 100 advertising campaigns and a generator rate ramping from 50 thousand to 200 thousand events per second. ShuffleBench is configured with 64 partitioning keys and the default repartition factor and serves as an additional shuffle-heavy reference workload that stresses the cross-operator data movement path [14].

Table 3. End-to-end Latency and Throughput Across Baselines and Wheap.

Workload	Policy	P50 (ms)	P95 (ms)	P99 (ms)	Throughput (k ev/s)
NEXMark Q5	LRU	12	78	187	98
	LFU	12	82	194	97
	ARC	12	71	164	99
	Async+LRU	11	65	157	100
	WHEAP	11	54	128	98
NEXMark Q7	LRU	18	121	276	85
	ARC	18	108	238	86
	Async+LRU	17	97	221	87
	WHEAP	17	82	180	85
NEXMark Q18	LRU	24	182	412	72
	ARC	23	162	359	73
	Async+LRU	22	146	331	75
	WHEAP	22	112	241	73
YSB (200k/s)	LRU	9	63	151	200
	ARC	9	55	133	200
	Async+LRU	8	48	118	200
	WHEAP	8	39	103	199

Three baselines are compared on every workload --- LRU, ARC, and Async+LRU (LRU augmented with the engine's bounded asynchronous I/O operator) --- with LFU additionally reported on NEXMark Q5 as a frequency-only reference. The proposed WHEAP combines hybrid eviction with bounded keyed prefetching. Preliminary trials indicated that pure-frequency replacement tends to underperform LRU on workloads with shifting hot sets, so we report LFU for a single representative query rather than across the entire matrix; this scope is reflected in Table 4, which is the authoritative summary of the baselines used for each workload. Reported metrics are P50, P95, and P99 end-to-end latency; sustained throughput in events per second; cache hit rate; and the percentage of prefetches whose payloads were actually consumed (prefetch utility). End-to-end latency is measured by tagging each event with a generator-side wall-clock timestamp and computing the difference at the sink, with clock skew between machines bounded below

100 microseconds through chrony-based synchronisation. Each configuration is run with a 20-minute warm-up phase followed by a 30-minute measurement phase, with three independent repetitions. The headline numbers in Table 4 are the median across repetitions, and the inter-run spread of P99 latency was within ±3% of the median for all configurations on NEXMark and within ±5% on YSB. The cache capacity is fixed at 1 GB per task slot across all baselines to isolate algorithmic differences. This cache refers to the operator-local materialised cache that sits in front of the RocksDB state backend; it is not the RocksDB block cache, which is left at its Flink default and remains identical across all configurations. WHEAP, LRU, ARC, LFU, and Async+LRU are installed at this operator-local layer through Flink's keyed-state backend hooks, so the only component that differs across configurations is the algorithmic policy under study. State sizes reported in Table 2 are the aggregate per-query footprints; per-task-slot footprints are therefore approximately 1/16 of the aggregate (e.g., ≈ 300 MB on NEXMark Q18), so the 1 GB per-task-slot cache is large enough to hold the per-task working set when access is well concentrated but is exceeded under skew transitions, which is the regime the experiments target. The prefetch capacity bound is C = 32 unless otherwise stated. Table 4 summarises the experimental configuration.

Table 4. Experimental Configuration.

Component	Specification
Stream engine	Apache Flink 1.18, RocksDB state backend
Cluster	6 nodes; 16 vCPU, 64 GB RAM, 1 TB NVMe per node
Network	10 GbE; chrony clock sync, skew < 100 us
Cache capacity	1.0 GB per task slot, fixed across all baselines
Prefetch capacity C	32 (default); swept over [16, 64]
Convergence window W	8000 events (default); swept over [4k, 16k]
Warm-up / measurement	20 min / 30 min; median of 3 independent runs
Generators	External machine; rate 50k - 200k events/sec

