Recent studies has shown a reinvigorated interest in disaggregted/distributed shared memory systems since the 1990s. While large-scale cluster systems predominantly make up the mainstream The interplay between (page) replacement policy and runtime performance of distributed shared memory systems has not been properly explored.

1 Overview of Distributed Shared Memory

A striking feature in the study of distributed shared memory (DSM) systems is the non-uniformity of the terminologies used to describe overlapping study interests. The majority of contributions to DSM study come from the 1990s, for example [**Treadmark, Millipede, Munin, Shiva, etc.**]. These DSM systems attempt to leverage kernel system calls to allow for user-level DSM over ethernet NICs. While these systems provide a strong theoretical basis for today's majority-software DSM systems and applications that expose a *(partitioned) global address space*, they were nevertheless constrained by the limitations in NIC transfer rate and bandwidth, and the concept of DSM failed to take off (relative to cluster computing).

Improvement in NIC bandwidth and transfer rate allows for applications that expose global address space, as well as RDMA technologies that leverage single-writer protocols over hierarchical memory nodes. [**GAS and PGAS (Partitioned GAS) technologies for example Openshmem, OpenMPI, Cray Chapel, etc. that leverage specially-linked memory sections and `/dev/shm` to abstract away RDMA access**].

Contemporary works on DSM systems focus more on leveraging hardware advancements to provide fast and/or seamless software support. Adrias [4], for example, implements a complex system for memory disaggregation over multiple compute nodes connected via the *ThymesisFlow*-based RDMA fabric, where they observed significant performance improvements over existing data-intensive processing frameworks, for example APACHE Spark, Memcached, and Redis, over no-disaggregation (i.e., using node-local memory only, similar to cluster computing) systems.

1.1 Move Data to Process, or Move Process to Data?

(TBD – The former is costly for data-intensive computation, but the latter may be impossible for certain tasks, and greatly hardens the replacement problem.)

2 Replacement Policy

In general, three variants of replacement strategies have been proposed for either generic cache block replacement problems, or specific use-cases where contextual factors can facilitate more efficient cache resource allocation:

- General-Purpose Replacement Algorithms, for example LRU.

- Cost-Model Analysis
- Probabilistic and Learned Algorithms

2.1 General-Purpose Replacement Algorithms

Practically speaking, in the general case of the cache replacement problem, we desire to predict the re-reference interval of a cache block [2]. This follows from the Belady's algorithm – the optimal case for the *ideal* replacement problem occurs when, at eviction time, the entry with the highest re-reference interval is replaced. Under this framework, therefore, the commonly-used LRU algorithm could be seen as a heuristic where the re-reference interval for each entry is predicted to be immediate. Fortunately, memory access traces of real computer systems agree with this tendency due to spatial locality [**source**]. (Real systems are complex, however, and there are other behaviors...) On the other hand, the hypothetical LFU algorithm is a heuristic that captures frequency. [**...**] While the textbook LFU algorithm suffers from needing to maintain a priority-queue for frequency analysis, it was nevertheless useful for keeping recurrent (though non-recent) blocks from being evicted from the cache [**source**].

Derivatives from the LRU algorithm attempts to balance between frequency and recency. [**Talk about LRU-K, LRU-2Q, LRU-MQ, LIRS, ARC here ...**]

Advancements in parallel/concurrent systems had led to a rediscovery of the benefits of using FIFO-derived replacement policies over their LRU/LFU counterparts, as book-keeping operations on the uniform LRU/LFU state proves to be (1) difficult for synchronization and, relatedly, (2) cache-unfriendly [5]. [**Talk about FIFO, FIFO-CLOCK, FIFO-CAR, FIFO-QuickDemotion, and Dueling CLOCK here ...**]

Finally, real-life experiences have shown the need to reduce CPU time in practical applications, owing from one simple observation – during the fetch-execution cycle, all processors perform blocking I/O on the memory. A cache-unfriendly design, despite its hypothetical optimality, could nevertheless degrade the performance of a system during low-memory situations. In fact, this proves to be the driving motivation behind Linux's transition away from the old LRU-2Q page replacement algorithm into the more coarse-grained Multi-generation LRU algorithm, which has been mainlined since v6.1.

