Learning-Based Prefetching Strategy

Rizky Ramadhana Putra Kusnaryanto
izky@vt.edu
Virginia Polytechnic Institute State University
Blacksburg, Virginia, USA

Hansen Idden
hansenidden@vt.edu
Virginia Polytechnic Institute State University
Blacksburg, Virginia, USA

Nathan Roberts
nathan03@vt.edu
Virginia Polytechnic Institute State University
Blacksburg, Virginia, USA

Hyun Myung Kang
dkang1013@vt.edu
Virginia Polytechnic Institute State University
Blacksburg, Virginia, USA

ABSTRACT

The main challenge in the modern data-intensive applications is when the working sets exceed available memory capacity. While prefetching techniques can mask access latency by proactively loading data into faster memory tiers, developing effective prefetching strategies remains challenging due to the need to accurately predict future data requirements while efficiently utilizing limited memory resources. Traditional prefetching approaches rely on fixed heuristics or simple pattern detection, which often fail to capture the complex I/O behaviors exhibited by modern applications. We propose a novel learning-based prefetching strategy that leverages neural networks to identify and adapt to complex access patterns. Our approach formulates prefetching as a regression task, predicting both the offset (where to prefetch) and size (how much to prefetch) of future I/O requests. We implement three neural network models: one predicting offset, one predicting size, and one predicting both simultaneously. Through rigorous evaluation against state-of-the-art prefetching techniques, including stride-based and Markov Chain algorithms, our models demonstrate superior performance, achieving up to 51× improvement over stride-based prefetchers and up to 15.4× improvement over Markov Chain-based approaches. Our best-performing model achieves 78.7% prefetch coverage and 26.1% hit ratio with a buffer size of 1000, significantly outperforming traditional methods. These results demonstrate the potential of learning-based approaches to revolutionize prefetching strategies for modern storage systems.

KEYWORDS

Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:
Rizky Ramadhana Putra Kusnaryanto, Hansen Idden, Nathan Roberts, and Hyun Myung Kang. 2018. Learning-Based Prefetching Strategy. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Modern computing systems have difficulty dealing with the storage performance gap, where storage access latencies are a lot slower than CPU speeds and DRAM access. Even with high-speed SSDs, a random read can consume in the order of 50–100 µs [5], about 10^3 times slower than an L1-cache hit and three orders slower than a DRAM access. Memory-disaggregation fabrics help by exposing larger pools, but even state-of-the-art CXL and RDMA setups report remote-page latencies of tens of microseconds, leaving the fundamental storage-performance gap largely intact [1]. Thus, I/O is still a critical bottleneck in various fields. Prefetching is known as a tool to bridge the gap of the delayed latency by anticipating future data accesses and loading data into faster memory ahead of time [2]. An effective prefetcher can mask much of the SSD access latency, improving throughput and reducing application stall time. Designing a high-accuracy prefetcher remains a challenge. Unlike simple sequential memory access patterns, real-world I/O workloads exhibit complex, non-linear access patterns across vast address spaces. Specifically, flash-based SSDs operate over huge logical block address ranges that are often sparse, making it difficult to predict the next accessed block. Additionally, Prefetching must be done at the right time, as the data should arrive right before it is needed, meaning the predictor must accurately predict when and how big of data to fetch [2].

Conventional prefecting techniques struggle to tackle this problem effectively. Simpler heuristics, such as sequential read-ahead or stride detection, work well for linear access patterns but struggle when accessing irregular or data-dependent patterns. Traditional prefecters often assume a limited working set or a small set of repeating patterns; they cannot predict the complex correlations in large data storage. The precision decreases when strides overlap, the address space grows into billions of logical blocks, and the table sizes grow beyond practical limits [3]. More complex profile-guided engines like vRPG2 raise CPU-cache hit rates with run-time-injected prefetches, but they operate on program counters and offer little assistance once requests spill past DRAM into the block layer [9]. When these heuristics mis-speculate, they face a drastic trade-off: small settings let stalls bleed through, while aggressive settings over-fetch, wasting useful data and saturating bandwidth.