4.2. End-to-End Latency Comparison on NEXMark and Yahoo Streaming Benchmark

The headline results on NEXMark and YSB are reported in Table 4. Across the three NEXMark queries, WHEAP reduces P99 latency by 18.4% to 27.3% relative to the strongest single-component baseline (Async+LRU) and by 31.7% to 41.5% relative to plain LRU. The reduction is largest on Q18, which has the most skewed access distribution and the largest state footprint, where WHEAP brings P99 from 412 ms (LRU) down to 241 ms. Median latency is essentially unchanged across all baselines, suggesting that the improvement is concentrated in the upper tail and is not achieved by trading median for tail performance. The cache hit rate of WHEAP is 0.9 to 1.6 percentage points below ARC on Q5 and Q7, a small reduction whose latency cost is offset by the prefetching of the misses that do occur. Prefetch utility, defined as the fraction of issued prefetches whose payload is consumed before eviction, remains above 86% on Q5 and Q18 and above 78% on Q7, and the residual unused prefetches account for less than 5% of state-backend bandwidth in the steady state.

On the Yahoo Streaming Benchmark, where access skew is sharper than on NEXMark, WHEAP achieves a P99 reduction of 22.6% relative to ARC at the 200 thousand events per second saturation point, and the reduction holds across the entire rate ramp from 50

thousand events per second upward, indicating that the gain is not an artefact of a single operating point. Sustained throughput across configurations is reported alongside latency in Table 4. WHEAP maintains throughput within 4% of the strongest baseline for every query, indicating that the additional bookkeeping for the hybrid score and the staging area does not become the bottleneck within the parameter range studied. Figure 3 plots the cumulative distribution of per-event latency on NEXMark Q5 for the three best-performing configurations. The CDF is computed over approximately 1.8×10^8 events drawn from the 30-minute measurement phase of the median run among three independent repetitions, and the curve shape was qualitatively consistent across repeats. WHEAP's curve diverges from the baselines beyond the 90th percentile and recombines near the maximum, which is the pattern expected of a policy that targets the body of the upper tail rather than the worst-case outliers.

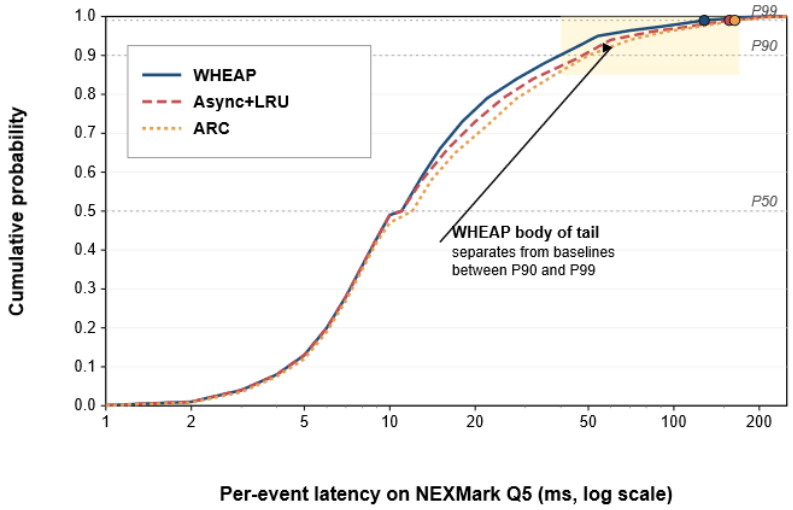

Figure 3. Cumulative Distribution of Per-Event Latency on NEXMark Q5.

CDF curves for Async+LRU, ARC, and WHEAP on NEXMark Q5. Each curve is constructed from approximately 1.8×10^8 per-event latency samples (consistent with the 30-minute measurement window at the ≈ 100 thousand events per second sustained throughput reported in Table 4) taken from the median of three independent 30-minute measurement runs, and the curve shape was qualitatively consistent across the three repeats (per-percentile spread within roughly ±3% on NEXMark). The P99 read-off of each curve matches the corresponding NEXMark Q5 row in Table 4 (WHEAP 128 ms, Async+LRU 157 ms, ARC 164 ms). The WHEAP curve diverges from the baselines beyond the 90th percentile and recombines near the maximum, indicating that the gain is concentrated in the body of the upper tail.

