Efficient SSD Caching by Avoiding Unnecessary Writes using Machine Learning

Hua Wang
HUST
Wuhan National Laboratory for Optoelectronics
hwang@hust.edu.cn

Xinbo Yi
HUST
Wuhan National Laboratory for Optoelectronics
orglanss@hust.edu.cn

Ping Huang*
Temple University & HUST
templestorager@temple.edu

Bin Cheng
Shenzhen Tencent Computer System Co., Ltd.
bencheng@tencent.com

Ke Zhou*
HUST
Wuhan National Laboratory for Optoelectronics
k.zhou@hust.edu.cn

ABSTRACT

SSD has been playing a significantly important role in caching systems due to its high performance-to-cost ratio. Since cache space is much smaller than that of the backend storage by one order of magnitude or even more, write density (writes per unit time and space) of SSD cache is therefore much higher than that of HDD storage, which brings about great challenges to SSD's lifetime. Meanwhile, under social network workloads, quite a few writes on SSD are unnecessary, e.g., Tencent's photo caching shows that about 61% of total photos are just accessed once whereas they are still swapped in and out of the cache. Therefore, if we can predict this kind of photos proactively and prevent them from entering the cache, we can eliminate unnecessary SSD cache writes and improve cache space utilization.

To cope with the challenge, we put forward a "one-time-access criteria" that is applied to cache space, and further propose a "one-time-access-exclusion" policy. Based on that, we design a prediction-based classifier to facilitate the policy. Unlike the state-of-the-art history-based predictions, our prediction is non-history-oriented, which is challenging to achieve a good prediction accuracy. To address this issue, we integrate a decision tree into the classifier, extract social-related information as classifying features, and apply cost sensitive learning to improve classification precision. Due to these techniques, we attain a predication accuracy over 80%. Experimental results show that "one-time-access-exclusion" approach makes caching performance outstanding in most aspects, taking LRU for instance, hit rate is improved by 17%, cache writes are decreased by 79%, and the average access latency is dropped by 7.5%.

*Corresponding authors: Ke Zhou and Ping Huang

ICPP 2018, August 13–16, 2018, Eugene, OR, USA

ACM ISBN 978-1-4503-6510-9/18/08.
https://doi.org/10.1145/3225058.3225126

CCS CONCEPTS

• **Computer systems organization → Cloud computing**;

KEYWORDS

SSD, Photo Caching, Machine Learning, Social Network

ACM Reference Format:
Hua Wang, Xinbo Yi, Ping Huang, Bin Cheng, and Ke Zhou. 2018. Efficient SSD Caching by Avoiding Unnecessary Writes using Machine Learning. In *ICPP 2018: 47th International Conference on Parallel Processing, August 13–16, 2018, Eugene, OR, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3225058.3225126

1 INTRODUCTION

Traditional HDDs have long been the main storage media for information storage. Entering the big data era where high storage performance is highly expected for various applications, storage systems are facing performance challenges. In order to accelerate the performance of services, flash-memory based SSD has been widely used as a cache layer on top of HDD based storage systems. Currently, researches on SSD buffer cache mainly target two goals: lifetime extension and performance improvement.

Lifetime of SSD A key factor influencing SSD's lifetime is the amount of write that go to an SSD, which we call it "write traffic". Unfortunately, compared with serving as backend storage, the write traffic to a caching SSD is much more significant than the backend storage systems, as the basic cache principle is to absorb popular content. Take a typical caching structure for instance: one SSD (1TB), serving as a caching layer, for a backend server array which is composed of 10 HDDs (10*2TB), the ratio of write density(defined as the number of writes per unit time and space) on the caching SSD to the underlying storage system is approximately 20:1, assuming accesses to the storage space is uniformly distributed. Serving as a caching media instead of backend storage makes an SSD subject to extensive writes. Therefore, how to judiciously manage the amount of write becomes a crucial problem for caching SSDs, otherwise they will wear-out quickly due to intensive writes.

At present, most of the researches on the lifetime extension of SSD have focused on the following aspects: optimizing the garbage collection [5, 33] and wear leveling mechanism [6, 11]; improving

the ECC robustness [14, 22]; employing a write cache to reduce SSD writes [26, 31]. While those approaches are effective in extending SSD lifetime, our proposed approach in this paper is complementary to them and reduces unnecessary writes to caching SSDs by enforcing an admission policy.

Performance of SSD Improvement of SSD performance is mainly achieved through two ways: hardware and algorithm. Ideally, we want an adequately large SSD space to accommodate all data. In reality, the SSD cache space can only be much smaller. On the other hand, replacement algorithms can greatly affect caching performance, from basic LRU, FIFO, LFU to more advanced S3LRU [18], LIRS [16], ARC [23], each one excelling some locality, e.g., temporal/spatial/content locality, therefore appropriate for specific scenarios and there is no "One-Size-Fit-All" algorithm.

Tencent Inc. is the largest social network service provider in China, whose online communication software, WeiXin and QQ, serve billions of clients, where the amount of data is huge and more importantly, it grows very fast. Take QQ for example, till August 2017, there are 2 trillion photos worth 300 PBs, with a daily increase of 300 million photos, and 50 billion daily views. To cope with such a scale of intensive accesses, cache servers are configured with SSD. During the past two years, we've been working with the company's engineering team on how to optimize Tencent's photo buffer cache. Originally, we tried to expand cache space and apply different cache algorithms, but they led to little improvements. The reason is that current cache space is already very large. We further analyzed the traces of photo cache, and found that a large number of photos are accessed only once, but they are still swapped into the cache as others. This observation inspired us that if we can avoid those photos entering to the SSD cache, we can greatly reduce the SSD writes. In doing so, cache performance will benefit from the increase of cache utilization.

To materialize this idea, we found that the biggest challenge comes from the prediction of future visits. Because most of the existing prediction methods are based on historical information, but the accesses we need to predict here are one-time-access, which has no historical information for reference. As a result, the accuracy of prediction using existing methods is hard to guarantee. On the other hand, service types and users are diverse and varying, which leads to dynamic and regularly unpredictable access patterns. Therefore, it is not realistic for the prediction to be implemented in a certain way. After various attempt efforts, we realize that machine learning provides a good solution in our scenario, as it not only excels at massive data analysis, but also adapts to dynamic workloads and automatically adjusts feature selection and prediction, which makes it appropriate for non-history-oriented prediction. We have experimented with seven mainstream Machine Learning methods, and finally we have chosen the decision tree in our classifier.

The contributions of our research include:

(1) We propose an adaptive "one-time-access-exclusion" policy and implement admission policy for a caching system. To the best of our knowledge, we are the first to propose the criteria of one-time-access.
(2) We suggest to use machine learning methods to predict future photo accesses. To address the challenge of unavailability of historical information, we rely on feature extraction and training model optimization to increase the prediction accuracy to reach over 80%.
(3) We implement a caching classifier married with "one-time-access-exclusion" policy, and experimental evaluations demonstrate that it improves in all the following aspects: hit rate, SSD writes, and access latency.

