Machine Learning-based Adaptive Migration Algorithm for Hybrid Storage Systems

Milan M. Shetti
Rocket Software
mshetti@rocketsoftware.com

Bingzhe Li
Oklahoma State University
bingzhe.li@okstate.edu

David H.C. Du
University of Minnesota, Twin Cities
du@umn.edu

2022 IEEE International Conference on Networking, Architecture and Storage (NAS) | 978-1-6654-5408-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/NAS55553.2022.9925545

***Abstract*—Hybrid storage systems are prevalent in most large-scale enterprise storage systems since they balance storage performance, storage capacity and cost. The goal of such systems is to serve the majority of the I/O requests from high-performance devices and store less frequently used data in low-performance devices. A large data migration volume between tiers can cause a huge overhead in practical hybrid storage systems. Therefore, how to balance the trade-off between the migration cost and potential performance gain is a challenging and critical issue in hybrid storage systems. In this paper, we focused on the data migration problem of hybrid storage systems with two classes of storage devices. A machine learning-based migration algorithm called K-Means assisted Support Vector Machine (K-SVM) migration algorithm is proposed. This algorithm is capable of more precisely classifying and efficiently migrating data between performance and capacity tiers. Moreover, this K-SVM migration algorithm involves a K-Means clustering algorithm to dynamically select a proper training dataset such that the proposed algorithm can significantly reduce the volume of migrating data. Finally, the real implementation results indicate that the ML-based algorithm reduces the migration data volume by about 40% and achieves 70% lower latency than other algorithms.**

I. Introduction

Unprecedented and ever-increasing 3 V's (Volume, Velocity and Variety) of data continues to put pressure on storage systems to find cost-effective solutions capable of delivering peak performance for all possible workloads [1]. Recently, different types of emerging storage devices come out [2]–[5], which have different density and performance. For example, flash-based Solid State Drives (SSDs) can achieve much faster random access performance with low latency compared to traditional Hard Disk Drives (HDDs) while HDDs are much cheaper than SSDs. Therefore, it is not cost-effective to build a petabyte byte (PB) storage system using only fast devices [6]. Compared with different types of emerging devices, they can have a 100x latency difference and more than 5x price difference. These differences have motivated storage vendors to build two-level hybrid storage systems with different types of storage devices.

A key characteristic of data that remains unchanged is that data has an access life cycle (i.e., not all data are accessed at all times by applications). The desired outcome for a hybrid storage system is to deliver almost all the IO operations from a high-performance tier (e.g., SSDs). To achieve this desired outcome, data have to be moved between tiers depending on the frequency of IO accesses (a process is referred to as data migration). Although data migration between tiers introduces overheads, given a 100x $/IOPS difference between SSDs and HDDs, this also presents an opportunity to design and develop a migration algorithm which can be cost-effective and can also deliver peak performance as demanded by applications. Some previous studies have investigated hybrid storage systems [7]–[11]. They formulated the characteristics of workloads and the properties of devices based on statistical analysis. However, migration optimization has the complexity of NP-hard [7]. To avoid the difficulty of solving the NP-hard problem, those researchers simplified the problem and proposed polynomial time bound heuristic solutions. However, the simplified formulas are not able to precisely express the behaviors of workloads. As a result, the misexpression may result in a large migration volume and decreasing the performance gain in a hybrid storage system. Machine learning (ML) as a classifier has been successfully used in many applications [12], [13]. It can be a good candidate to solve the data migration problem with less migration volume and higher performance gain. This is because the data migration in hybrid storage systems can be regarded as a classification issue to determine/classify data to which storage tier they should reside.

