WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Low-Level Memory Management in Scalable Distributed Architectures: Approaches to Improving Reliability and Performance of Digital Services

Terletska Khrystyna [a]

[a] *Lviv Polytechnic National University, 79013, Ukraine, Lviv*

A B S T R A C T

This paper examines modern theoretical approaches to low-level memory management in scalable distributed architectures. It provides an analysis of memory allocation strategies, including slab allocators, region-based and stack-based methods, as well as NUMA-aware techniques. Special attention is paid to the impact of data structures and algorithms on the performance of digital services and to memory optimization under constrained resource environments. The study includes practical examples from RocksDB and LMDB. The findings highlight the potential of these approaches in enhancing the reliability and resilience of distributed computing systems.

Keywords: low-level memory management, NUMA, distributed systems, slab allocator, performance.1.

1. Introduction

Low-level memory management is one of the critical factors that determine the efficiency and reliability of modern digital services in distributed, scalable architectures. As information systems become more complex and the volume of data processed grows, the need to find new approaches to optimal memory allocation and use becomes more urgent. This is due to the fact that the efficiency of memory management algorithms depends on the performance of applications, their resistance to failures and the ability to serve a large number of parallel requests, which is especially important in the context of constantly growing requirements for the fault tolerance of digital services.

The aim of this study is to provide a theoretical and comparative analysis of modern low-level memory management approaches used in scalable distributed architectures, as well as to assess the impact of these approaches on the reliability and performance of digital services. The author sets out to review and compare existing memory management strategies, identify their advantages and disadvantages, and identify promising areas of development in this area.

The paper discusses in detail the basic theoretical principles and algorithms of low-level memory management, approaches to optimizing its use in the NUMA architecture, algorithmic solutions for environments with limited memory, as well as approaches to memory optimization in distributed data warehouses. The result of the study will be an assessment of modern approaches with the identification of their practical significance and development prospects, which can serve as a basis for further scientific developments.

2. Main part. Theoretical foundations and approaches to low-level memory management

The efficiency of memory management in distributed architectures largely depends on the selected memory allocation and deallocation strategies. Among these strategies, there are approaches using slab allocators, which allow minimizing fragmentation and speeding up memory allocation by pre-reserving fixed-size areas. Slab allocators are especially effective in conditions of high intensity of access to memory objects, for example, when serving a large number of short-term requests [1].

Another important approach is the use of regional memory allocation (region-based allocation), which involves group allocation and subsequent release of memory blocks by entire regions. This approach improves the performance of distributed applications by reducing memory allocation and cleanup overhead and improving data locality. Despite its high performance, the regional approach requires careful planning of the size of regions to avoid excessive memory consumption and possible memory overflow in the long run.

Along with the regional approach, memory allocation on the stack is widely used in practice. The stack provides fast access to data through a simple mechanism for allocating and freeing memory based on the LIFO principle («last in, first out»). This approach has extremely low overhead and virtually

no fragmentation, but its application is limited by the short data lifetime and fixed size of allocated memory blocks [2]. Stack memory allocation is suitable for local computing and short-term storage of intermediate data in high-performance computing, but it is inefficient for long-term storage or dynamically changing data structures.

Additionally, it is worth noting that a common approach in the development of modern digital services is microservice architecture, especially when designing user interfaces. Separating complex front-end applications into separate microservices allows for more flexible memory management by isolating the resources of each service, thereby reducing the risks of excessive consumption of shared memory. This approach not only increases the reliability of individual system components, but also provides simplified monitoring and scaling of services depending on the current load on the interfaces [3].

2.1 Impact of data structures and algorithms on the performance of digital services

The performance of distributed digital services directly depends on how efficiently data structures and algorithms that work with low-level memory are organized. One of the most important aspects here is ensuring data locality (data locality), that is, placing data in such a way as to maximize the caching capabilities of the central processor. Proper use of data locality reduces the number of accesses to main memory and reduces the latency of read-write operations.

Along with data locality, so-called cache-friendly structures have a significant impact on performance. Examples of such structures are continuous element placement arrays and aligned data structures that minimize cache misses and efficiently use the processor's cache lines. The use of cache-friendly structures can significantly reduce the execution time of algorithms due to more efficient use of the hardware capabilities of modern processors.