The rest of this paper is structured as follows. Section 2 presents the overview of Tencent photo caching and the motivations of our research. Section 3 describes the application of machine learning to caching. Section 4 describes the design and implementation of intelligent caching. Section 5 presents our experimental results. Section 6 discusses the related works, and we conclude in section 7.

2 BACKGROUND AND MOTIVATIONS

2.1 Tencent Photo Caching

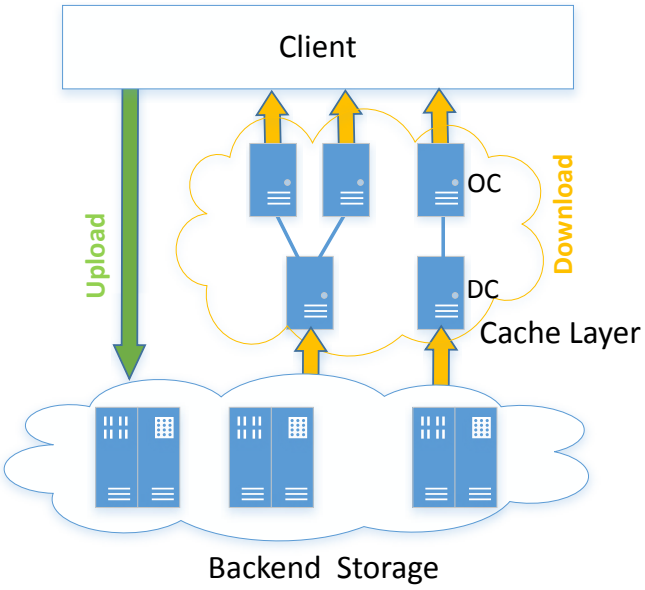


Figure 1: Architecture of Tencent Photo Caching

There are about 850 million active users in Tencent QQ, for which QQPhoto serves as the backend photo storage system. As it is shown in Figure 1, the main feature of QQPhoto is that the upload and download channels are independent from each other. Reason for this is that upload and download traffic are dramatically different: the download channel's data traffic is about several thousand times more than that of the upload channel. Since the upload and download channels are separate, there is no read/write disturbance.

In the download channel, there are a lot of cache servers sitting between the user and backend storage system to provide high performance and save downstream bandwidth. More specifically, the Outside Cache layer (OC) consists of many cache servers that are close to the user, with the purpose of reducing access latency. The Datacenter Cache server (DC) is located within a data center. Its main purpose is to reduce the traffic burden of the backend storage system and promote the performance. Both the OC and the

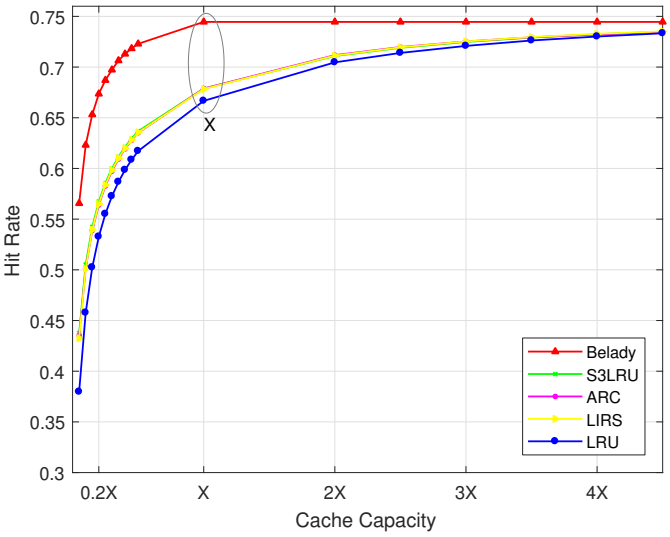


Figure 2: Hit Rate under Different Cache Capacity

DC are equipped with SSDs, forming a distributed buffer cache to handle the extraordinarily large number of requests that QQPhoto is subject to.

2.2 Large Number of Invalid SSD Writes

In order to gain insight into the characteristics of the photo workload, we collect and investigate a 9-days long access log of QQ photo album. In the log set, there are totally 1,481,617,402 objects and 5,856,501,598 accesses, in which 910,568,705 objects were accessed only once, accounting for about 61.5% of total objects and contributing 25.5% of the whole accesses. Suppose cache space is large enough to accommodate all the objects, the cache hit rate would be capped at 74.5%.

In the traditional caching method, if an access is not hit in the cache, the missed file is swapped into cache immediately, regardless whether it will be accessed again or not in the future. From the trace analysis, we find that there are considerable one-time-access files, which leads to unnecessary SSD cache writes. The unnecessary cache writes can cause two negative effects. First, they fasten SSD wearing and shorten the usage of SSD cache. Second, they cause pollution to the cache space, which could otherwise be used to cache other useful data to improve cache performance.

2.3 Impact of Cache Space and Replacement Algorithm

In order to investigate the impact of cache space and replacement algorithms on cache performance, we conduct extensive experiments as shown in Figure 2, in which LRU and Belady are set to be the baseline algorithm and the ideal one respectively. Several other advanced replacement algorithms, including S3LRU, ARC and LIRS, are also used to demonstrate their benefits over LRU.

First, we adjust the size of cache space to see how the size affects cache performance. We have found that there exists an inflection point (denoted as X). When the cache space exceeds X, the hit rate of the Belady algorithm tends to be stable, and the growth of other algorithms slow down gradually. Second, we have observed negligible differences among the three advanced algorithms and all of them only outperform the baseline LRU by around 1%. Moreover, the differences between the ideal Belady and other algorithms are about 9% at the cache size of X and decrease as cache size increases, reaching 4% at $4X$ cache size. In summary, expanding cache size and choosing different cache algorithms have limited impacts on cache performance. We need to resort to other methods to improve cache performance.

2.4 Our Idea

SSDs are inherently subject to intensive writes when used as cache. If we can cut down writes and extend its lifetime, storage service providers can benefit from the reduction of hardware cost. In terms of performance improvement, since it is unpractical to provision sufficient cache space to accommodate all data, the key to optimizing a cache system is improving cache utilization.

In a traditional caching process, once a file access does not hit, the missed file will be fetched into cache, with the premise that the currently accessed file will be accessed soon. However, in the context of an internet-scale distributed system, access patterns could become complicated and unpredictable, which would compromise that premise.

In Tencent's photo caching, there are a large number of one-time-access photos. Once these photos are accessed, they will be swapped in cache and reside in cache without being accessed until they are evicted out of cache. Since their entries will bring about considerable writes, and their occupation will cause cache pollution and reduce cache utilization, we propose to bar them from entering the cache in the first place. Especially, if the cache space is limited, the benefits of barring those writes would be more pronounced.