In this paper, we focus on a hybrid storage system containing two types of storage devices (e.g., SSD and HDD) and propose a K-Means assisted Support Vector Machine (K-SVM) migration algorithm. In this algorithm, time is partitioned into periodical duration. In each period, the request access patterns are collected. At the end of the current period, a K-SVM classifier is used based on the request access patterns of this period. Then, a classifier is used to determine which data should be migrated to a different tier in the following period. To increase the precision of the classifier, the K-Means clustering algorithm is introduced to dynamically select a proper training dataset such that the overall migration size can be reduced. Furthermore, we investigate the influence of different capacity ratio between two types of storage on the performance of the migration algorithm. Finally, we conduct the implementation of a large-scale system. We investigate the influence of different system parameters on the performance of the migration algorithm including the time of periodical duration, slice size, the capacity ratio between two types of storage and available back-end bandwidth.

The structure of the paper is as follows. Section II gives a description of a basic SVM migration algorithm and the

978-1-6654-5408-7/22/$31.00 ©2022 IEEE

Authorized licensed use limited to: Oklahoma State University. Downloaded on May 15,2023 at 03:23:21 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Terms and notations used in this paper

PT	performance tier (i.e., fast device)
CT	capacity tier (i.e., slow device)
C	the capacity of the whole system
Slice	the granularity of the unit for data migration
S_s	indicates the slice size (the default value is 200MB)
T	the time intervals to measure request density
Access density	the total number of IO accesses of one slice during the period T
N_s	total number of slices in the system ($N_s = C/S_s$)
N_{PT}, N_{CT}	numbers of slices in PT and CT, respectively
M_{PT}, M_{CT}	The sets of migration candidates. M_{PT}: the set of candidates of $PT \rightarrow CT$; M_{CT}: the set of $CT \rightarrow PT$
Training dataset ratio	training dataset ratio is calculated by the size of training dataset divided by N_s.
r_{PT}	the ratio between PT capacity and the total capacity. ($r_{PT} = N_{PT}/N_s$)
PT hit ratio	the number of requests in PT divided by the total number of requests.
BW	available back-end bandwidth and is indicated by the number of slices migrated in one period (# of slices/T)

other baseline algorithms. The preliminary comparison results and the issues of the basic SVM migration algorithm are provided in Section III. Section IV proposes an K-SVM migration algorithm. Section V shows the experimental result comparison between K-SVM and baseline algorithms. The results of a real large-scale implementation on a large cloud system are provided in Section VI. Finally, the conclusion and future work are described in Section VII.

II. Basic SVM Migration Algorithm

In this section, we introduce a basic support vector machine (SVM) migration algorithm and also describe the basic steps of classification and migration of this algorithm. After that, some baseline algorithms are introduced as well. The terms and notations used in this paper are defined in Table I.

A. Algorithm Description

SVM first proposed by Vapnik *et al.* [14] is a widely used supervised machine learning technique. SVM became popular because of its success in the handwritten digit recognition use case. SVM is a two-class classifier based on the two vectors from the training dataset. It can provide a hyperplane that maximizes the distance between two closest vectors in each of two classes [15]. For the hybrid storage system, the maximum distance between two clusters built by SVM can provide more precise classification/prediction and thus improve the performance and reduce the migration overhead.

In this work, we use SVM to categorize storage slices (slices are units of migration in hybrid storage systems) into two groups based on the historical workload access patterns. After classification, the slices will be migrated to a new location if its current location is mismatched with the SVM classification. The proposed SVM algorithm introduced in this section for

Algorithm 1 Basic SVM Migration Algorithm: training

Input: C, S_s, T
Output: Hyperplane-Z

1: **procedure** TRAINING PROCEDURE
2: $N_s \leftarrow C/S_s$
3: Collecting access density of N_s slices in one T period
4: Sorting N_{PT} and N_{CT} slices based on the access density for PT and CT, respectively
5: Training dataset $(\mathbf{X}, \mathbf{Y}) \leftarrow$ top $x\% \times N_s/2$ slices in PT + the least active $x\% \times N_s/2$ non-zero slices based on the sorted access density. (default $x\% = 10\%$, so the size of training dataset is $x\% \times N_s$)
6: Training linear SVM based on training dataset $(\mathbf{X}, \mathbf{Y})$ to obtain a hyperplane-Z: $Z = AX + B$

storage migration is called a basic SVM migration algorithm (**basic-SVM**) in order to distinguish from the later introduced K-SVM migration algorithm (**K-SVM**).

