Enhancing Computing Performance through Advanced Memory Allocation Techniques

Jiankun Dong CS575

Abstraction

This paper explores several innovative memory allocation algorithms designed to enhance the performance of complex systems, especially in large scale data processing environments or complex server setups. These advanced methods promise significant improvements over traditional techniques by integrating machine learning and context-aware strategies. A comparative analysis identifies vital principles that can be adopted to refine memory management practices. The exploration includes a critical review of current research and its application in real-world scenarios.

Introduction

The evolution of computing systems demands increasingly sophisticated memory management strategies to handle growing data volumes and complex applications efficiently. Traditional memory allocation techniques introduced in the textbook often fall short in such demanding environments, necessitating environment-specific or even task-specific solutions. This research focuses on innovative memory allocation strategies incorporating machine learning algorithms to optimize resource distribution dynamically. The aim is to compare these modern approaches

against conventional methods, highlighting system performance and operational efficiency improvements.

Based Method of Malloc: A Brief Review

Before we dive into the novel and innovative research in memory allocation algorithms, let's review the basic malloc introduced in the textbook and its shortcomings. To address the external fragmentation issue, malloc was designed to work using a paging method. However, page sizes has been steadily increasing, while more cores are added, creating slowdowns though contentions, 'pollution' of cache, and internal fragmentations of pages. Such drawbacks can lead to prominent performance issues depending on the environment and tasks. Following are some examples addressing said issues.

Nextgen-Malloc: Dedicated Core for Memory Management

One proposal for machine learning applications is to offload the memory allocation task to a dedicated core, Nextgen-Malloc[1]. The authors aim to address the "memory wall" issue, which is the disparity between processor speed and memory access time. Due to the memory intensive nature of machine learning applications, the better fitting memory allocator can increase performance by up to 72%, despite only 2% of execution time being spent on malloc and free calls. The paper explained that with increase in thread counts, the number of misses of the LLC(Last Level Cache) and dTLB (data Translation Lookaside Table) loads becomes significant, taking up to half of CPU cycle for warehouse-scale systems. [3] Morden UMA falls short with multithreading tasks due to the overhead of cross-core communication and the cache pollution [2]. Nextgen-Malloc proposes to have a delicate core for memory allocation tasks.

From the hardware perspective, Nextgen-Malloc can implement an idle core (or the core closest to memory, for energy efficiency and reduced latency) as the specialized memory allocation core. To minimize the inter-core communication and contention, memory architecture should also be redesigned to decouple the metadata and data, so that it operates independently to the user's application data, thus reducing cache pollution. Inter-core communication should also be improved, better to handle the "fine-grained" calls of malloc () and free(). As most memory allocations are short-live (5μs)[4], reducing inter-core communication cost is paramount to the success of dedicated-core strategy for memory allocator.

From the software standpoint, the paper proposes the following improvements: decoupling allocation from the application; reducing synchronization overhead through elimination of contention; and adaptation to different cores and computing environments. Decoupling includes redesigning the allocator's functions to operate independently and manage its metadata without interfering with the application's execution. Reducing overhead is achieved naturally by executing malloc with a single core, theoretically reducing contention.

The proposed solution shows promise: it improves execution time by 4.51% compared to Mimalloc and significantly reduces misses per kilo instructions by 43.33%.

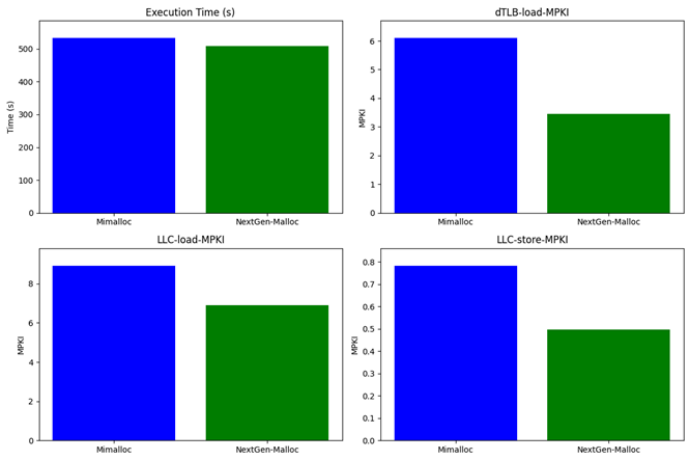

Figure 1 - Nextgen-Malloc Performance vs Mimalloc [1]

Nextgen-Malloc introduces a dedicated core for memory tasks, effectively addressing the "memory wall" by minimizing cross-core communication and cache pollution. This approach not only boosts operational efficiency but also suggests improvements in system architecture that could influence future standards.