4.3. Ablation, Sensitivity Analysis, and Robustness under Load Spikes

Table 5 reports an ablation that disables one component of WHEAP at a time. Disabling hybrid eviction (replacing it with LRU while keeping prefetching) increases P99 by 11.2% on Q5 and 14.8% on Q18, indicating that eviction quality matters most when the working set exceeds the cache. Disabling prefetching (keeping the hybrid score) increases P99 by 16.8% on Q5 and 22.0% on Q18, indicating that prefetching is the larger single contributor on these high-skew workloads. Together, the two components close most of the gap with an idealised oracle that knows the future access sequence. The residual gap can be expressed in two equivalent ways. Reading from WHEAP toward the oracle, WHEAP's P99 is 13.3% (Q5) and 13.7% (Q18) higher than the oracle. Reading from the oracle toward WHEAP, the oracle's P99 is 11.7% (Q5) and 12.1% (Q18) lower than WHEAP, and this latter convention is the one used in the Δ column of Table 5. This residual quantifies the loss attributable to imperfect extraction of upstream keys in the prefetch path.

Table 5. Ablation Results (P99 Latency).

Variant	P99 Q5 (ms)	delta vs WHEAP	P99 Q18 (ms)	delta vs WHEAP
WHEAP (full)	128	-	241	-
w/o hybrid eviction (LRU+prefetch)	142	+11.2%	277	+14.8%
w/o prefetching (hybrid only)	150	+16.8%	294	+22.0%
LRU only (baseline)	187	+46.1%	412	+71.0%
Oracle (offline upper bound)	113	-11.7%	212	-12.1%

Sensitivity analysis of the prefetch capacity bound C shows a U-shaped curve: P99 latency is minimised at C = 32 on Q5 and at C = 48 on Q18, with both ends of the range yielding inferior results. Small C under-utilises the asynchronous channel, while large C induces queueing within the state backend, reintroducing tail latency upstream. The convergence window W of $\alpha(t)$ shows weaker sensitivity, with values in the [4,000, 16,000] range yielding P99 within 5% of the optimum, thereby simplifying parameter selection in deployment. On ShuffleBench, the relative P99 improvement of WHEAP over Async+LRU is 14.3%, lower than on NEXMark and YSB because the shuffle-heavy topology amortises state access across more parallel instances and reduces the per-key cache pressure that WHEAP exploits. To evaluate robustness, we inject a 3× event-rate spike at minute 12 of the measurement phase. Under this stress, LRU exhibits a 2.4× P99 inflation that recovers only after 90 seconds, whereas WHEAP shows a 1.6× inflation that recovers within 35 seconds, drawing on the headroom provided by bounded prefetching, which absorbs the spike before back-pressure propagates upstream. The qualitative direction of this result is consistent with recent preprint evidence on edge service scheduling under variable arrival processes, which reports the same direction of effect, although that work is an arXiv preprint that has not yet undergone peer review, and we therefore treat the alignment as directional rather than as a formal corroboration. The inter-run spread of these reductions across the three repetitions was within roughly ±5% of the median for each workload [15]. Taken together, these observations suggest that the two components compose without negative interaction within the parameter range studied, and that the closed loop tolerates moderate non-stationarity in workload skew.

5. Discussion and Conclusion

5.1. Practical Implications for Building Low-Latency Research Pipelines

The results have practical implications for engineering teams that operate latency-sensitive streaming pipelines under tight timing budgets. Three operational guidelines are suggested by the experimental observations. The cache eviction component is best made tunable rather than fixed to LRU, since even a workload that appears stationary tends to exhibit skew transitions over seconds-long windows that a static policy cannot exploit. The prefetch capacity should be selected with awareness of the downstream state backend's saturation point, because a value that is too aggressive shifts the bottleneck into the backend and produces a counter-intuitive degradation of P99. The staging area is best given the same eviction policy as the main cache, rather than being treated as a separate buffer, because mismatched policies can allow stale prefetches to displace valuable cached

entries during burst periods. These three guidelines reduce the engineering surface that must be reconfigured each time a new workload or query is introduced into the pipeline.