There are two major steps in the basic SVM migration algorithm (training the basic SVM classifier and classifying and migrating). During each period T, the system records the access density (the number of times being accessed) of each slice. At the end of the period T, based on Algorithm 1 the training dataset is selected. The training dataset is used for building a new SVM classifier (new hyperplane). The classifier is used for classifying all slices based on the information collected last period. Finally, all migration candidates (storage slices) are determined. The migrating process happens in the next period $T + 1$.

Step I – Training: Algorithm 1 indicates the procedure of training. Assume a training dataset (X, Y) consisting of n points in the form of (X_1, Y_1) to (X_n, Y_n), where X_i is the ith slice in the training dataset and Y_i is the label of the ith slice and can be either 1 (Performance Tier (PT)) or -1 (Capacity Tier (CT)) indicating the class which the ith slice belongs to.

For the training dataset, $x\%$ total slices are selected. For the basic SVM migration algorithm, $x\% = 10\%$ is set as the default value. For the later defined K-SVM algorithm, x will be adaptively changed. As indicated in Algorithm 1, the training dataset of the basic SVM is selected from the most active and the least active non-zero slices (slices with activities) of the performance and capacity tiers, respectively. In this way, it uses the most represented data to train the SVM.

After training, an SVM (Hyperplane-Z) is built, and then the Hyperplane-Z will be used for the classification process (testing). The output Hyperplane-Z in Algorithm 1 is a classifier that distinguishes which storage type the input slices should belong to as seen in Eq. 1.

$$\begin{cases} i^{th}\ slice \leftarrow PT, & \text{if Hyperplane-Z(i) == 1} \\ i^{th}\ slice \leftarrow CT, & \text{otherwise} \end{cases} \tag{1}$$

where Hyperplane-Z() function is obtained from Algorithm 1. i is the input slice number. PT means the performance tier. CT indicates the capacity tier. Therefore, if the output of Hyperplane-Z function with input i is 1, that means the ith

Authorized licensed use limited to: Oklahoma State University. Downloaded on May 15,2023 at 03:23:21 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2 Basic SVM Migration Algorithm

1: **procedure** CLASSIFYING PROCEDURE
2: **while** $X_i \in$ PT slices **do**
3: **if** Hyperplane-Z(i) $\neq Y_i$ **then**
4: $M_{PT} \leftarrow M_{PT} + X_i$
5: $i \leftarrow i + 1$
6: **while** $X_i \in$ CT slices **do**
7: **if** Hyperplane-Z(i) $\neq Y_i$ **then**
8: $M_{CT} \leftarrow M_{CT} + X_i$
9: $i \leftarrow i + 1$
10: **end**
11: **procedure** MIGRATING PROCEDURE
12: # of migration slices = min(len(M_{PT}), len(M_{CT}))
13: Ascending sorting M_{PT}
14: Descending sorting M_{CT}
15: **for** $i \leq$ # of migration slices **do**
16: Exchange slices of $M_{PT}(i)$ and $M_{CT}(i)$
17: Updating the labels of slices of $M_{PT}(i)$ and $M_{CT}(i)$
18: **end**

slice should be located in PT. Otherwise, the ith slice should be located in CT.

Step II – Classifying and Migrating: After getting the Hyperplane-Z function from Algorithm 1, we start to identify the migration candidates and do the migration in the next period $T + 1$. Algorithm 2 indicates the procedures of classifying and migrating. First, based on the Hyperplane-Z function, for all slices in PT or CT, if the classification result of the ith slice is not equal to its original label Y_i, then the ith slice is added into its corresponding migration candidate set (M_{PT} or M_{CT}). During the migrating process, we first determine the number of migration slices by using the minimum number between the sizes of M_{PT} and M_{CT}. This is because some slices cannot be migrated/exchanged if the numbers of slices in M_{PT} and M_{CT} are not the same. By ascending sorting M_{PT} and descending sorting M_{CT} (Lines 13-17 in Algorithm 2), it promises that the most active slices in CT and the least active slices in PT are migrated first. The final step is to update the labels of migrated slices and those labels will be used for the next iteration period.

