Custom Page Replacement Policy in the Linux Kernel: Frequency-Aware Hot Page Protection with Adaptive Thresholding

Gao Xinghao
225040135

Fu Yushu
225040139

***Abstract*—The Linux kernel's page reclaim subsystem relies on a two-list LRU (active/inactive) with a single reference signal to distinguish hot pages from cold ones. This mechanism is vulnerable to scan pollution: a one-time sequential scan can make cold pages look recent while genuinely hot pages are demoted too aggressively. We present a custom page replacement policy implemented directly in `mm/vmscan.c` that augments the standard LRU with per-folio access frequency tracking, recency windowing, and adaptive pressure-based thresholding. The policy hooks into `shrink_active_list` to influence active-list demotion decisions based on both frequency and recency, and exposes runtime statistics via `/proc/os_project_stats`. Evaluation with controlled trace-driven workloads confirms that hot-page protection, adaptive demotion, and frequency decay execute as designed, while the procfs interface makes policy behavior directly observable inside the reclaim path.**

***Index Terms*—Linux kernel, page replacement, virtual memory, LRU, scan pollution, vmscan, memory reclaim, folio**

I. Introduction

Virtual memory management in Linux is handled primarily by the reclaim logic in `mm/vmscan.c`. The default design uses an active list and an inactive list. Pages on the active list are assumed to be more likely to be reused, while pages on the inactive list are closer to eviction. During reclaim, `shrink_active_list()` scans active folios and decides whether to keep them active or demote them to the inactive list.

The standard decision path relies heavily on `folio_referenced()`, which provides a reclaim-visible reference signal. This is intentionally lightweight, but it is also coarse: a folio touched once during a large sequential scan can look similar to a folio that has been accessed repeatedly over a longer interval. Under scan-heavy workloads, that behavior can pollute the active list and eventually hurt the true working set.

This project explores a conservative kernel modification that keeps the Linux reclaim structure intact while adding a stronger notion of hotness. Instead of replacing the whole algorithm, we record per-folio access frequency and recency inside `vmscan.c`, then use those signals to decide whether a folio should remain active. The goal is not to build a new cache from scratch, but to improve the demotion decision already made inside the reclaim path.

II. Background

A. Linux Page Reclaim

Linux reclaim is driven by both background reclaim (`kswapd`) and direct reclaim on allocation pressure. Folios are organized into per-node LRU vectors. Within that design, `shrink_active_list()` is responsible for moving folios from the active list to the inactive list, while `shrink_folio_list()` later attempts actual reclamation from inactive folios.

This separation matters for our project: the hook point is not the final victim selection across the entire cache. It is one stage in the reclaim pipeline, specifically the active-to-inactive demotion decision.

B. Limitations of Reference-Bit LRU

The reference signal used by reclaim captures whether a folio appeared referenced since the last scan, but it does not encode access frequency. A single sequential access and many repeated accesses can therefore collapse into the same binary outcome. This is close in spirit to LRU-1.

Classic algorithms such as LRU-K [3] and ARC [4] address this by making frequency more explicit. Linux's two-list design already approximates more than pure recency, but it still does not maintain a direct per-folio frequency counter in the reclaim path.

C. Related Work

Denning's working-set model [2] motivates the use of a time window for deciding which pages are actively in use. Frequency decay is also a familiar idea in replacement research, for example in aging-style approximations and CLOCK-Pro [6]. Our design combines a small frequency counter with a recent-access window, while staying within the existing Linux reclaim framework.

III. Design

A. Goals

1) Protect genuinely hot folios from scan pollution without modifying kernel subsystems outside `vmscan.c`.
2) Adapt policy aggressiveness to memory pressure so reclaim can still make progress under scarcity.
3) Prevent historical frequency from dominating forever by using decay.
4) Expose runtime statistics so the policy path is observable.

B. Per-Folio Metadata

We introduce a lightweight tracking structure:

```
struct os_project_folio_stat {
    struct list_head list;
    struct folio *folio;
    unsigned long access_count;
    unsigned long last_access;  /* jiffies */
};
```

Each tracked folio stores a capped frequency counter and the timestamp of its most recent reclaim-visible access. Entries are stored in a global linked list protected by a spinlock. This keeps the implementation local to `vmscan.c`, although it also means lookup cost is linear in the number of tracked folios.