3 APPLICATION OF MACHINE LEARNING TO CACHE

In order to prevent one-time-access files from entering into cache, it is necessary to predict if currently accessed file is a one-time-access file or not. Prediction target here is one-time-access file, which is never accessed before, therefore no historical information can be used. Non-history-oriented prediction is more difficult than history-based one that has been studied a lot [8, 27, 32]. Furthermore, there is another problem that increases the difficulty of non-history-oriented prediction in cloud storage service environment where service types and users are diverse and varying, which leads to dynamic workloads. Therefore, it is not realistic for the prediction to be handled in a certain way. Machine learning methods not only accomplish efficient analysis for massive data, but also adapt to dynamic workloads and automatically adjust feature selection. Recently, there is also research emerged to adopt machine learning in optimizing dynamic tradeoff in NVMs [9]. Therefore, they are appropriate to be used for non-history-oriented prediction.

3.1 Classifying Algorithm

3.1.1 Selection of Classifying Algorithm. There are many Machine Learning algorithms that can be employed for prediction. Our goal is to find a good tradeoff between high precision and

low complexity. To achieve this goal, we conducted extensive comparison among currently mainstream classifying algorithms. We sample from log dataset, retaining 100 records in every minute out of a total of 144 thousand records. Then we split the sampled data set into training data set and testing data set through cross validation. Performance of each classifier is shown in Table 1. For the sake of narrative clarity, machine learning metrics are shown in Table 2 and Table 3. As can be seen from Table 1, decision tree, ensemble learning methods(such as AdaBoost, Random Forest) all achieve good classifying accuracy. Intuitively, ensemble learning which skillfully integrated multiple decision trees is a better choice for classification algorithm because of better accuracy. However, we found that when using an AdaBoost/Random Forest ensemble learning classifier, even if the number of base learners (decision tree) is increased to 30, it only improves the accuracy by 1%, while bringing about 30 times of computational cost. Therefore, we select decision tree [3] as our prediction algorithm.

Table 1: Performance Comparison of Different Classifiers

Algorithm	Precision	Recall	Accuracy	AUC
Naive Bayes	0.377596	0.99272	0.459069	0.688827
Decision Tree	0.800459	0.765024	0.859903	0.898646
BP NN	0.625511	0.158107	0.691771	0.721861
KNN	0.686851	0.544037	0.768306	0.826307
AdaBoost	0.80709	0.785428	0.867597	0.935989
Random Forest	0.801581	0.77895	0.863792	0.932453
Logic Regression	0.893082	0.173785	0.721236	0.834967

Algorithms in this table could be found in [1]

Table 2: Confusion Matrix for Prediction Result

Reality	Prediction Result	
	Positive	Negative
Positive	TP (True Positive)	FN (False Negative)
Negative	FP (False Positive)	TN (True Negative)

Table 3: Definitions of Metric in Machine Learning

Metric	Definition
Precision (P)	P = TP / (TP + FP)
Recall (R)	R = TP / (TP + FN)
Accuracy	The proportion of the number of correctly classified samples to the total number of samples.
ROC curve	A curve whose vertical axis is True Positive Rate and lateral axis is False Positive Rate.
AUC	Area Under ROC Curve

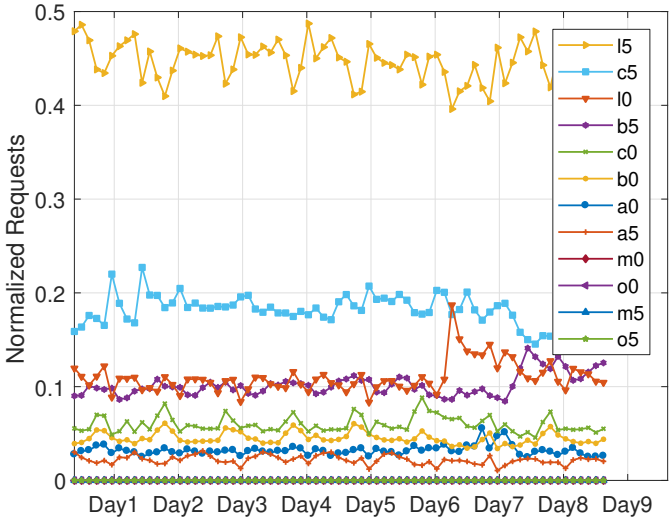


Figure 3: Number of Requests for Different Type of Photos.

3.1.2 Configuration of Classifying Algorithm. In order to avoid decreasing generalization performance caused by over-fitting, we set the upper limit of splitting times to 30 for the decision tree, which is approximately 3 times the number of features. The prediction complexity of a decision tree classifier is highly related to the height of a tree. After comprehensive tests, we found that the height of a tree in our classifier model is 5 in most cases, therefore, in the worst case, only five comparisons to complete prediction.

3.2 Feature Extraction

The implementation effect of prediction algorithm rests with feature extraction to a great extent, let alone our prediction is non-history-oriented, which is much more difficult than existed history-based predictions.

3.2.1 Feature Overview. (1) Photo owner's social information

Active friends: The number of users who have interacted with the photo owner in the recent past.

Average views of owner's photo: Ratio of the total photo views of all images of the owner to the number of user's photos.

(2) Photo information

Photo Type: It is embodied by the resolution and specification of the photo. There are six resolutions(a, b, c, m, l, and o) and two picture specifications (i.e., png and jpg, represented by 0 and 5 respectively), the combination of which leads to a total of twelve photo types. There are significant gaps in the number of requests among different types of photo as shown in Figure 3. Among them, *l5* has the most requests, accounting for about 45%. For a certain type of photo, the access probability is relatively stable in general, but it will fluctuate over time. This phenomenon implies that time feature should be considered in the classifier.

Photo size: Photo size has strong correlation with the resolution of the image. In general, for the same image, the higher the resolution, the larger the photo.

Photo Age: The age of picture measured by the time interval between current time point and the uploaded time point of the given

photo. Intuitively, newer pictures are more popular. Experimental result also confirms the intuition.

Recency: Time interval between the current photo access time and its last access time. If the picture has never been accessed before, it comes to be the time interval between photo access time and its uploaded time.

(3) Information of cache system

Terminal Type: Terminal type mainly includes personal computer and mobile device.

Recent requests: The number of requests accepted by the system in the last minute. Recent requests can indirectly reflect the activity of the whole user group, and a higher number means a more active user group, whose average access probability of each photo is accordingly higher.

Access time: Time in a day when accesses happen. Users often use QQ at relatively fixed time, generally at 8:00 PM, which also means photos are accessed at different time intervals with different probabilities.