Machine-learning approaches promise a more complex understanding of access behavior. Google's *Voyager* neural prefetcher lifted temporal-coverage rates from 51 % to nearly 74 % on complex

SPEC workloads by using a hierarchical sequence model [8], while CACHEUS applied learning to cache-replacement decisions and surpassed well-tuned LIRS and ARC across diverse server traces [7]. Yet most published machine-learning prefetchers demand tens of megabytes of weights and milliseconds-scale interface budgets, making them impractical for SSD controllers that typically house only a few megabytes of SRAM and sub-GHz ARM cores [4].

To meet those deployability constraints while retaining machine learning's predictive power, we deployed a dual-head LSTM prefetcher. A single lightweight LSTM backbone ingests recent address-delta tokens and I/O size data, then incorporates into two cooperative heads: one classifies the next offset delta—pinpointing *where* the next access will land—while the other regresses request size and lead time—specifying *how much* data to fetch and exactly *when* it must arrive. Careful dimensioning and quantization shrink the entire network to 30 KB. Because the model is compact, we can refine it online, allowing the prefetcher to track phase shifts and mixed-tenant contention without offline retraining.

2 METHOD

As noted in the introduction, the "acmart" document class can be used to prepare many different kinds of documentation — a double-blind initial submission of a full-length technical paper, a two-page SIGGRAPH Emerging Technologies abstract, a "camera-ready" journal article, a SIGCHI Extended Abstract, and more — all by selecting the appropriate *template style* and *template parameters*.

This document will explain the major features of the document class. For further information, the *LaTeX User's Guide* is available from https://www.acm.org/publications/proceedings-template.

3 EVALUATION

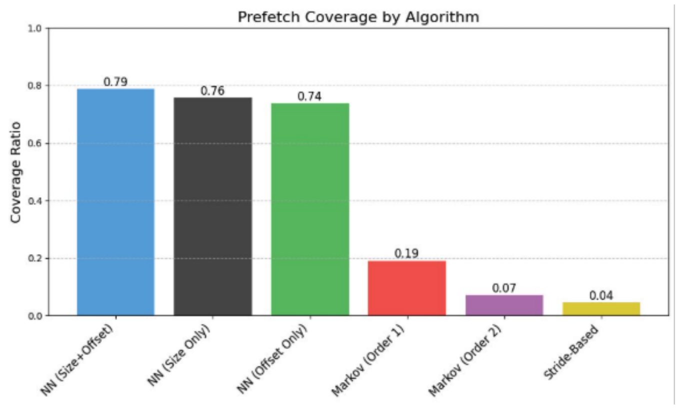

Figure 1: Prefetch Coverage Comparison Between Our Approach VS. Conventional Approach

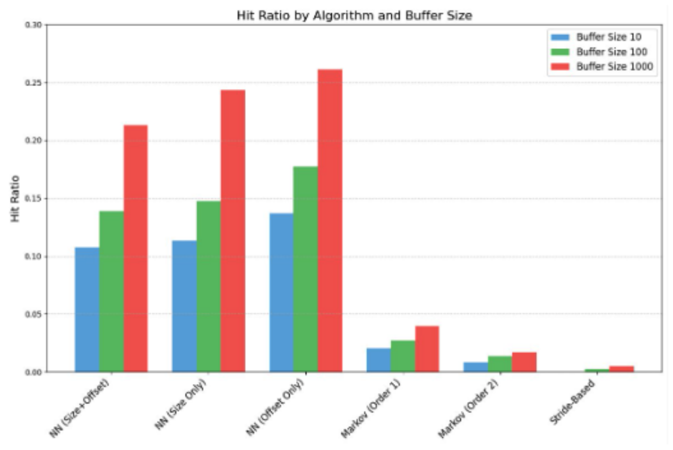

Figure 2: Hit Ratio Comparison Between Our Approach VS. Conventional Approach

4 CONCLUSION

In this project, we successfully implemented and evaluated a machine learning based prefething strategy for SSD storage systems that can significantly outperforms traditional techniques. Our neural network models effectively predict future I/O access patterns by learning the relationships of the previous offset and size of the storage requests.

Despite the challenges of predicting both the offset (location) and size of the future storage requests with the enormous address space, our models demonstrated remarkable accuracy and efficiency compared to the conventional prefetching algorithm. We developed three approaches in this project-predicting offset only, size only, and both offset and size-each showing significant improvements over the conventional prefetching approach.

