TECHNICAL DOSSIER // LINUX KERNEL ENGINEERING & MEMORY SUBSYSTEMS

Linux Virtual Memory Compaction Subsystem Architecture

Physical Page Frame Scanner Algorithms, Tail-Latency Elimination, and Transparent Hugepages

Published: October 30, 2026 | Distribution: Open Engineering Community | Document ID: LT-2026-B16-10

EXECUTIVE SUMMARY & MICROARCHITECTURAL SCOPE

In high-throughput databases and memory-intensive computing workloads, memory allocation latency outliers often stem from the Linux Buddy Allocator struggling with physical memory fragmentation. When order-9 (2MB Hugepage) allocations fail, the kernel invokes synchronous direct compaction, stalling application threads for hundreds of milliseconds. This whitepaper analyzes the two-scanner compaction algorithm and provides deterministic sysctl tuning strategies.

Metric / Parameter	Default Kernel Settings	Tuned Proactive Compaction	Pre-allocated HugeTLB Pool	CMA (Contiguous Mem Allocator)
Compaction Trigger	Direct on Allocation Failure	Proactive kcompactd Daemon	Zero (Pre-reserved at Boot)	Dedicated Device Pool
P99 Allocation Latency	42 ms	1.8 ms	0.08 ms (Sub-microsecond)	0.15 ms
P99.9 Latency Outlier	180 - 350 ms	8.5 ms	< 0.10 ms	< 0.25 ms
Allocation Success Rate	82% under High Load	98.5% under High Load	100% (Guaranteed)	99.9%
Key Tunable Parameter	vm.compaction_proactiveness =20	vm.compaction_proactiveness =60	vm.nr_hugepages=N	cma=size@base

1. Physical Memory Fragmentation & The Buddy System

The Linux kernel Buddy Allocator manages physical memory in power-of-two page orders (from order-0 at 4KB to order-10 at 4MB). As systems operate under heavy file I/O and process churn, physical memory becomes externally fragmented: plenty of aggregate memory remains free, but contiguous multi-page blocks are unavailable. When an application requests an order-9 page for a 2MB Transparent Hugepage (THP), the kernel triggers memory compaction.

2. The Dual-Scanner Compaction Algorithm

Memory compaction operates two pointers within each memory zone in Physical Page Frame Number (PFN) space: the migration scanner and the free scanner. The migration scanner searches from the bottom of the zone upwards for movable pages, while the free scanner searches from the top downwards for free pages. Pages are isolated and migrated from the lower scanner to the upper scanner until contiguous high-order blocks are formed.

3. Root Cause of Tail-Latency Spikes (Direct Compaction)

If kcompactd background compaction cannot keep up with high-order demand, application threads calling malloc() or mmap() are trapped in synchronous direct compaction. The calling thread must lock page lists, invalidate TLB entries across CPU cores, copy 4KB physical pages, and update page tables before returning, causing devastating 100ms+ P99.9 latency spikes in Redis, RocksDB, and JVM garbage collectors.

4. Kernel Tuning & HugeTLB Architectural Determinism

To eliminate compaction stalls, production systems tune vm.compaction_proactiveness (increasing background compaction aggressiveness from 20 to 60) and lower vm.extfrag_threshold. For mission-critical low-latency databases, deploying explicit HugeTLB pools (vm.nr_hugepages) pre-allocates contiguous 2MB or 1GB pages at system boot, bypassing runtime compaction entirely with zero latency outliers.

STRATEGIC ENGINEERING TAKEAWAYS & DEPLOYMENT CHECKLIST

- **Dual-Scanner Physics:** Migration and free scanners move toward each other to defragment physical memory zones.
- **Direct Compaction Stall Elimination:** Avoids synchronous allocation trapping by elevating proactive background compaction.
- **Proactive Compaction Tuning:** Setting vm.compaction_proactiveness=50-70 ensures continuous high-order page availability.
- **Deterministic HugeTLB:** Static boot-time reservation of 2MB/1GB hugepages provides deterministic sub-microsecond allocations.