Figure 1 provides an example of the basic SVM migrating algorithm. According to Algorithm 1, the hyperplane-Z is trained based on the x% most and least active non-zero slices in CT and PT, respectively. In Figure 1, after determining the hyperplane-Z, the migration candidates are classified as shown in shaded red regions. Finally, those candidate slices will be scheduled to be migrated to the region that they are supposed to reside.

According to the above description, the basic-SVM algorithm helps classify the migration slices which have similar or different features as the training dataset. The goals of the proposed migration algorithm are to improve the performance (higher PT region hit ratio (e.g., SSD hit ratio)) or to reduce

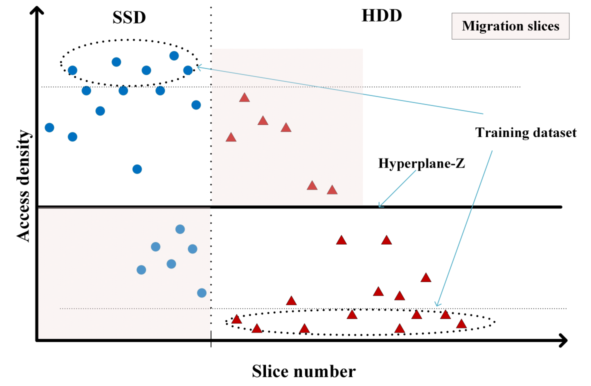


Fig. 1: The basic SVM migration algorithm.

total migration overhead (lower amount of migration data).

B. Baseline Algorithms

In this section, we introduce four baseline algorithms used for comparison in this paper. One is the popularity-based algorithm, which is very popular with solutions from storage vendors. Some previous studies [16] [17] can be simplified to the popularity algorithm. The basic idea is that the algorithm first collects the access density of each storage slice at each period T. Then, according to the access densities, the popularity-based algorithm is to exchange the slice of the highest access density in the CT region with the slice of the lowest access density in the PT region if the lowest value in the PT region is smaller than the highest value in CT region. The migration process will continue until the access densities of slices in the PT region are no longer smaller than the densities of any slices in the CT region. The HAT algorithm [18] is a migration algorithm considering both frequency and recency. The basic idea of HAT in a hybrid storage system (two types of disks) is that there is an LRU queue to record the recency of the historical data. The LRU queue size is the number of slices in the PT region (N_{PT}). The slice at its first-time access will be put into one LRU queue (LRU_Q). If the slice is reaccessed and located in LRU_Q, the slice will be put in PT_LRUQ. At the end of the algorithm, since the PT region only has the size of (N_{PT}), the first N_{PT} slices in PT_LRUQ should be put into the PT region. By comparing the locations of current slices, the migration slices will be put into M_{CT} and M_{PT}.

Another baseline algorithm is the Least Recently Used algorithm (LRU), which is a popular policy used in the eviction algorithm of memory cache. The LRU algorithm keeps the least recently accessed slices in an LRU queue for the PT region and keeps the most recently accessed slices in the MRU (most recently used) queue for the CT region. After period T, the algorithm exchanges the slices in the LRU queue with the slices in the MRU queue. The migration size of the LRU algorithm for each period is proportional to the sizes of MRU and LRU queues. By default, we set the LRU and MRU queue size to $N_s * 10\%$.

ChewAnalyzer algorithm [19] is another migration scheme for hybrid storage systems. The scheme is based on a hi-

Authorized licensed use limited to: Oklahoma State University. Downloaded on May 15,2023 at 03:23:21 UTC from IEEE Xplore. Restrictions apply.