Our implementation achieves up to 51× better performance than stride-based prefetching algorithm and up to 15.4× better performance than Markov Chain-based prefetching algorithm approaches. The high prefetching coverage, around 78%, combine with high hit ratios, up to 26%, demonstrates that our models efficiently utilize prefetching resources.

These result confirms the viability of applying machine learning techniques to storage systems, particularly for prefetching mechanism challenges that conventional heuristic-based approaches struggle to address. our work establishes a foundation for future research into learning-based storage optimization that can adapt to diverse and evolving workload.

5 FUTURE WORKS

Based on our promising results, we identify several important directions for future research.

SSD Emulator Implementation Our first goal is to deploy our model in FEMU SSD emulator environment [6]. However, FEMU does not have any caching mechanism within the emulator to utilize our prefetcher. To do so, we need to develop our custom caching system inside the emulator. By testing it on an emulator, it will provide insights into real-world performance characteristics and system integration challenges for our approach.

Real System Deployment Given the 51× improvement over a conventional stride-based algorithm and 15.4× improvement over a conventional Markov-chain-based algorithm, we aim to implement our prefetching strategy in a real storage system. This will allow us to evaluate our approach in a real system while carefully measuring and minimizing the latency overhead.

Workload-Specific Optimization Our current model was trained on general block I/O traces from a server. we plan to explore workload-specific optimization, particularly on machine learning workloads. By training our models on I/O traces from Machine Learning model training and inference operations, we may accelerate these workloads by exploiting their unique I/O patterns. This specialized approach could provide significant performance benefits for AI systems with intensive storage requirements.

REFERENCES

[1] Hasan Al Maruf and Mosharaf Chowdhury. 2023. Memory disaggregation: advances and open challenges. 57, 1, (June 2023), 29–37. DOI: 10.1145/3606557.3606562.

[2] 2021. *Learning i/o access patterns to improve prefetching in ssds*. (Feb. 2021), 427–443. ISBN: 978-3-030-67666-7. DOI: 10.1007/978-3-030-67667-4_26.

[3] Chandranil Nil Chakraborttii and Heiner Litz. 2022. Deep learning based prefetching for flash. In https://api.semanticscholar.org/CorpusID:248883763.

[4] Quang Duong, Akanksha Jain, and Calvin Lin. 2024. A New Formulation of Neural Data Prefetching. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, Los Alamitos, CA, USA, (July 2024), 1173–1187. DOI: 10.1109/ISCA59077.2024.00088.

[5] Piyush Kashyap. 2024. Understanding latency numbers: a practical guide to system design. (Nov. 2024). https://medium.com/@piyushkashyap045/understanding-latency-numbers-a-practical-guide-to-system-design-55fa7951cc49.

[6] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Matias Bjørling, and Haryadi S. Gunawi. 2018. The case of femu: cheap, accurate, scalable and extensible flash emulator. In *Proceedings of 16th USENIX Conference on File and Storage Technologies (FAST)*. Oakland, CA, (Feb. 2018).

[7] Liana V. Rodriguez, Farzana Yusuf, Steven Lyons, Eysler Paz, Raju Rangaswami, Jason Liu, Ming Zhao, and Giri Narasimhan. 2021. Learning cache replacement with CACHEUS. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, (Feb. 2021), 341–354. ISBN: 978-1-939133-20-5. https://www.usenix.org/conference/fast21/presentation/rodriguez.

[8] Zhan Shi, Akanksha Jain, Kevin Swersky, Milad Hashemi, Parthasarathy Ranganathan, and Calvin Lin. 2021. A hierarchical neural model of data prefetching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS '21). Association for Computing Machinery, Virtual, USA, 861–873. ISBN: 9781450383172. DOI: 10.1145/3445814.3446752.

[9] Yuxuan Zhang, Nathan Sobotka, Soyoon Park, Saba Jamilan, Tanvir Ahmed Khan, Baris Kasikci, Gilles A Pokam, Heiner Litz, and Joseph Devietti. 2024. Rpg2: robust profile-guided runtime prefetch generation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (ASPLOS '24). Association for Computing Machinery, La Jolla, CA, USA, 999–1013. ISBN: 9798400703850. DOI: 10.1145/3620665.3640396.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009