Another important area of performance optimization is the implementation of data prefetching mechanisms prefetching. This approach involves loading data into the processor cache before the algorithm actually needs it, thus preventing waiting for subsequent read operations. However, excessively aggressive prefetching can, on the contrary, reduce the efficiency of work due to excessive consumption of memory bandwidth, so it requires careful configuration and adaptive management depending on the type of load.

In the context of modern approaches to the development of digital services, the use of reactive programming, in particular the use of tools such as RxJS and NgRx, is of particular importance [4]. These libraries allow you to efficiently manage asynchronous event flows, reducing memory load and ensuring high responsiveness of interfaces.

2.2 Features of memory management in NUMA architectures

Architectures based on NUMA are widely adopted in modern multi-node systems, as they enable the scaling of computational resources by distributing memory across different nodes. However, in such systems, there are specific problems associated with inter-node delays due to different processor access times to local and remote memory.

One of the most common methods for reducing the negative impact of inter-node delays is the first-touch approach, in which a memory page is assigned to the node whose processor first accessed this page. This approach allows you to localize memory and minimize the number of remote accesses. In practice, first-touch allocation can reduce memory access time compared to random page allocation in NUMA systems (fig. 1).

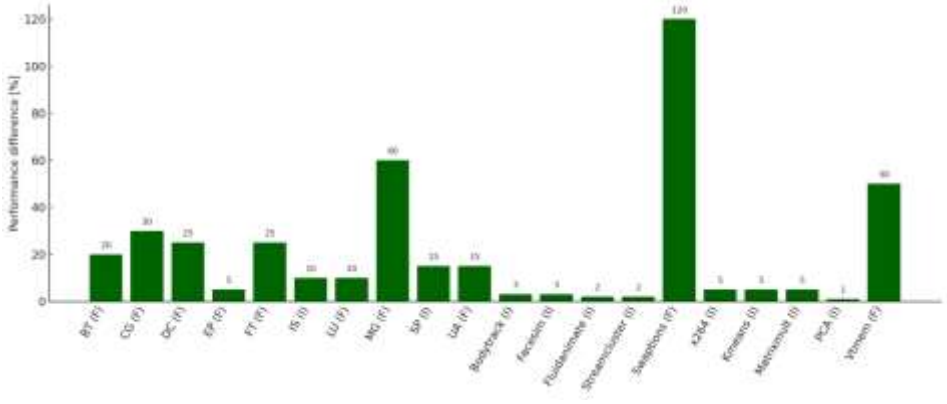

Fig. 1 - First-touch versus interleave differences for multithreaded applications [5]

However, this method requires detailed load planning and monitoring, as incorrect initial handling can lead to uneven load distribution and subsequent delays. Another approach that has become widespread is affinity-based scheduling, in which tasks are distributed to processors in such a way as to minimize inter-node requests. This approach assumes that tasks that intensively use certain areas of memory are assigned to specific nodes that are directly connected to these areas. At the same time, excessively rigid assignment of tasks can reduce the overall flexibility of the system and complicate its dynamic scaling, so you need to use a balanced approach and monitor the distribution of tasks in real time.

2.3 Algorithmic methods for memory-constrained environments

Modern distributed systems often operate under strict limits on available memory, which requires the use of special algorithmic approaches. One of the most effective methods is external memory algorithms, which are designed for processing data that significantly exceeds the size of RAM. These algorithms minimize the number of accesses to slow external storage devices, such as hard drives and SSD, ensuring an optimal balance between access speed and RAM usage.

Another promising area is in-place computing, which does not require additional memory over and above the input data already used. Such algorithms are extremely relevant for environments with a limited amount of RAM, as they allow you to save resources by reallocating and reusing already allocated memory. However, applying in-place calculations is not always convenient, as it requires careful design and can complicate the structure of the program code, especially when implementing complex data processing algorithms.

It is also worth highlighting approaches related to efficient data encoding. The use of compressed data representations and encodings can significantly reduce the amount of data stored and processed without losing information. Using, for example, bit representations and data compression methods (such as run-length encoding or Lexico) can поддерживатьsupport 90-95% of the original performance while using only 15-25% of the full key-value cache in large language models. This reduces memory usage by up to 75% with minimal impact on performance [6].

