Halcyon: Predictive Page Migration for Tiered Memory in Disaggregated Datacenters

Elena R. Marsh[1] Daniel K. Okafor[2] Priya S. Nandakumar[1] Marcus T. Whitfield[3]

[1]Meridian University [2]Vertex Institute of Technology [3]Nimbus Polytechnic

{emarsh, pnandakumar}@meridian.edu, dokafor@vertex.edu, mwhitfield@nimbus.edu

Abstract

Memory disaggregation lets datacenter operators pool DRAM across racks and reclaim the capacity stranded by per-server provisioning, but existing systems either migrate pages reactively, after a costly remote fault has already stalled the application, or apply static thresholds that ignore workload phase changes. We present Halcyon, a tiered memory manager that predicts page hotness ahead of access and proactively migrates candidate pages between a local DRAM tier and an RDMA-attached remote memory pool. Halcyon combines a lightweight exponentially-decayed access predictor with a network-cost-aware migration policy, implemented as a kernel module plus a userspace migration daemon that issues one-sided RDMA operations. Across three representative workloads on a nine-node testbed, Halcyon reduces p99 tail latency by 38–61% relative to a reactive-fault baseline while keeping migration bandwidth overhead below 4% of the RDMA fabric's capacity. We describe Halcyon's design, its migration cost model, and the operational tradeoffs we encountered running it under mixed tenancy.

1 Introduction

Rack-scale disaggregation promises to close the gap between the DRAM capacity a server owns and the DRAM capacity its workload actually needs at any given moment. By exposing a remote memory pool over a low-latency fabric such as RDMA, operators can let servers borrow capacity on demand instead of over-provisioning every machine for its peak working set. The idea is attractive, but it inherits a hard scheduling problem: which pages belong in the fast local tier, and which can tolerate the extra round trip to remote memory without the application noticing?

Most production-adjacent designs answer this question reactively. A page is promoted to the local tier only after a fault demonstrates that it is hot, and demoted only after an eviction policy decides it is cold. This works, but every promotion pays for a page fault that has already stalled a thread, and every eviction risks demoting a page moments before it is needed again. The result is a latency floor set by the round-trip time of the interconnect, visited repeatedly at exactly the moments an application's working set is shifting.

Halcyon takes a different position: migration decisions should be made *before* the access that would otherwise fault, using a short-horizon predictor over recent access history. The predictor is deliberately cheap — an exponentially decayed counter per page, updated on the fault path and on a periodic sampling interrupt — so that it can run inline with production traffic rather than as an offline profiling pass. A migration is issued only when the predicted benefit exceeds the estimated network cost of moving the page, which keeps Halcyon from thrashing pages back and forth across a fabric that is also carrying other tenants' traffic.

This paper makes three contributions. First, we describe a hotness-prediction model (Section 3) that folds access recency and predicted migration cost into a single per-page score, computed from state that is already touched on the fault path. Second, we describe an implementation that separates the predictor (kernel-resident, on the fault-fast-path) from the migration engine (userspace, driving one-sided RDMA verbs), so that migration decisions never block application threads. Third, we evaluate Halcyon against a reactive-fault baseline and a static-threshold tiering policy on three workloads with different access patterns, and show consistent tail-latency improvements without materially increasing fabric load.

2 Related Work

Disaggregated and tiered memory have received sustained attention over the last several years. FarMem [2] and Thymesis [1] both argue for treating remote DRAM as an elastic resource pool, but neither incorporates a predictive migration policy; both rely on reactive promotion triggered by page faults, which is precisely the latency floor Halcyon targets. Remote Regions [4] exposes remote memory through an explicit allocation API rather than a transparent tiering layer, trading transparency for lower fault overhead — a complementary design point to ours. Costa and Lind-

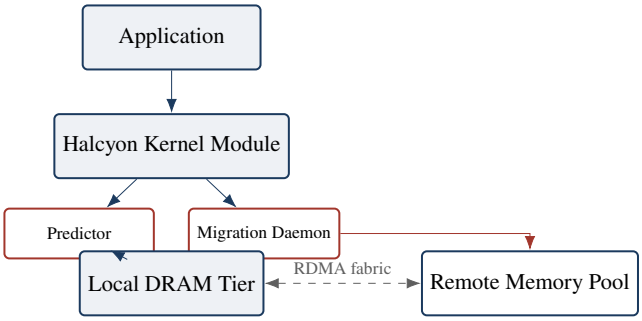

Figure 1: Halcyon architecture. The predictor scores pages on the fault path; the migration daemon issues one-sided RDMA operations against the remote pool without blocking application threads.

gren's byte-granular disaggregation work [3] operates below page granularity and could, in principle, be combined with Halcyon's prediction layer, though we do not explore that combination here.