TABLE II: Trace characteristics

	# of requests	Total request size (GB)	Trace length (h)	Maximum offset (GB)
MSR Cambridge traces [21]				
prn_1	1.04E+07	212.1	168	385.0
proj_1	1.47E+07	775.9	168	820.0
usr_1	3.63E+07	2135.4	168	820.0
usr_2	1.02E+07	441.8	168	530.0
src1_0	3.00E+07	1538.3	168	273.0
web_2	4.25E+06	263.6	168	169.0
stg_1	2.13E+06	85.5	168	101.7
mds_1	1.54E+06	88.7	168	474.0
proj_3	2.09E+06	20.9	168	220.0
Systor'17 traces [22]				
LUN0	6.38E+07	1607.8	36	4737.2
LUN1	6.27E+07	1794.9	36	4418.6
LUN3	6.54E+07	1638.6	36	4016.5
FIU traces [23]				
home3	9.18E+05	3.6	504	18.6
online	5.70E+06	21.7	720	7.9
webuser	7.73E+06	30.9	672	7.9
webmail	7.80E+06	29.7	720	18.2

erarchical classifier [20] to classify the access patterns of workloads. They used different storage I/O workload characterization dimensions and the classifier analyze the access patterns step by step. To make a fair comparison, we simplify the ChewAnalyzer to a two-tier storage system. The first step is to classify the I/O density. Then, the second step is to distinguish the read and write performance. Finally, the sequence/randomness of workloads is classified. The high I/O intensive, write-intensive, and random workloads are assigned to PT (i.e., fast) devices and others are scheduled to CT (i.e., slow) devices.

III. Performance of Basic-SVM Algorithm

A. Trace Characteristics and System Configuration

In the performance comparison, we use two types of traces, MSR Cambridge traces [21] and Systor'17 traces [22] to evaluate the performance of a hybrid system and migration overhead of all these algorithms. The trace characteristics are summarized in Table 2. Two metrics are used to indicate the performance of migration algorithms, PT hit ratio and total migration size. The PT hit ratio is defined as the number of requests satisfied by the slices in the PT region (i.e., fast device) divided by the total number of requests. The total migration size indicates how much data have been migrated between the two tiers. Therefore, a migration algorithm with a higher PT hit ratio and a smaller migration size will be better than others.

At the beginning of running traces, we preconditioned the storage system by writing all the slices that responded to the first portion of the requests to the PT region until the PT region is full. Then, the rest of the storage slices are written to the CT region (i.e., slow device). This precondition is practically used by industries to simply initialize a hybrid storage system. This preconditioning process is applied to all algorithms and is used in all simulations and experiments in this paper.

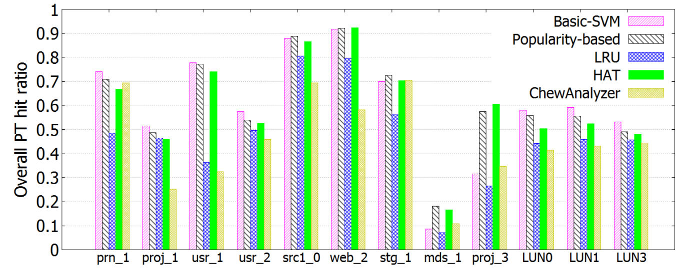


Fig. 2: PT hit ratio comparison between basic-SVM algorithm, popularity-based, HAT and LRU algorithms.

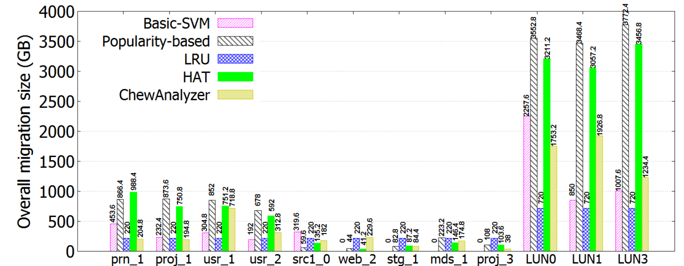


Fig. 3: Migration size comparison between basic SVM, Popularity-based, HAT and LRU migration algorithms. "0" indicates there are no migration data.