2.4 Memory management in distributed storage systems

Distributed data storage systems are now actively used to support high-load digital services, in which the reliability and speed of data access become critical parameters. These solutions include the popular RocksDB and LMDB systems, which provide efficient data storage and processing under heavy request flows. One of the key tasks when working with such systems is to optimize recording, which allows you to increase throughput and reduce data processing delays (table 1).

Table 1 - Memory management issues under high load in distributed storage systems

Problem	Cause	Manifestation	Potential solution
Frequent data flushes to disk	MemTable overflow	Write latency spikes	Increase MemTable size / tune compaction
Increasing memory consumption	Uncollected SST files	Background memory leaks	Adjust TTL / implement background GC
Read latency	Insufficient cache	Frequent disk access	Increase BlockCache size
Memory fragmentation	Repeated dynamic allocations	Lower throughput, uneven memory usage	Use slab allocators / defragmentation logic
NUMA performance degradation	Non-local memory access	Cross-node latency, CPU stall	Apply first-touch policy / bind thread affinity

For example, RocksDB actively uses the mechanism of LSM-trees (Log-Structured Merge-Trees), which optimize writing by accumulating data in RAM and then sequentially writing it to disk in large chunks. This strategy reduces the load on the I / O subsystem and provides an increase in data write speed compared to traditional systems that do not use the LSM approach [7]. However, this approach requires proper memory management, since uncontrolled accumulation of data in memory can lead to a decrease in overall system performance due to the need for frequent flushes of accumulated data to disk.

Particular attention is paid to performance optimization issues in the microservices architecture used in fintech projects, where transaction and query processing requires minimal delays and high stability [8]. For such systems, it is especially important to apply effective strategies for working with memory and data, which can reduce the latency of inter-service interaction and optimize data routing.

At the same time, the LMDB system is focused on reading data as quickly as possible and effectively uses the memory-mapped files mechanism, so it achieves high record sampling speed with minimal RAM consumption. However, this approach limits performance for a large number of write operations, as it requires additional work with mapped memory areas and associated pages.

3. Conclusion

The analysis shows that low-level memory management plays a key role in ensuring the reliability and high performance of digital services in scalable distributed architectures. The considered theoretical approaches-from slab allocators and regional memory allocation to NUMA-oriented strategies and algorithmic solutions for limited environments-demonstrate significant potential for optimizing the performance of computing systems.

Particular attention should be paid to the choice of data structures and organization of memory access, since these aspects largely determine the effectiveness of caching and interaction between nodes. Hybrid strategies that combine the advantages of different memory management models remain

promising. In addition, further research can be directed to the development of adaptive algorithms that can change the strategy of working with memory in real time, depending on the nature of the load and system architecture.

References

1. Momeu, M., Kilger, F., Roemheld, C., Schnückel S., Proskurin, S., Polychronakis, M., & Kemerlis, V.P. (2024). Islab: Immutable memory management metadata for commodity operating system kernels, *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 1159-1172.
2. Duraisamy, P., Xu, W., Hare, S., Rajwar, R., Culler, D., Xu, Z., Fan, J., Kennely, C., McCloskey, B., Mijailovic, D., Morris, B., Mukherjee, C., Ren, J., Thelen, G., Turner, P., Villavieja, C., Ranganathan, P., & Vahdat, A. (2022). Towards an adaptable systems architecture for memory tiering at warehouse-scale, *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 3*, 727-741.
3. Dudak, A. (2024). Microservice architecture in frontend development, *Norwegian Journal of development of the International Science, 145*, 99-102.
4. Garifullin, R. (2024). Application of RxJS and NgRx for reactive programming in industrial web development, *Int. J. of Professional Science, 12-2*, 42–47.
5. Gaud, F., Lepers, B., Funston, J., Dashti, M., Fedorova, A., Quema, V., Lachaize, R., & Roth, M. (2015). Challenges of Memory Management on Modern NUMA System: Optimizing NUMA systems applications with Carrefour, *Queue, 58(12)*, 59-66.
6. Kim, J., Park, J., Cho, J., & Papailiopoulos, D. (2024). Lexico: Extreme KV Cache Compression via Sparse Coding over Universal Dictionaries, *Preprint arXiv:2412.08890.*
7. RocksDB Tuning Guide, https://github.com/facebook/rocksdb/wiki/RocksDB-Tuning-Guide
8. Bolgov, S. (2025). Optimizing microservices architecture performance in fintech projects, *Bulletin of the Voronezh Institute of High Technologies, 19(1).*