Llama: Learning Based Malloc

As mentioned before, the growing page size can be expected in the current systems. For large-scaled C++ servers, each item in memory can have wildly different lifespan. Combining the two, the internal fragmentation issue has become increasingly prevalent, leading to a thirst of huge heap space for tracking the pages, and a great area to tackle for improving server performance. Llama proposes usage of a neural network model for prediction of lifetimes of memory items, providing a possible algorithm for consolidating fragments in memory, in order to effectively reduce the page size and consolidate under-utilized pages.

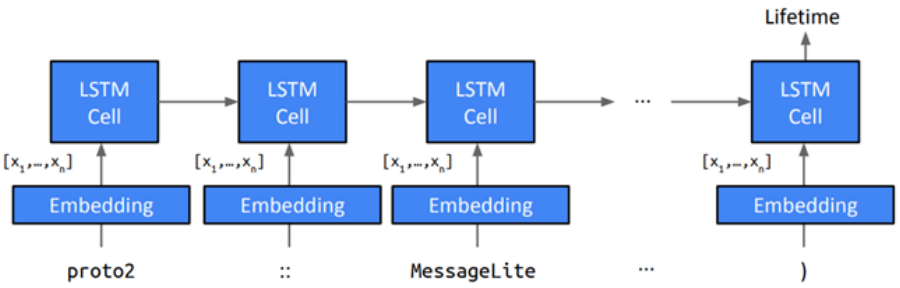

Figure 2 - Neural Network Model for Llama [5]

Llama uses a LTSM-based model to predict the lifetime of a memory item, which is typically used for prediction of the next word given a sentence. Such a model is trained on a sample server model that performs similar tasks to the targeted system.

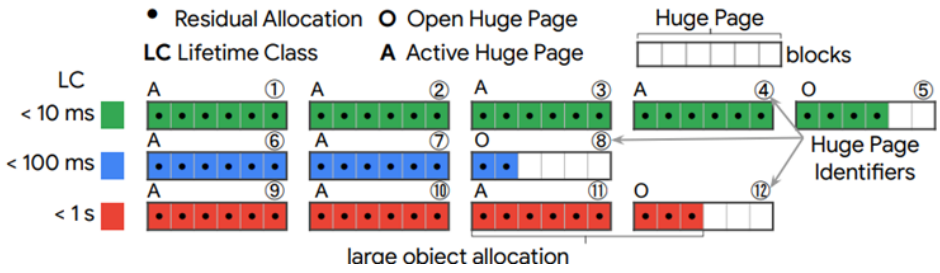

Figure 3 - Object Allocated and Page Segmentation for Llama [5]

Memory items, when allocated into memory, are split into categories based on predicted lifetime (<10ms, <100ms and <1s)[5]. Llama also segregates each page into blacks, and tracks the availability of each block. When blocks become available and pages freed to OS, it moves items in memory from shorter lifetime section to longer lifetime section, expecting the page not to expire before the moved item does. In this sense it consolidates pages and reduces internal fragmentation. To make the system automatically handle misprediction, which inevitably will

happen, Llama would drastically increase the lifetime class of misclassified items. Thus the system corrects itself and adjusts its predictions overtime.