B. Performance Comparisons

In this section, the performance comparisons between the basic-SVM algorithm, Popularity-based, HAT and LRU are made. In the experiments, the system capacity is set to 500GB, which contains 100GB SSD and 400GB HDD. The default slice size (S_s) is set to 200MB. Thus, there are a total of 2500 slices, 500 slices in SSD and 2000 in HDD. For those traces having larger maximum offsets than 500GB (like LUN0, LUN1 and LUN3), the offset is scaled into the range of 0-500GB, which is directly divided by a constant value. For example, for those traces from Systor'17, the offsets of traces are divided by 10. The configuration with scaling is equivalent to the configuration of 5TB total capacity and 2GB slice size without scaling. For the convenience of comparisons, the scaling is able to put the results of all traces in the same figures. For the basic-SVM algorithm, the training dataset is set to 10%. The size of the LRU queue is also set to 10%. The migration time interval (T) is 14 hours for MSR Cambridge traces and 1 hour for Systor'17 traces. By doing that, the total number of requests per T in each type of trace keeps similar.

As shown in Figures 2 and 3, the LRU algorithm has the worst overall PT hit ratio. The reason is that the LRU algorithm always migrates the least recently used slices and cannot reflect the characteristics of workloads. Therefore, it causes a much low PT hit ratio. The migration only happens for each period. So, the LRU policy is capable of improving the cache hit ratio by immediately replacing the most recent accessed data but is not good at in the storage migration scheme. For the other four algorithms, the overall PT hit ratio for most of the traces is similar, while the basic SVM algorithm achieves a much smaller overall migration size. This is because ChewAnalyzer, HAT and Popularity based

Authorized licensed use limited to: Oklahoma State University. Downloaded on May 15,2023 at 03:23:21 UTC from IEEE Xplore. Restrictions apply.

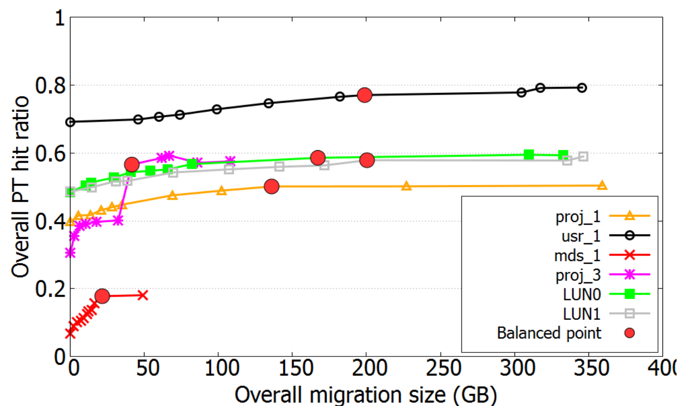


Fig. 4: Relationship between overall migration size and overall PT hit ratio for the SVM migration algorithm.

Algorithm 3 K-SVM Migration Algorithm: training

Input: C, S_s, T
Output: Hyperplane-Z

1: **procedure** TRAINING PROCEDURE
2: $N_s \leftarrow C/S_s$
3: Collecting access density of N_s slices in one T period
4: Sorting all slices in PT
5: Remove top 0.2% slices
6: Do K-Means clustering for PT region (K=2).
7: Adding the removed top 0.2% slices to the cluster at the top position.
8: Do K-Means clustering for CT region (K=2).
9: Training dataset $(\mathbf{X}, \mathbf{Y}) \leftarrow$ all slices at the top cluster of PT $+$ all slices at the bottom cluster of CT
10: Training linear SVM based on training dataset $(\mathbf{X}, \mathbf{Y})$ to obtain a hyperplane-Z: $Z = B$

schemes use the constant schemes to determine the access patterns. Therefore, they cannot dynamically follow the change of workloads and they achieve either a lower PT ratio or higher migration overhead than the basic SVM scheme. However, there are three exceptions. For the traces mds_1 and proj_3, the basic SVM migration algorithm only gets about 8% and 31% overall PT hit ratio, respectively. They are much smaller than the PT hit ratios of the popularity-based and HAT algorithm (18% and 17% for mds_1, and 57% and 60% for proj_3). For trace src1_0, although the basic-SVM, popularity-based and HAT algorithms achieve similar PT hit ratio, the basic-SVM needs to transfer 5x and 3x larger migration size than the popularity-based and HAT algorithms. According to these three exceptions, the issues of the basic SVM migration algorithm are investigated and discussed in the following subsection. After that, a new K-SVM migration algorithm is proposed for solving those issues in Section IV.