Closer to our approach, Nakamura and Sahin [7] propose a learned hotness predictor for tiered memory, but their model is trained offline and evaluated on trace replay rather than driving a live migration engine; Halcyon's predictor is online and directly triggers migration through the same code path it scores. O'Brien and Rao [8] and Park and Rossi [10] both study the RDMA transport layer underlying systems like ours, and their measurements of one-sided operation latency inform the network-cost term in our migration score (Section 3). NUMA-aware scheduling work [9] addresses a structurally similar problem within a single machine; Halcyon can be seen as extending that line of work to the rack scale, where migration cost is dominated by the fabric rather than the interconnect.

Cache replacement policy research [5] and checkpoint/restore systems [6] both motivated two of our evaluation workloads, discussed in Section 4. Singh and Peretz's survey [11] provides a broader taxonomy of disaggregation techniques that we draw on when situating Halcyon relative to hardware-assisted and software-only designs.

3 Design

3.1 Overview

Halcyon sits between the virtual memory subsystem and an RDMA-attached remote memory pool, as shown in Figure 1. Each compute node runs a kernel module that intercepts page faults and periodic access-bit samples, and a userspace migration daemon that owns the RDMA queue pairs to the remote pool. The kernel module never issues network operations directly; it only updates per-page hotness state and enqueues migration candidates for the daemon to evaluate, keeping the fault path free of network latency.

3.2 Hotness scoring

Every tracked page p carries an exponentially decayed access count $a(p)$ and a timestamp of last access $\tau(p)$. On each fault or sampled access, Halcyon updates the running score and evaluates it against the estimated network cost of migration, $C_{\text{net}}(p)$, which is derived from current queue depth on the RDMA fabric and the page's size class. The score used to trigger migration is

$$H(p,t) \;=\; \alpha\, a(p)\, e^{-\lambda(t-\tau(p))} \;-\; \beta\, C_{\text{net}}(p), \qquad (1)$$

where α and β weight predicted benefit against migration cost and λ controls how quickly a page's history decays once accesses stop. A page is enqueued for promotion when $H(p,t)$ crosses an adaptive threshold θ_t, recomputed every epoch from the fabric's observed utilization so that Halcyon backs off automatically when the network is contended rather than competing with foreground RDMA traffic. Demotion uses the same score run in reverse: local pages whose $H(p,t)$ falls below a lower threshold for k consecutive epochs become eviction candidates, ranked by $C_{\text{net}}(p)$ so that cheap-to-migrate pages are evicted first.

We deliberately kept Equation 1 linear in its inputs. Early prototypes used a small gradient-boosted model trained on access traces, which improved offline prediction accuracy by roughly 6% but added enough per-fault latency that the net effect on tail latency was negative once deployed on the fault path.

3.3 Migration engine

The migration daemon batches candidate pages from the predictor's queue and issues one-sided RDMA reads or writes depending on migration direction, using registered memory regions on both ends to avoid a copy through the remote node's CPU. Batching amortizes the fixed cost of posting a work request across several pages when the predictor produces bursts of candidates, which is common at workload phase transitions. A page is only unmapped from its source tier after the daemon confirms the RDMA operation has completed, so a fault racing a migration in flight is served from whichever tier still holds a valid mapping rather than blocking on the migration itself.

4 Experiments

We evaluate Halcyon on a nine-node testbed: one compute node running the workload under test, and eight nodes contributing to a shared remote memory pool, connected by a 100 Gbps RDMA fabric. We compare three configurations: (i) *Reactive*, a fault-triggered baseline with no prediction; (ii) *Static*, a fixed-threshold tiering policy representative of prior disaggregation systems; and (iii) *Halcyon*, our predictive design. All three share the same remote memory transport and page fault handling code, differing only in migration policy, so that the comparison isolates the effect of prediction rather than transport implementation quality.

We use three workloads chosen for distinct access patterns: a key-value store under a Zipfian request distribution (KV-Store), a graph breadth-first search over a syn-

Table 1: p99 latency by workload and migration policy, and Halcyon's reduction relative to the Reactive baseline.