There are also prominent drawbacks of using Llama. Due to the nature of malloc, allocation and adjustment anticipate quick response, often less than 100 cycles. Researchers addressed this by placing a set of predicted time in the cache. When a miss happens, a prediction is then made using the model, which takes hundreds of μs to complete, and cache the prediction for future use. The paper also delves into the challenges of integrating machine learning into system-level solutions, discussing issues such as the overhead of continuous profiling and the accuracy of lifetime predictions. The authors address these with innovative solutions like sampling-based data collection and a caching mechanism to reduce the overhead of machine learning predictions.[5]

Despite the challenges, Llama presents a significant forward step in memory allocation. When evaluated against large production code bases, the paper reports up to 78% reduction in memory fragmentation [5] in production environments by effectively managing memory allocation through lifetime predictions rather than just object sizes. Llama's design allows it to use huge pages efficiently, subdivide them into blocks and lines, and manage them based on the predicted and observed lifetimes, significantly mitigating fragmentation issues.

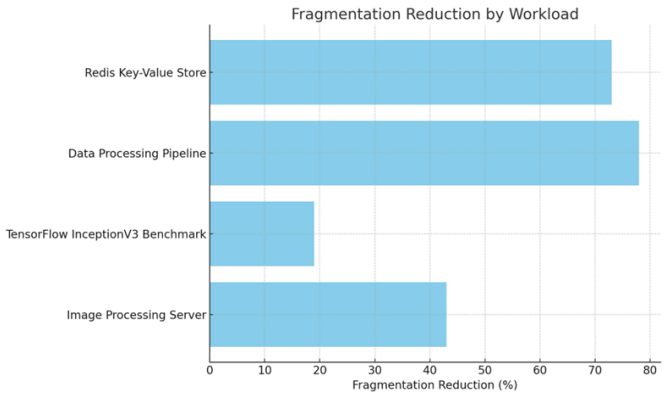

Figure 4 - Llama Performance vs Workload [5]

Llama employs machine learning to predict memory item lifetimes, dynamically adjusting allocations to reduce internal fragmentation in large-scale server environments. This adaptive method improves memory resource utilization and incorporates a self-correcting mechanism that adapts to prediction errors, enhancing overall system performance.

Conclusion

This paper takes notes of pivotal advancements in memory allocation techniques, shifting from traditional methods towards innovative strategies that enhance performance. By integrating machine learning and context-aware algorithms, the proposed solutions—Nextgen-Malloc and Llama—show significant potential to surpass the limitations of conventional memory management.

The comparative analysis underscores these advanced techniques' superiority over traditional methods, setting new benchmarks for memory management research. As computing demands escalate, memory management strategies must evolve, harnessing modern technology to create more resilient, efficient, and adaptive computing environments. This research provides a

foundational framework for future advancements, proposing sophisticated, performance-oriented memory management solutions that could redefine operational efficiencies across computing domains.

Reference

Credit and Disclaimer:

All figures presented in this paper are from the referenced works and have been referenced in the caption.

[1] Ruihao Li, Qinzhe Wu, Krishna Kavi, Gayatri Mehta, Neeraja J. Yadwadkar, and Lizy K. John. 2023. NextGen-Malloc: Giving Memory Allocator Its Own Room in the House. In Proceedings of the 19th Workshop on Hot Topics in Operating Systems (HOTOS '23). Association for Computing Machinery, New York, NY, USA, 135–142. https://doi.org/10.1145/3593856.3595911

[2] Hermann Schweizer, Maciej Besta, and Torsten Hoefler. 2015. Evaluating the cost of atomic operations on modern architectures. In 2015 International Conference on Parallel Architecture and Compilation (PACT). IEEE, 445–456.

https://doi.org/10.48550/arXiv.2010.09852

[3] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a warehouse-scale computer. In 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA). 158–169. https://doi.org/10.1145/2749469.2750392

[4] Svilen Kanev, Sam Likun Xi, Gu-Yeon Wei, and David Brooks. 2017. Mallacc: Accelerating Memory Allocation. In Proceedings of the TwentySecond International Conference on

Architectural Support for Programming Languages and Operating Systems (Xi’an, China) (ASPLOS ’17).

[5] Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, and Colin Raffel. 2020. Learning-based Memory Allocation for C++ Server Workloads. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20). Association for Computing Machinery, New York, NY, USA, 541–556. https://doi.org/10.1145/3373376.3378525