3.2.2 Approach of Feature Extraction. Initially, all n features form a full set, $\{a_1, a_2, ..., a_n\}$, and we use the information gain (the greater the information gain, the more helpful to classification) to quantify the quality of each feature, and then we choose the optimal feature, a_i, which has the biggest information gain, to construct the goal set, $\{a_i\}$, and remove a_i from the original full set. Again, information gain of each feature in full set is judged and optimal feature, a_j, is determined and transferred from the full set to the goal set, that is $\{a_i, a_j\}$. If the goal set $\{a_i, a_j\}$ is superior to the previous $\{a_i\}$, which means the effect of new classification is better than before, then the process will be repeated accordingly. Otherwise, the process stops.

Based on this method, we choose average views of owner's photo, photo access recency, photo age, photo access time, and photo type in our final feature set.

3.2.3 Feature Processing. Feature processing comprises of data tagging, discretized processing, and processing of time value.

Data tagging labels for all the one-time-access files. Definition of one-time-access will be explained in Section 4.3, in which LRU algorithm is used. The definition is also applicable to current advanced algorithms, such as ARC, S3LRU, FIFO and etc.

Discretized processing includes two parts. One is mapping twelve types of photo (a0, a5, b0, b5, c0, c5, m0, m5, o0, o5, l0, l5) to twelve discrete value (1-12). The other mapping PC and mobile device to 0 and 1 respectively.

Processing of time value is mainly aimed at access time and time interval. Access time is represented by hour of the day, because workload in social application is highly correlated with human activity and demonstrates some kind of periodicity. Photo age and recency are processed in the granularity of 10 minutes, which accelerate training process convergence.

4 DESIGN AND IMPLEMENTATION OF INTELLIGENT CACHING

4.1 "One-Time-Access-Exclusion" Policy

"One-Time-Access-Exclusion" policy means if a currently accessed photo will not be accessed in a foreseeable long time in future, it should not be admitted into cache. In order to demonstrate the effect of the policy, we give the overview of Tencent cache structure (Section 4.2), where the "One-Time-Access-Exclusion" policy is deployed and experimented. To improve the quality of prediction by machine learning, we set a new criteria of one-time-access for caching (Section 4.3), then we employ cost sensitive learning and daily training model to guarantee the quality of prediction (Section 4.4).

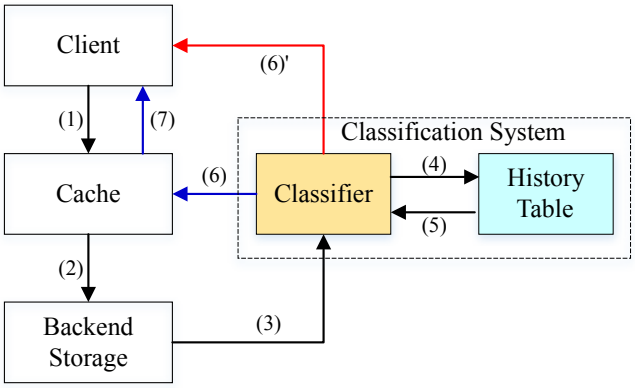


Figure 4: Cache Architecture with Classification.

4.2 Overview of Cache Structure

Figure 4 shows the cache architecture which consists of three parts: SSD cache, backend storage and classification system, in which classification system is the added component compared with traditional cache structure. The main function of classification system is to predict whether the photo to be added to the cache will be accessed again in the near future. If not, the photo will be returned directly to client without being cached, reducing SSD writes and thus increasing the SSD space utilization.

Classification system includes two components: classifier and history table. Classifier makes a judgement based on the features of the photo. The history table retains metadata information of which have been recently determined as one-time-access files. If the photo is judged by the classifier to be a one-time-access file, there still has the chance to rectify it as a non-one-time-access file through the investigation of metadata in the history table. The history table optimizes the results of the classifier in that the whole classification system prefers to classify photos as files to be used later. Because if such photos are wrongly classified, it will lead to subsequent cache misses, which is more costly than allowing it into the cache.

With the classification system added, the workflow could be described as follows:

- (1) Cache receives a request from the client. If hit, then the requested photo is returned to the client and the request is complete.
- (2) If the request is missed in the cache, the cache forwards the request to the backend storage.
- (3) Backend storage returns the target photo to classifier.
- (4), (5) Classifier scans the history table and decides if it is necessary to cache the photo.

- (6), (7) If necessary, the photo will be cached. At the same time, cache will return the photo to the client.
- (6)' If not necessary, the photo will be returned directly to client with history table updated.

Compared with the traditional caching process, classification system is added so that "one-time-access" photos can be prevented from entering cache. Classifier is mainly based on offline classification policy, and it can enlarge cache utilization without increasing caching time.

4.3 One-Time-Access Criteria

To exclude one-time-access photos in the cache, we need to well identify those photos, which calls for a judiciously defined criteria. A rudimentary criteria is to only account in photos which indeed have been accessed only one time during the entire trace period. As we mentioned previously, in the QQPhoto log, about 61.5% photos are accessed only one time and one time accesses account for 25.5% of the total accesses. With this criteria, we are able to reduce 25.5% accesses.

A more sophisticated and accurate criteria should also disallow the photos, which will not be accessed during its cache life if had been admitted in the cache, even though those photos have appeared more than once in the entire trace. Because the cache space is limited in size, it is possible that a swapped in file could be replaced out before cache hit. As a result, writing this type of photos would not bring any benefits to cache performance, but only incur unnecessary SSD writes. We want a new one-time-access criteria which is also able to capture this type of photos.

The key point of the new criteria is to determine the reaccess distance (denoted as M) of a photo file, which is defined as the number of successive accesses between the time when it is brought into the cache and the time when it is accessed again. The problem comes down to determine the value of M so that the file can be classified as one-time-access or not and based on this result determined to be cached or not. In what follows, we develop a simplified model to determine the value of M.

Assume the cache hit rate is h, then there are $M(1-h)$ misses within M accesses, for a full cache in normal working state, which means $M(1-h)$ times of replacements will occur. Take the LRU algorithm for example, suppose cache space is C, average size of one photo is S, then a one-access-time photo will be evicted from the cache after C/S times of replacement, resulting in the following equation:

$$M(1-h) = C/S \tag{1}$$

Therefore, $M = C/[S(1-h)]$.

Since we aim to filter out as many one-time-access files as possible, which means not all the $M(1-h)$ missed files will enter cache. Suppose the percentage of one-time-access files is p, then M times of access will cause $M(1-h)(1-p)$ times of replacement. Therefore, the above equation can be refined as following:

$$M(1-h)(1-p) = C/S \tag{2}$$

With a simple transformation, we have $M = C/[S(1-h)(1-p)]$.

From the above equation, we know that with a constant h, M increases with p. For photo cache, larger M means more rigorous judgement criteria, and files whose reaccess distance is larger than M becomes fewer, which means p is smaller. The trend can be summarized as: $p \uparrow \rightarrow M \uparrow \rightarrow p \downarrow$. Likewise, there also exists the trend of $p \downarrow \rightarrow M \downarrow \rightarrow p \uparrow$.