C. Hot Page Decision

A folio is treated as hot if both of the following hold:

$$\text{access\_count} > \theta \ \wedge \ (t_{\text{now}} - t_{\text{last}}) < \tau \tag{1}$$

Here θ is a dynamic frequency threshold and τ is a recency window implemented as `OS_PROJECT_RECENT_WINDOW = 5 * HZ`. The first term filters out one-time scan pages; the second term prevents stale historical frequency from protecting pages that are no longer active.

D. Adaptive Threshold

The threshold is selected by `get_freq_threshold()` from coarse memory pressure bands:

$$\theta = \begin{cases} 8 & \text{if pressure} > 90\% \\ 4 & \text{if pressure} > 70\% \\ 2 & \text{otherwise} \end{cases} \tag{2}$$

This design makes the policy more selective when memory pressure is high. Under light pressure, a folio needs only modest evidence to stay active. Under severe pressure, only very recent and repeatedly accessed folios should be protected.

E. Frequency Decay

After each hot-or-cold decision, the implementation decays frequency:

```
entry->access_count >>= 1;
```

This exponential decay keeps the policy responsive to changing workloads. A folio that was hot earlier but stops being accessed will gradually lose its protection.

IV. Implementation

A. Modified File

All kernel changes are confined to `mm/vmscan.c`. No header files or other memory-management subsystems were modified.

B. Initialization

Global counters and the folio-stat list are declared at file scope:

```
static atomic_long_t os_project_evict_count =
    ATOMIC_LONG_INIT(0);
static atomic_long_t os_project_hot_keep_count =
    ATOMIC_LONG_INIT(0);
static atomic_long_t os_project_demote_count =
    ATOMIC_LONG_INIT(0);
static atomic_long_t os_project_decay_count =
    ATOMIC_LONG_INIT(0);
static atomic_long_t os_project_tracked_count =
    ATOMIC_LONG_INIT(0);

static LIST_HEAD(os_project_folio_stats);
static DEFINE_SPINLOCK(os_project_folio_stats_lock);
```

Using `atomic_long_t` avoids races when counters are updated from the reclaim path.

C. Hook in `shrink_active_list`

The policy is injected into the folio scan loop. The implementation first records reclaim-visible accesses, preserves the upstream executable-file protection path, and only then applies the custom hot-page decision:

```
referenced = folio_referenced(folio, 0,
    sc->target_mem_cgroup, &vm_flags);
if (referenced > 0)
    update_page_access(folio);

if (referenced > 0 && (vm_flags & VM_EXEC) &&
    folio_is_file_lru(folio)) {
    folio_set_active(folio);
    list_add(&folio->lru, &l_active);
    continue;
}

if (os_project_should_keep_active(folio)) {
    folio_set_active(folio);
    list_add(&folio->lru, &l_active);
    atomic_long_add(folio_nr_pages(folio),
        &os_project_hot_keep_count);
    os_project_decay_access(folio);
    continue;
}

os_project_decay_access(folio);
folio_clear_active(folio);
folio_set_workingset(folio);
list_add(&folio->lru, &l_inactive);
atomic_long_add(folio_nr_pages(folio),
    &os_project_demote_count);
```

The key semantic change is that a folio is kept active only if it passes the frequency-and-recency test, not merely because its reference signal was set once during the current reclaim interval.

D. Metadata Lifetime and Eviction Counter

When reclaim succeeds, the policy increments the eviction counter and deletes the associated metadata entry:

```
atomic_long_add(nr_reclaimed,
    &os_project_evict_count);
os_project_delete_folio_stat(folio);
```

This avoids the unbounded metadata growth that would otherwise occur if entries were never removed.

E. Procfs Interface

The policy registers `/proc/os_project_stats` in `kswapd_init()`:

```
proc_create_single("os_project_stats", 0,
    NULL, os_project_stats_show);
```

Reading the proc file returns counters such as:

```
OS_PROJECT_STATS
EvictCount: 1482
HotKeepCount: 307
DemoteCount: 1175
DecayCount: 2890
TrackedCount: 512
FreqThreshold: 2
RecentWindowJiffies: 500
```

These fields are important because they make the policy observable even in cases where the final hit-rate difference is small.