2.2 Cost-Model Analysis

The ideal case for the replacement problem fails to account for invalidation of cache entries. It also assumes for a uniform, dual-hierarchical cache-store model that is insufficient to capture the heterogeneity of today's massively-parallel, distributed systems. High-speed network interfaces are capable of exposing RDMA interfaces between computer nodes, which amount to almost twice as fast RDMA transfer when compared to swapping over the kernel I/O stack, while software that bypass the kernel I/O stack is capable of stretching the

bandwidth advantage even more (source). This creates an interesting network topology between RDMA-enabled nodes, where, in addition to swapping at low-memory situations, the node may opt to "swap" or simply drop the physical page in order to lessen the cost of page misses.

[Talk about GreedyDual, GDSF, BCL, Amortization]

Traditionally, replacement policies based on cost-model analysis were utilized in content-delivery networks, which had different consistency models compared to finer-grained systems. HTTP servers need not pertain to strong consistency models, as out-of-date information is considered permissible, and single-writer scenarios are common. Consequently, most replacement policies for static content servers, while making strong distinction towards network topology, fails to concern for the cases where an entry might become invalidated, let along multi-writer protocols. One early paper [3] examines the efficacy of using page fault frequency as an indicator of preference towards working set inclusion (which I personally think is highly flawed – to be explained). Another paper [1] explores the possibility of taking page fault into consideration for eviction, but fails to go beyond the obvious implication that pages that have been faulted *must* be evicted.

The concept of cost models for RDMA and NUMA systems are relatively underdeveloped, too. (Expand)

2.3 Probabilistic and Learned Algorithms for Cache Replacement

Finally, machine learning techniques and low-cost probabilistic approaches have been applied on the ideal cache replacement problem with some level of success. **[Talk about LeCaR, CACHEUS here]**.

3 Cache Coherence and Consistency in DSM Systems

(I need to read more into this. Most of the contribution comes from CPU caches, less so for DSM systems.) **[Talk about JIAJIA and Treadmark's coherence protocol.]**

Consistency and communication protocols naturally affect the cost for each faulted memory access ...

[Talk about directory, transactional, scope, and library cache coherence, which allow for multi-casted communications at page fault but all with different levels of book-keeping.]

References

[1] J. Aguilar and E.L. Leiss. "A Coherence-Replacement Protocol For Web Proxy Cache Systems". In: *International Journal of Computers and Applications* 28.1 (2006), pp. 12–18. DOI: 10.1080/1206212X.2006.11441783. eprint: https://doi.org/10.1080/1206212X.2006.11441783. URL: https://doi.org/10.1080/1206212X.2006.11441783.

[2] Aamer Jaleel et al. "High performance cache replacement using re-reference interval prediction (RRIP)". In: *ACM SIGARCH computer architecture news* 38.3 (2010), pp. 60–71.

[3] Richard P. LaRowe and Carla Schlatter Ellis. "Page placement policies for NUMA multiprocessors". In: *Journal of Parallel and Distributed Computing* 11.2 (1991), pp. 112–129. ISSN: 0743-7315. DOI: https://doi.org/10.1016/0743-7315(91)90117-R. URL: https://www.sciencedirect.com/science/article/pii/074373159190117R.

[4] Dimosthenis Masouros et al. "Adrias: Interference-Aware Memory Orchestration for Disaggregated Cloud Infrastructures". In: *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE. 2023, pp. 855–869.

[5] Juncheng Yang et al. "FIFO can be Better than LRU: the Power of Lazy Promotion and Quick Demotion". In: *Proceedings of the 19th Workshop on Hot Topics in Operating Systems*. 2023, pp. 70–79.