If p is set to be 0 initially, a corresponding M can be computed. Using this M, p can be obtained though testing the real-world traces, and after several iterative processing, p converges to a constant. Empirically, we set the iterations to be 3. Similarly, we can also estimate the value of h.

4.4 Guarantee Mechanisms for Quality of Prediction

To ensure prediction quality, we adopt cost-sensitive learning approach, the history table, and daily updating model. Cost-sensitive learning approach and the history table try to reduce the negative impact of false predictions on the cache system. Both of these two methods increase the confidence of the prediction results for the one-time-access photos that we care about. Meanwhile, the training model is updated daily to ensure that the classification coincides with current user access pattern.

4.4.1 Cost Sensitive Learning Approach. There are two kinds of errors of misclassification: one is classifying one-time-access into non-one-time-access, i.e., false negative; the other is opposite, classifying non-one-time-access into one-time-access, false positive. The first error will lead to cache space wastage, while the second one will cause subsequent cache miss of the given file. In order to alleviate the cost caused by error, we employ a cost matrix [10] , as shown in Table 4.

Table 4: Cost Matrix

Actual	Predict	
	Positive	Negative
Positive: one-time-access	0	1
Negative: Non-one-time-access	v	0

Since the second type of error causes a higher penalty than the first one, $v > 1$. When cache space is lower, penalty of the second error is relatively higher, vice versa. Through extensively sensitive test, we set $v = 2$ when cache space is in the range of 2GB~12GB, and $v = 3$ when cache space is in the range of 12GB~20GB. Please note that the cache space corresponds to 100 times size of realistic cache due to our 1:100 sample method.

4.4.2 Details of History Table. The history table plays an important role in correcting previous misclassifications. In particular, misclassification here refers to misclassification as a one-time-access photo which will lead to subsequent cache misses. The history table employs a HashMap to store information of the photos that were classified as one-time access recently. For a photo in the history table, if the reaccess distance is smaller than M, it means that the photo was misclassified into a one-time-access photo last time. Therefore, it will be regarded as a non-one-time-access photo this time and removed from the history table. We set the capacity of the

history table stored in DRAM to $M(1-h)p \times 0.05$, nearly 2%~5% of the SSD cache meta-data table, and evict item using FIFO policy.

4.4.3 Daily Updating Model. We adopt the cost matrix and implement training on the data set described in Section 3.1.1, then apply the classifier into the whole trace. We find that classifying performance drops down significantly over time, implying that the prediction accuracy is time-bounded.

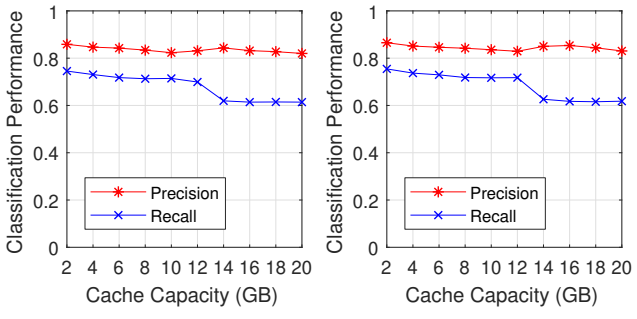
There are two solutions to this problem. One is incrementally updating classification model in a real-time manner. The other is an offline learning manner, i.e., updating the classifier at intervals. We choose the second one, which minimally affect caching performance. In order to determine the update cycle, we analyzed the percentage of one-time-access, p and found that p changes at daily periodicity, reaching the highest and the lowest at 5:00 am and 20:00 pm respectively every day. Considering the system load, we train the classifier at 5:00 am each day, using sample data of the previous 24 hours before 5:00 am. Due to sampling and efficiency of the CART decision tree algorithm, the entire training procedure takes only a few minutes. Then the classifier is used to classify data of the next day.

5 EXPERIMENTAL RESULTS

5.1 Test Design

The original trace contains 5.8 billion records, with a total data volume of 5TB. In order to simplify processing, we randomly sampled trace data using the following steps. First, distinct objects were extracted to construct an object set L. Second, random sampling at 1:100 was conducted on L and object set L' is obtained. Then records included in original data set and whose object id belong to L' were extracted to construct a new trace sequence according to the timestamp. Our final sample data set comprises of about 14 million objects. We conduct our experiments on the sampled trace and range the cache space from 2GB to 20GB, which correspond to about 200GB and 2TB realistic cache space, respectively. Our test environment are as follows: Intel Xeon CPU E5-2609@2.5GHz, 32GB memory, 2TB SSD.

5.2 Experimental Results for Classification System



(a) LRU **(b) LIRS**

Figure 5: Performance of Classification System

We choose the following cache algorithms in our experiments: LRU, ARC, S3LRU, LIRS and FIFO. As discussed in Section 4.3, one-time-access criteria for LRU, ARC, S3LRU and FIFO are the same, whereas the criteria for LIRS is somewhat different, for which $M_{LIRS} = M_{LRU} \times R_s$, where $R_s = C_s/C$. C_s means the cache space of STACK(S), reserving blocks with smaller Inter-Reference Recency, and C means the whole cache space. Without loss of generosity, we conduct experiments for two algorithms, LRU and LIRS, whose results are shown in Figure 5a and Figure 5b respectively.

We can see from the figure that the prediction accuracy of LIRS is slightly better than that of LRU. The reason is that $M_{LIRS} < M_{LRU}$, which means that LRU algorithm needs to predict further into the future, causing it more difficult to guarantee the prediction accuracy. However, in Section 5.3, we will find that the improvement for LRU algorithm using classification prediction is even more pronounced than LIRS. That attributes to two factors: one is that LRU algorithm has more room for improvement. The other is that advanced algorithms, such as LIRS, ARC, have their own strategies in reducing the adverse impact of one-time-access files, thus higher classification accuracy is required for further improvement.

5.3 Experimental Results for Caching System

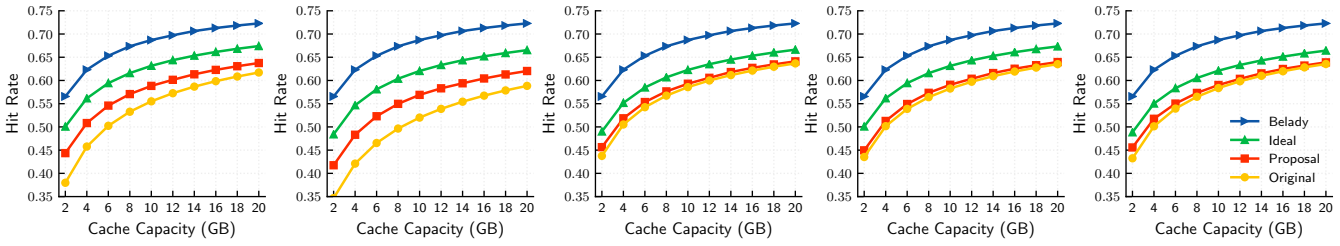
In this sub-section, we present the experimental results of file hit rate, byte hit rate, write rate, byte write rate and response time for LRU, FIFO, S3LRU, ARC, LIRS respectively. In each group of comparison, *Belady* provides the upper limit of cache performance, *Original* means corresponding cache replacement algorithm, such as LRU, FIFO, S3LRU, ARC, LIRS, *Proposal* represents cache system using our classifier, *Ideal* denotes cache system using ideal classifier with its classification accuracy be 100%.