F. Concurrency

All accesses to the metadata list are protected by `spin_lock_irqsave()` and `spin_lock_irqrestore()`. Allocation uses a double-checked pattern so the spinlock is not held across `kzalloc(GFP_ATOMIC)`. This keeps the implementation compatible with reclaim-path constraints while avoiding duplicate entries for the same folio.

V. Evaluation

A. Experimental Setup

We evaluated the design with controlled trace workloads consisting of three stages: repeatedly warming a hot working set, scanning a larger stream of cold pages, and immediately re-accessing the hot set. This structure aligns closely with the reclaim behavior targeted by `shrink_active_list()`, making it suitable for inspecting how the custom policy reacts to reuse, sequential pressure, and short-term working-set return.

B. Policy Activation Under Trace Workloads

During scan phases, the custom policy records hot-page protections and frequency-decay events while distinguishing frequently reused folios from one-time scan pages. The observed behavior matches the design goal: folios with strong recent-access history are preferentially kept active, while colder folios continue to flow toward the inactive list under reclaim pressure.

C. Observability Through `/proc/os_project_stats`

The evaluation also focuses on direct observability inside the reclaim path. `HotKeepCount` shows how often the policy identifies folios that should remain active, `DemoteCount` tracks continued demotion activity, and `DecayCount` confirms that the aging mechanism is applied throughout reclaim. `TrackedCount` exposes the size of the metadata set being maintained at runtime. Together, these counters provide a clear view of policy behavior during workload transitions.

D. Behavior Across Access Phases

The phase-based workload structure highlights three aspects of the design. The warm-up phase accumulates access history for the active working set. The scan phase exercises the policy under sequential pressure and triggers hot-page protection and decay. The immediate re-access phase then shows how the policy responds when the workload returns to frequently used data. Across these transitions, the recorded counters are consistent with the intended frequency-plus-recency design.

VI. Discussion

A. Comparison with Standard Linux LRU

The standard `shrink_active_list()` relies on a reclaim-visible reference signal that is close in spirit to LRU-1. Our policy adds a modest frequency counter and a recency window, making the decision more selective without replacing Linux's two-list reclaim structure. This is still far simpler than introducing a fully separate replacement subsystem.

B. Relation to ARC

ARC [4] uses ghost lists and adaptive balancing between recency and frequency. Our approach is more conservative: it keeps Linux's existing active/inactive structure and adjusts protection with a threshold rather than ghost-list feedback. The tradeoff is lower implementation complexity but also less precise adaptation.

C. Limitations

The current metadata table uses a linked list, so lookup cost remains $O(n)$ as the number of tracked folios grows. Although reclaimed folios now delete their metadata entries, the extra bookkeeping still adds overhead to the reclaim path. This makes metadata organization an important target for future optimization.

VII. Conclusion

We implemented a custom page replacement policy in the Linux kernel by modifying `mm/vmscan.c`. The policy augments the standard two-list LRU with per-folio frequency tracking, recency windowing, and adaptive pressure-based thresholding. The main contributions are:

1) A lightweight `os_project_folio_stat` structure that tracks access frequency and recency without modifying the folio itself.
2) An adaptive threshold that makes the policy more selective under higher pressure.

3) A frequency decay mechanism that prevents historical counts from permanently biasing reclaim decisions.
4) A `/proc/os_project_stats` interface for runtime observability.

The completed implementation establishes a kernel-integrated mechanism whose behavior can be inspected directly in the reclaim path. Controlled trace-based evaluation validates hot-page protection, adaptive demotion, and frequency decay, while the procfs interface makes those effects observable at runtime. Future work includes replacing the linked list with a hash table for $O(1)$ lookup and extending evaluation to broader workload families.

REFERENCES

[1] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[2] P. J. Denning, "The working set model for program behavior," *Communications of the ACM*, vol. 11, no. 5, pp. 323–333, 1968.

[3] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," *ACM SIGMOD Record*, vol. 22, no. 2, pp. 297–306, 1993.

[4] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proc. USENIX FAST*, 2003, pp. 115–130.

[5] M. Gorman, *Understanding the Linux Virtual Memory Manager*. Prentice Hall, 2004.

[6] S. Jiang, X. Zhang, and B. Chen, "CLOCK-Pro: An effective improvement of the CLOCK replacement," in *Proc. USENIX ATC*, 2005, pp. 323–336.