Deep Learning based Prefetching for Flash

Chandranil Chakraborttii
Trinity College
nil.chakraborttii@trincoll.edu

Heiner Litz
University of California Santa Cruz
hlitz@ucsc.edu

ABSTRACT

Prefetching in solid-state drives (SSDs) is a process of predicting future block accesses and loading them into the main memory ahead of time. In this paper, we describe the challenges of prefetching in SSDs, elaborate why prior approaches fail to achieve high accuracy, and present a deep neural network (DNN) based prefetching technique that significantly outperforms the state-of the-art. We address the challenges of prefetching in very large sparse address ranges, as well as prefetching in a timely manner by predicting ahead of time. We show our proposed technique outperforms prior prefetching approaches based on Markov chains by up to 8× and the existing stride prefetchers by up to 800× and on real-world applications running on cloud servers.

1 INTRODUCTION

Real-world applications not only perform sequential accesses, but also follow complex workload patterns [1]. Applications are frequently used simultaneously by multiple users performing independent tasks, resulting in a mix of sequential and random IO requests. Learning SSD storage access patterns for prefetching is a challenging task for the following reasons. As SSDs are increasing in their storage capacity to 16TB and beyond, drives are now supporting billions of LBAs. Since prefetching is successful only if every bit of the logical block address is predicted accurately, models are required to predict which logical block address (LBA) to prefetch with perfect accuracy within a large LBA space. This space is often sparse, as the operating system allocates blocks within the filesystem layer, and hence, even sequential data within the files may be mapped to arbitrary LBAs within SSD. Furthermore, as prefetching needs to be timely, simply predicting the next LBA and the requested IO size is not sufficient, and it is required to predict multiple accesses into the future.

Existing prefetching mechanisms [2, 5] are limited by the computational complexity and difficulty of correctly predicting future IO accesses. For instance, the read-ahead prefetcher [2] is limited to prefetching the next data item within a file to accelerate sequential accesses. More advanced prefetchers that can learn complex IO access patterns have been dismissed because of their computational costs. Modern SSDs and operating systems offer a wide range of telemetry data for analysis. Utilizing the IO access tracing in hardware and software enables collection of large, clean, and automatically labeled datasets that can fuel powerful machine learning models. In this work, we utilize sequence-to-sequence neural networks to learn spatial IO access patterns of applications from block level IO traces collected from a diverse set of data center applications. Specifically, we leverage Long Short-Term Memory (LSTM) [3] and make the following contributions. First, our model provides high accuracy even in the presence of complex interleaved IO streams. Second, we address the challenge of timeliness by predicting multiple IO accesses ahead of time.

2 METHODOLOGY

We used a total of 10 block-level IO traces for our experiments from three sources running applications in live production servers [1, 6]. We preprocess the input traces to address the problem of large LBA range. The number of unique memory addresses within an SSD is typically of the order of billions, rendering a separate class for each memory address impractical. To reduce the address space, we take the l_1 norm of each pair of consecutive LBAs (LBA delta). For example, if consecutive IO accesses starting from LBA 100 are requested as 101, 103 and 106, the corresponding LBA deltas were recorded as 1, 2, and 3, respectively. This significantly reduces the number of classes that our model has to predict. We identify the top 1000 frequently occurring LBA deltas and assign each one of them to a class in decreasing order of their frequencies. The remaining LBA deltas are assigned to a separate class representing a “no prefetch" operation, thus limiting the number of classes for a model to predict to 1001.

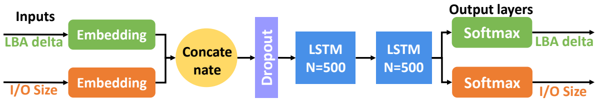

Figure 1: Model Architecture

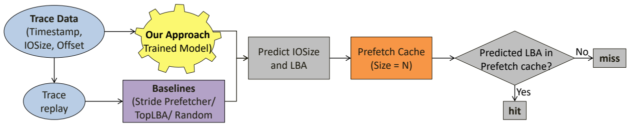
Trace Data (Timestamp, IOSize, Offset
Trace replay
Our Approach Trained Model)
Baselines (Stride Prefetcher/ TopLBA/ Random
Predict IOSize and LBA
Prefetch Cache (Size = N)
Predicted LBA in Prefetch cache?
No
miss
Yes
hit

Figure 2: Block diagram of the proposed simulator

Table 1: Comparison with baseline prefetchers

Dataset Name	No. Samples	Naive Prefetcher	Stride Prefetcher	Markov Prefetcher	Our (Accuracy)	Our (Precision)	Our (Recall)
VDI_1	5226120	0.17	0.01	0.09	0.73	0.76	0.71
VDI_2	4443487	0.21	0.01	0.07	0.59	0.75	0.49
VDI_3	2902328	0.19	0.02	0.12	0.66	0.73	0.57
VDI_4	2408227	0.21	0.05	0.09	0.73	0.77	0.69
MSR_1	2244642	0.14	0.01	0.21	0.41	0.66	0.31
MSR_2	1211034	0.09	0.21	0.17	0.49	0.65	0.33
MSR_3	45746222	0.12	0.001	0.16	0.79	0.89	0.46
MSR_4	45283980	0.33	0.007	0.15	0.53	0.66	0.38
MS_1	1600430	0.27	0.02	0.25	0.63	0.79	0.53
MS_2	1714151	0.41	0.003	0.07	0.77	0.83	0.61