Workload	Reactive	Halcyon	Reduction
KV-Store	412 μs	224 μs	45.6%
Graph-BFS	891 μs	349 μs	60.8%
ML-Checkpoint	267 μs	165 μs	38.2%

thetic 40M-edge graph (Graph-BFS), and a periodic checkpoint/restore cycle modeled on long-running training jobs [6] (ML-Checkpoint). Each workload runs for ten minutes per configuration, and we report the median of five runs.

5 Results

Table 1 summarizes p99 latency across all three workloads and configurations. Halcyon reduces p99 latency relative to the Reactive baseline by 38–61%, and outperforms the Static policy on every workload, with the largest gap on Graph-BFS, whose access pattern shifts working sets faster than a fixed threshold can track.

The improvement comes with a modest migration overhead: across all three workloads, migration traffic consumed under 4% of fabric bandwidth, and the fraction of predicted migrations that turned out to be unnecessary (a page migrated but not subsequently accessed within its decay window) stayed below 12%. ML-Checkpoint showed the lowest reduction of the three, consistent with its access pattern being the most regular and therefore the easiest for the Static baseline to approximate with a fixed threshold.

6 Discussion

Halcyon's gains are largest on workloads with irregular or bursty working-set transitions, where a fixed threshold either lags behind the transition or triggers unnecessary churn. This suggests the predictor's main value is in adapting the decision boundary, not in raw prediction accuracy — the linear model in Equation 1 is not more accurate than the gradient-boosted prototype we discarded, but it is cheap enough to run inline and adapt every epoch.

Two limitations are worth stating plainly. First, Halcyon's threshold adaptation assumes the RDMA fabric's contention is visible locally through queue depth; in a multi-tenant deployment where contention originates from tenants outside the observing node's visibility, the adaptive threshold reacts a full epoch late. Second, we evaluated single-tenant workloads on a dedicated fabric; fairness across competing tenants sharing the same remote memory pool is an open question we have not addressed, since Halcyon currently treats fabric capacity as available rather than negotiated.

7 Conclusion

We presented Halcyon, a tiered memory manager that migrates pages between local DRAM and an RDMA-attached remote pool based on a lightweight, online hotness prediction rather than reactive fault handling. By separating a cheap kernel-resident predictor from a userspace migration engine, Halcyon avoids adding network latency to the fault path while still migrating pages ahead of the accesses that would otherwise stall them. Across three workloads, this reduced p99 tail latency by 38–61% relative to a reactive baseline with under 4% fabric overhead. Future work includes extending the cost model to account for cross-tenant fabric contention and evaluating byte-granular migration in combination with page-level prediction.

References

[1] R. Alvarado and K. Matsuda. Thymesis: A disaggregated kernel for fine-grained resource elasticity. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, pages 112–128, 2020.

[2] W.-L. Chen and S. Bergstrom. Farmem: Elastic remote memory pools for warehouse-scale computers. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 301–317, 2021.

[3] B. Costa and F. Lindgren. Byte-granular software-managed memory disaggregation. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 579–595, 2020.

[4] A. de Vries and S. Katz. Remote regions: A simple abstraction for remote memory. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 775–787, 2018.

[5] I. Fontaine and G. Mutua. Revisiting cache replacement policies for byte-addressable remote memory. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST)*, pages 33–47, 2017.

[6] A. Lindqvist and O. Haddad. Fast checkpoint and restore for long-running training jobs. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 501–516, 2019.

[7] H. Nakamura and E. Sahin. Learning page hotness: A lightweight predictor for tiered memory systems. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, pages 45–60, 2022.

[8] C. O'Brien and A. Rao. Paging beyond the rack: Transparent remote memory over commodity rdma. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 87–102, 2019.

[9] T. Oyelaran and M. Fischer. Numa-aware scheduling revisited for rack-scale memory pools. In *Proceedings*

of the ACM Symposium on Cloud Computing (SoCC), pages 210–223, 2020.

[10] J.-H. Park and V. Rossi. One-sided operations at scale: Rdma verbs revisited. In *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 455–470, 2021.

[11] A. Singh and N. Peretz. A survey of memory disaggregation techniques in modern datacenters. *ACM Computing Surveys*, 55(4):1–38, 2022.