5.3.1 File Hit Rate. When cache capacity ranges from 2GB to 20GB, we can see from Figure 6 that after using classifier, the promotions of FIFO and LRU are the most significant, which are 5%~20% and 3%~17% respectively. Whereas for advanced algorithms, such as S3LRU, the improvement is much less, which is 0.7~4%.

In addition, with the increase of cache capacity, the improvement of hit rate become smaller. The reason is that when cache capacity increases, the judgement threshold (M) for one-time-access file is raised and the classifying performance drops, which leads to lower improvement of hit rate.

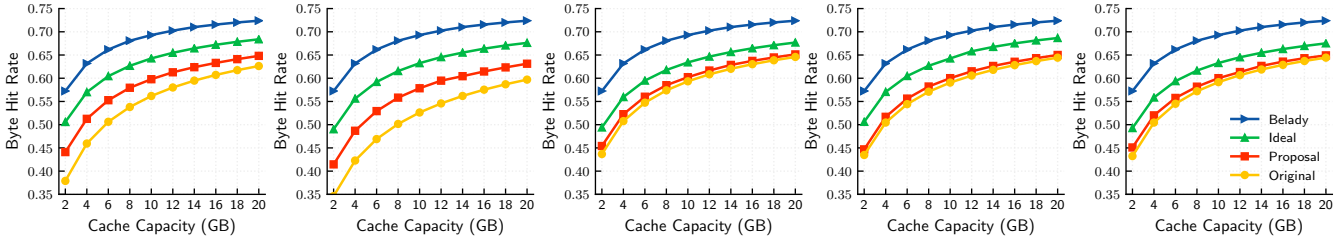
5.3.2 Byte Hit Rate. Different from file hit rate, byte hit rate takes file size into account as well represents the ratio of hit data volume to the whole accessed data, which can indicate the throughput performance of cache system. Similar to hit rate, using classifier, the promotions of FIFO and LRU are quite noticeable, which are 6%~20% and 4%~16% respectively, but it is only 0.9~4% for S3LRU, as shown in Figure 7. Most photos of QQ album are of approximately the same size, and in the meanwhile our classifier is not sensitive to file size, therefore we have not observed significant differences between hit rate and byte hit rate.

5.3.3 File Writes of SSD. With the traditional caching method, every file miss will cause the file to be written to the SSD cache. In our proposed cache system, once a file miss occurs, the file will be written to the SSD cache only when it is predicted not to be one-time-access file, which can decrease the number of SSD



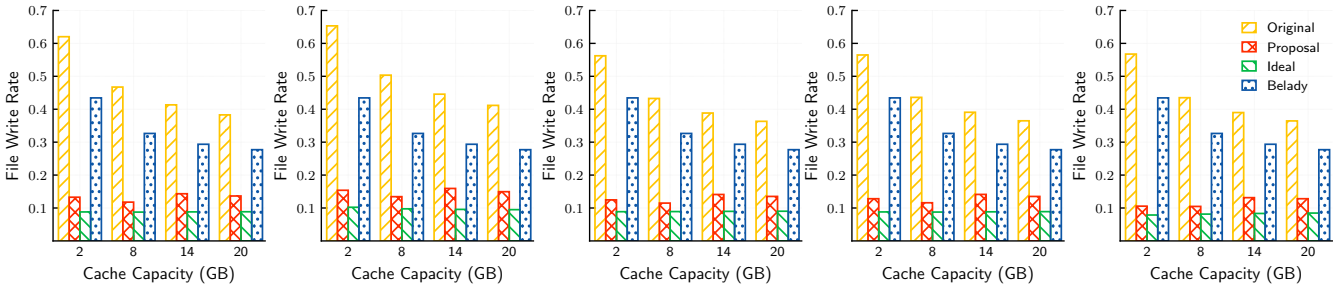
(a) LRU (b) FIFO (c) S3LRU (d) ARC (e) LIRS

Figure 6: File Hit Rate of Cache Replacement Algorithms



(a) LRU (b) FIFO (c) S3LRU (d) ARC (e) LIRS

Figure 7: Byte Hit Rate of Cache Replacement Algorithms



(a) LRU (b) FIFO (c) S3LRU (d) ARC (e) LIRS

Figure 8: File Write Rate of Cache Replacement Algorithms

writes. We can see from Figure 8 that the number of file writes drops significantly for all replacement algorithms if the classifier is applied. Among them, LIRS obtains the most write reduction of 65%~81%. The write rate represents (the written data to SSD) / (the total amount of accessed data).

5.3.4 Byte Writes of SSD. Byte writes consider file size, representing the amount of total data written into SSD. Similar to file writes, byte writes are reduced for all cache algorithms, achieving 60%~80% for LIRS, as it is shown in Figure 9.

5.3.5 Comparison of Performance. Since classification process will inevitably have a certain impact on the response time of cache system, now we discuss it briefly. For the sake of simplicity, notation $t_{classify}$ represents the execution time of the classification system consisting of classifier and history table. Generally, the access latency in a cache system can be calculated using the following equation:

$$T = hit\ rate \times Hit\ cost + (1 - hit\ rate) \times Miss\ penalty \quad (3)$$

For a file access, if the file hits in the cache, the hit cost of the original caching system without prediction is the same as that of our proposed system, which is the sum of query time (t_{query}) and read time (t_{ssdr}) from SSD and is expressed as following:

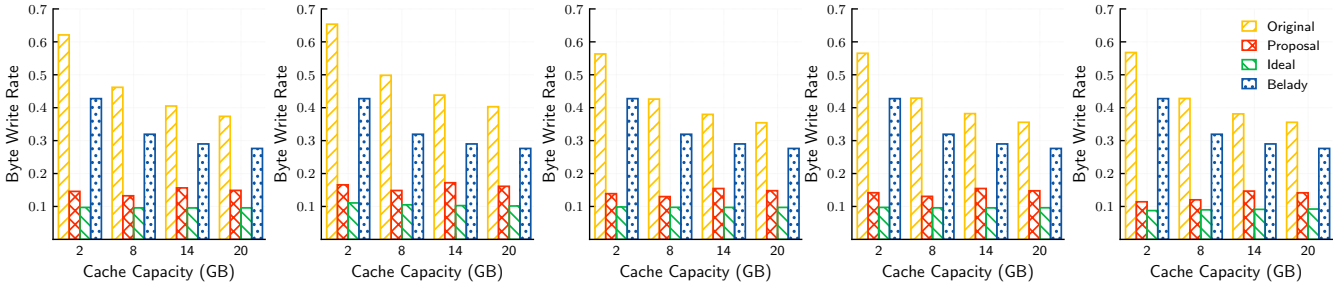
$$Hit\ cost = t_{query} + t_{ssdr} \quad (4)$$