The requested IO sizes for the chosen applications ranged from 4KB to several MBs with up to 10,000 different IO sizes for an individual application. In order to reduce the number of possible target IO size values, we round off each observed IO size to the nearest number that is a power of 2, i.e. 2^n, and use n as an IO size class. This reduces the number of possible target IO sizes for most applications to 16 while still supporting requests of up to 64MB size. A limitation of this approach is that, in the worst case, roughly twice as many as required 4KB blocks may be prefetched from the SSD. We designed our DNN model to concurrently predict both LBA delta and IO size during inference. The model architecture is shown in Figure 1. The model has two separate input layers, one for IO size and one for LBA delta, where each input layer is an embedding layer consisting of 500 neurons. The inputs to the model are categorical, one-hot, representation of the two features, LBA deltas and IO size, each being fed to separate embedding layers. The model has two hidden LSTM [3] layers, where each LSTM layer has 500 hidden nodes. The outputs of the two embedding layers are first concatenated and then fed to the shared LSTM layers. The final output layer is split into two branches, where each branch is a dense layer consisting of softmax nodes. The number of neurons in the LBA delta output layer is 1001, representing top 1000 LBA deltas and a "no prefetch" LBA delta, and the number of neurons in the IO size output layer ranged between 12-20, depending on the dataset. The number of neurons in each of the first three layers of the model was set to 500 to ensure a good representation of input features, and we used a dropout of 0.2 to prevent overfitting of the model. Having an initial embedding layer facilitates better representation of the input features and helps the LSTM layers to learn effectively from sequential data.

For a fair comparison of our proposed prefetcher against prior baselines, evaluating just the precision and recall is not sufficient. As motivated before, analyzing the prefetcher's timeliness is necessary to evaluate the end-to-end performance gains of prefetching, as even the most accurate prefetcher will not improve the performance if it lacks timeliness. In order to compensate for the model's prediction latency and the latency to perform a read from the SSD, it is required to generate predictions ahead of time (PA). We evaluate the end-to-end performance as follows. As we iterate through the test dataset, the trained models continuously generate prefetch candidate predictions that are inserted into the cache. Every IO access is checked against the cache, and if the LBA is present in cache, it is recorded as a hit, otherwise it is recorded as a miss. We utilize the Least Recently Used (LRU) [5] eviction policy for our experiments. The architecture of the simulator is presented in Fig. 2.

3 RESULTS

Table 1 shows the comparative performance of our neural network based pre-fetcher against three chosen baselines based on accuracy, precision, and recall results. For each sample, our prefetcher predicts both LBA and IO size in increments of 4KB blocks, as the minimum block size for a drive operation in SSD is typically of 4KB size. We only count the actual blocks that are correctly prefetched. For each data sample, we prefetch only the top predicted LBA and IO size using the prediction with the highest confidence. For model training, we used a batch size of 64, look back of 64, and predict-ahead of 64 in our experiments. Each prefetcher was provided a cache size of 1000. Our proposed prefetcher consistently outperforms all three baselines delivering up to 11× improvements over the stride prefetcher [4] using Microsoft SNIA traces with the same cache size. For VDI traces, our proposed prefetcher achieved the highest accuracy, providing 800× improvements over the stride prefetcher. Our prefetcher also achieved the highest precision and recall compared to the baselines. The Markov chain based prefetcher performed considerably worse compared to our prefetcher, with the accuracy ranging between 7% and 25%, performing even worse than the Naive prefetcher in several cases.

REFERENCES

[1] Chandranil Chakraborttii and Heiner Litz. 2020. Learning i/o access patterns to improve prefetching in ssds. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 427–443.

[2] WU Fengguang, XI Hongsheng, and XU Chenfeng. 2008. On the design of a new linux readahead framework. *ACM SIGOPS Operating Systems Review* 42, 5 (2008), 75–84.

[3] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[4] Ando Ki and Alan E Knowles. 2000. Stride prefetching for the secondary data cache. *Journal of systems architecture* 46, 12 (2000), 1093–1102.

[5] Arezki Laga, Jalil Boukhobza, Michel Koskas, and Frank Singhoff. 2016. Lynx: A learning linux prefetching mechanism for ssd performance model. In *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE, 1–6.

[6] Chunghan Lee, Tatsuo Kumano, Tatsuma Matsuki, Hiroshi Endo, Naoto Fukumoto, and Mariko Sugawara. 2017. Understanding storage traffic characteristics on enterprise virtual desktop infrastructure. In *Proceedings of the 10th ACM International Systems and Storage Conference*. 1–11.