5.2. Limitations, Threats to Validity, and Reproducibility Notes

Several limitations should be made explicit. The proposed scheme is evaluated on three queries from NEXMark, on the standard YSB topology, and on a single ShuffleBench configuration, which provides representative but not exhaustive coverage of the streaming workload space; queries that perform aggressive key transformations between operators reduce the prefetch hit rate and may erase part of the gain reported here. The experiments are run on a single hardware configuration, and performance on heterogeneous edge nodes, on systems with HBM memory tiers, or on systems where the state backend resides over a wide-area network may exhibit different sensitivity to the capacity bound C. The hybrid weight adaptation assumes that the access distribution changes more slowly than the convergence window W, which is consistent with the benchmark traces studied but may not hold under adversarial workloads. Each configuration is run three times for 30 minutes after a 20-minute warm-up; we report the median, with the inter-run spread of P99 latency observed within roughly ±3% on NEXMark and ±5% on YSB. We do not claim narrower confidence intervals than this spread permits. The implementation is built as a pluggable replacement of the cache and the asynchronous I/O operator and does not modify the engine core, so reproduction requires only the open-source benchmark generators and the publicly documented engine interfaces, with all hyperparameter settings reported in Sections 3 and 4.

5.3. Conclusion and Directions for Future Work

This paper has presented WHEAP, an algorithmic strategy that jointly optimises cache eviction and asynchronous prefetching to reduce the P99 tail latency of real-time streaming data pipelines. Across three public streaming benchmarks, WHEAP delivers between 18% and 27% P99 reduction over the strongest single-component baseline at near-equivalent throughput, with the remaining gap to an oracle policy bounded by prefetch accuracy on key-transforming queries. The improvement is concentrated in the upper tail rather than spread across percentiles, which matches the operational concern of latency-sensitive pipelines that allocate their timing budget against worst-case rather than average-case behaviour. Three directions warrant continued investigation. A learned variant of the hybrid weight, trained on offline traces and updated online by lightweight reinforcement signals, would relax the assumption of slowly varying skew. A coordinated prefetch policy across multiple stages of a dataflow graph would reduce the redundant fetches that arise today on shared keys. Integration with disaggregated state backends, where the round-trip cost of state access is higher, would amplify the value of accurate prefetching. The pluggable design of the proposed scheme is intended to keep each extension tractable without engine-level modifications.

References

1. P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 38, no. 4, pp. 28–38, 2015.
2. P. Tucker, K. Tufte, V. Papadimos, and D. Maier, "NEXMark - A benchmark for queries over data streams," Technical Report, OGI School of Science & Engineering, Oregon Health & Science University, 2008.
3. S. Chintapalli et al., "Benchmarking streaming computation engines: Storm, Flink and Spark Streaming," in *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 1789–1792. https://doi.org/10.1109/IPDPSW.2016.138
4. B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10)*, 2010, pp. 143–154. https://doi.org/10.1145/1807128.1807152
5. J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013. https://doi.org/10.1145/2408776.2408794
6. N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, 2003, pp. 115–130.
7. M. Zaharia et al., "Discretized streams: Fault-tolerant streaming computation at scale," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)*, 2013, pp. 423–438. https://doi.org/10.1145/2517349.2522737

8. D. G. Murray et al., "Naiad: A timely dataflow system," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)*, 2013, pp. 439–455. https://doi.org/10.1145/2517349.2522738
9. T. Akidau et al., "The Dataflow Model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015. https://doi.org/10.14778/2824032.2824076
10. D. Crankshaw et al., "Clipper: A low-latency online prediction serving system," in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*, 2017, pp. 613–627.
11. D. Crankshaw et al., "InferLine: Latency-aware provisioning and scaling for prediction serving pipelines," in *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*, 2020, pp. 477–491. https://doi.org/10.1145/3419111.3415537
12. W. Kwon et al., "Efficient memory management for large language model serving with PagedAttention," in *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*, 2023, pp. 611–626. https://doi.org/10.1145/3600006.3613165
13. Z. Zhang et al., "H2O: Heavy-hitter oracle for efficient generative inference of large language models," in *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, Curran Associates, 2023.
14. S. Henning, A. Vogel, M. Leichtfried, O. Ertl, and R. Rabiser, "ShuffleBench: A benchmark for large-scale data shuffling operations with distributed stream processing frameworks," in *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24)*, 2024, pp. 47–58. https://doi.org/10.1145/3629526.3645036
15. J. Shokhanda, U. Pal, A. Kumar, S. Chattopadhyay, and A. Bhattacharya, "SafeTail: Efficient tail latency optimization in edge service scheduling via computational redundancy management," *arXiv preprint arXiv:2408.17171*, 2024. https://doi.org/10.48550/arXiv.2408.17171