In the case of a file miss, the miss penalty of original system includes the time of querying the cache and reading data from HDD to DRAM, which can be represented as Equation 5. Please note that writing data to SSD should not be taken into account since it can be done in the background.

$$Miss\ penalty_o = t_{query} + t_{hddr} \quad (5)$$

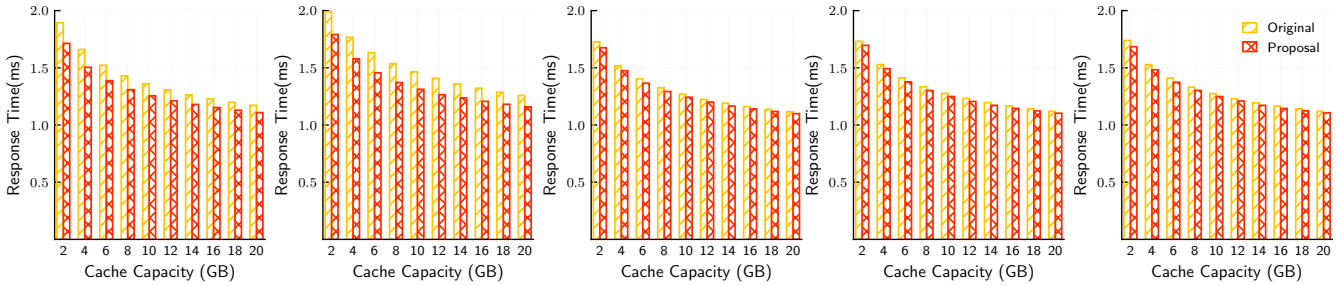
where, t_{hddr} represents the completion time of HDD reading.

In our proposed system, if the current file is predicted to be a one-time-access file, it will be read from HDD and forwarded to DRAM directly. Otherwise, it will be dealt with in the same way as the common system without classification. The missing penalty of the proposed system only includes querying cache, classification, and



(a) LRU (b) FIFO (c) S3LRU (d) ARC (e) LIRS

Figure 9: Byte Write Rate of Cache Replacement Algorithms



(a) LRU (b) FIFO (c) S3LRU (d) ARC (e) LIRS

Figure 10: Response Time of Cache Replacement Algorithms

reading data from the HDD to the DRAM. Therefore, the missing penalty of proposed system can be represented as Equation 6.

$$Miss\ penalty_p = t_{query} + t_{classify} + t_{hddr} \quad (6)$$

where, $t_{classify}$ indicates the execution time of classification.

In this manner, we can compare the access time of no-classification system with our proposed system, shown as Figure 10. For a 32K picture, we set $hddr = 3ms$, $t_{query} = 1us$, $t_{classify} = 0.4us$, please note that t_{query} and $t_{classify}$ are the results of our experiment. We can see that the most improvement is FIFO, 8%~11%, the least is ARC, 1.5%~2.5%.

6 RELATED WORK

Our work focuses on optimizing photo cache performance and extending SSD lifetime by filtering the one-time-access photos from cache. In this section, we mainly discuss related work in two categories: one is caching, especially photo caching in social networks and the other is image popularity prediction.

6.1 Caching

Caching has been an active research topic for many years. Traditional cache replacement algorithms are designed to make use of different characteristics of access patterns such as recency (LRU), frequency (LFU), and some advanced replacement algorithms attempt to exploit both, such as SLRU [18], ARC [23]. Based on these traditional algorithms, there have been many researches on how to further optimize cache performance for specific scenarios [13]. [7, 30] discuss how to improve caching performance in the web environment. [19, 20] focus on predicting when data will be reused to optimize the cache performance on CPU, while [17, 24] leverage machine learning techniques to achieve the similar goals. Their main idea is to imitate Beladay algorithm, that is estimating reuse distance to achieve a higher hit rate, whereas we filter unnecessary data to improve the utilization of the cache space. Besides, it is more difficult to predict reuse distance in our scenario because the data locality and the correlation between features and reuse distance are weak. [17, 20, 25] investigate bypass cache policies to improve cache performance, whereas none of them provides criteria to determine what kind of data should not be put into the cache.

In recent photo caching researches, [15] gives a detailed analysis of Facebook photo caching service stack. [2] discusses how to improve the performance of processing different resolution images in the photo cache. [28] try to solve the problem of random write to Flash when advanced cache algorithm is applied to photo caching. We start from different perspectives and leverage the user's behavior to optimize the photo caching system, enriching the research work in this area.

6.2 Image Popularity Prediction

In recent years, popularity prediction of network content has attracted many research interests. In paper [4, 15], they demonstrate that the access pattern of objects in the cloud environment is Zipf-like or Pareto distribution. [21] points out that social information of users is of great significance for image popularity prediction. Based on [21], [12] introduces sentiment and context features, and analyzes the influence of different sentiments on image popularity. Most of those works focus on popular photos and assume that currently popular photos are more likely to be popular later. Unlike

them, we try to identify cold photos and apply it to a real-world problem. It is worth mentioning that [29] applies popularity prediction to a large-scale production environment. In [29], the popularity of videos on Facebook is predicted, so that the quality of most watch time can be improved, along with reduced overhead.

7 CONCLUSION

When SSD is deployed as a caching layer, write density will be high and its lifetime is threatened. Through the analysis of real-world photo cache traces, we find that it is possible to cut down SSD writes from in the first place. Take Tencent's QQ photo album for instance, user's daily browsing is up to 50 billion, of which there are large number of one-time-access photos. In the regular cache configuration, these photos will be swapped into cache, which introduces a large number of unnecessary SSD writes and reduces cache utilization.

Motivated by this observation, we endeavored to predict future photo accesses and prevent one-time-access photos from entering cache. Owning to the lack of history information, it is hard to make such predictions accurately using traditional cache algorithms. Therefore, we propose to leverage machine learning approaches to extract the most related features dynamically and optimize the training model. In our approach, we obtain a prediction accuracy of over 80% and achieve various extents of improvement in hit rate, SSD writes, and access latency. Though our research aims at Tencent's photo caching, it is also applicable to other buffer caching in cloud storage environments.

8 ACKNOWLEDGMENT

This work was supported in part by the Natural Science Foundation of China (NSFC) under Grant No.61502189 and the National Key Research and Development Program of China under Grant No.2016YFB0800402.

REFERENCES

[1] Ethem Alpaydin. 2014. *Introduction to machine learning.* MIT press.
[2] Xiao Bai, B Barla Cambazoglu, and Archie Russell. 2016. Improved Caching Techniques for Large-Scale Image Hosting Services. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval.* ACM, 639–648.
[3] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees.* Wadsworth.
[4] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. 1999. Web caching and Zipf-like distributions: Evidence and implications. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Vol. 1. IEEE, 126–134.
[5] Li-Pin Chang, Yu-Syun Liu, and Wen-Huei Lin. 2016. Stable Greedy: Adaptive Garbage Collection for Durable Page-Mapping Multichannel SSDs. *ACM Trans. Embed. Comput. Syst.* 15, 1, Article 13 (Jan. 2016), 25 pages.
[6] F. H. Chen, M. C. Yang, Y. H. Chang, and T. W. Kuo. 2015. PWL: A progressive wear leveling to minimize data migration overheads for NAND flash devices. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE).* 1209–1212.
[7] Asaf Cidon, Assaf Eisenman, Mohammad Alizadeh, and Sachin Katti. 2015. Dynacache: Dynamic Cloud Caching. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15).* USENIX Association.
[8] Riley Crane and Didier Sornette. 2008. Robust dynamic classes revealed by measuring the response function of a social system. *Proceedings of the National Academy of Sciences* 105, 41 (2008), 15649–15653.
[9] Zhaoxia Deng, Lunkai Zhang, Nikita Mishra, Henry Hoffmann, and Frederic T Chong. 2017. Memory cocktail therapy: a general learning-based framework to optimize dynamic tradeoffs in NVMs. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture.* ACM, 232–244.
[10] Charles Elkan. 2001. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, Vol. 17. Lawrence Erlbaum Associates Ltd, 973–978.
[11] Eran Gal and Sivan Toledo. 2005. Algorithms and data structures for flash memories. *ACM Computing Surveys (CSUR)* 37, 2 (2005), 138–163.
[12] Francesco Gelli, Tiberio Uricchio, Marco Bertini, Alberto Del Bimbo, and Shih-Fu Chang. 2015. Image popularity prediction in social media using sentiment and context features. In *Proceedings of the 23rd ACM international conference on Multimedia.* ACM, 907–910.
[13] Ping Huang, Wenjie Liu, Kun Tang, Xubin He, and Ke Zhou. 2016. ROP: Alleviating Refresh Overheads via Reviving the Memory System in Frozen Cycles. In *Parallel Processing (ICPP), 2016 45th International Conference on.* IEEE, 169–178.
[14] Ping Huang, Pradeep Subedi, Xubin He, Shuang He, and Ke Zhou. 2014. FlexECC: Partially Relaxing ECC of MLC SSD for Better Cache Performance. In *2014 USENIX Annual Technical Conference (USENIX ATC 14).* USENIX Association, 489–500.
[15] Qi Huang, Ken Birman, Robbert van Renesse, Wyatt Lloyd, Sanjeev Kumar, and Harry C Li. 2013. An analysis of Facebook photo caching. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles.* ACM, 167–181.
[16] Song Jiang and Xiaodong Zhang. 2002. LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. *ACM SIGMETRICS Performance Evaluation Review* 30, 1 (2002), 31–42.
[17] Daniel A Jiménez and Elvira Teran. 2017. Multiperspective reuse prediction. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture.* ACM, 436–448.
[18] Ramakrishna Karedla, J Spencer Love, and Bradley G Wherry. 1994. Caching strategies to improve disk system performance. *Computer* 27, 3 (1994), 38–46.
[19] Georgios Keramidas, Pavlos Petoumenos, and Stefanos Kaxiras. 2007. Cache replacement based on reuse-distance prediction. In *2007 25th International Conference on Computer Design.* IEEE, 245–250.
[20] Mazen Kharbutli and Yan Solihin. 2008. Counter-Based Cache Replacement and Bypassing Algorithms. *IEEE Trans. Comput.* 57, 4 (2008), 433–447.
[21] Aditya Khosla, Atish Das Sarma, and Raffay Hamid. 2014. What makes an image popular?. In *Proceedings of the 23rd international conference on World wide web.* ACM, 867–876.
[22] Ren-Shuo Liu, Chia-Lin Yang, and Wei Wu. 2012. Optimizing NAND flash-based SSDs via retention relaxation. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12).* USENIX Association, 11–11.
[23] Nimrod Megiddo and Dharmendra S. Modha. 2003. ARC: A Self-tuning, Low Overhead Replacement Cache. In *Proceedings of the 2Nd USENIX Conference on File and Storage Technologies (FAST'03).* USENIX Association, 115–130.
[24] Leeor Peled, Shie Mannor, Uri Weiser, and Yoav Etsion. 2015. Semantic Locality and Context-based Prefetching Using Reinforcement Learning. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15).* ACM, 285–297.
[25] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. 2009. Scalable High Performance Main Memory System Using Phase-change Memory Technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09).* ACM, 24–33.
[26] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. 2010. Extending SSD Lifetimes with Disk-based Write Caches. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10).* USENIX Association, 101–114.
[27] Gabor Szabo and Bernardo A Huberman. 2010. Predicting the popularity of online content. *Commun. ACM* 53, 8 (2010), 80–88.
[28] Linpeng Tang, Qi Huang, Wyatt Lloyd, Sanjeev Kumar, and Kai Li. 2015. RIPQ: Advanced Photo Caching on Flash for Facebook. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15).* USENIX Association, 373–386.
[29] Linpeng Tang, Qi Huang, Amit Puntambekar, Ymir Vigfusson, Wyatt Lloyd, and Kai Li. 2017. Popularity Prediction of Facebook Videos for Higher Quality Streaming. In *2017 USENIX Annual Technical Conference (USENIX ATC 17).* USENIX Association, 111–123.
[30] Qiang Yang, Haining Henry Zhang, and Tianyi Li. 2001. Mining web logs for prediction models in WWW caching and prefetching. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 473–478.
[31] Rui Ye, Wentao Meng, and Shenggang Wan. 2017. Extending Lifetime of SSD in Raid5 Systems through a Reliable Hierarchical Cache. In *Networking, Architecture, and Storage (NAS), 2017 International Conference on.* IEEE, 1–8.
[32] Qingyuan Zhao, Murat A Erdogdu, Hera Y He, Anand Rajaraman, and Jure Leskovec. 2015. Seismic: A self-exciting point process model for predicting tweet popularity. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 1513–1522.
[33] Ke Zhou, Shaofu Hu, Ping Huang, and Yuhong Zhao. 2017. LX-SSD: Enhancing the lifespan of NAND flash-based memory via recycling invalid pages. In *Proceedings of the 2017 IEEE 33rd Symposium on Massive Storage Systems and Technology.*