C. Issues of Basic-SVM Migration Algorithm

After investigating the three traces that the basic-SVM algorithm has worse performance than that of popularity-based algorithm, we found that the issue is selecting improper training datasets. As discussed in Section III-B, the basic-SVM migration algorithm ended up with a larger migration size for trace src1_0. The migration size is determined by the SVM hyperplane which is trained by a selected training dataset. Thus, we first investigate the relationship between the overall migration size and training dataset ratio under the system as configured and discussed in Section III-B.

To find the relationship between migration size and PT hit ratio, we vary the training dataset ratio to obtain different migration sizes for the basic-SVM migration algorithm. As shown in Figure 4, the PT hit ratio is increased with the raising migration size at the beginning and then the overall PT hit ratios become saturated. The goal of migrating data in a hybrid system is to achieve a higher PT hit ratio while maintaining a small migration size. So, those so-called **balanced points** in Figure 4 have good trade-offs between the migration size and the PT hit ratio. As for the issue of large migration sizes for the basic SVM migration algorithm in Section III-B, it is because the result of the basic-SVM algorithm locates far away from the balanced point of src1_0 (at the right side) in Figure 4. The reason for having a large migration size is caused by an improper training dataset due to the constant training dataset ratio. For different traces, the request access patterns are different and the same training dataset ratio is not a good choice. Moreover, even for the same trace, the request access patterns are changed and different at different iterations. Therefore, the training dataset ratio directly affects the performance of a migration algorithm (the PT hit ratio and total migration size). A proper training dataset ratio is useful for solving the issue of large migration sizes (investigated in Section V).

IV. K-SVM MIGRATION ALGORITHM

In this section, a modified SVM migration algorithm called K-SVM migration algorithm (**K-SVM**) is introduced to remedy the two issues of the basic SVM algorithm as discussed in Section III-C.

The proposed K-SVM migration algorithm is shown in Algorithm 3. Compared to the basic-SVM algorithm in Algorithm 1, the main differences are the training dataset selecting (Lines 4-8 in Algorithm 3). The proposed K-SVM has two advantages compared to other schemes. One is to accurately predict the workloads and then can achieve a better hit ratio. Moreover, the precise classification can help reduce the migration overhead since those incoming requests are put in the 'right' location based on the classification and there is no need to migrate those slices. As a result, the migration cost will be reduced. In this paper, the maximum migration is limited by the available bandwidth. In the following results, we assume that the system has adequate bandwidth to migrate slices between two tiers.

To remedy the improper training dataset issue, the basic idea is to include the most representative slices as many as possible into the training dataset for SVM. For example, we want to include most of the relatively highly accessed slices in the training dataset of the PT region. By doing that, those relatively high accessed slices can effectively represent

Authorized licensed use limited to: Oklahoma State University. Downloaded on May 15,2023 at 03:23:21 UTC from IEEE Xplore. Restrictions apply.

the feature of the PT region. Additionally, those slices in the training dataset will not be migrated due to the feature of SVM and thus it potentially reduces the migration size. Similarly, the training datasets should also exclude the slices which cannot represent the feature of the region. Therefore, by replacing a constant training dataset ratio, we use the K-Means clustering algorithm [24] to group the similar slices in PT and CT regions, respectively (K=2 used in this paper). The K-Means clustering algorithm is used for PT and CT regions, respectively, with one dimension input (access density).

In some cases, one or two slices located in the PT region have really high access frequencies than others. However, we do not want to only use one or two points to represent the PT region. Therefore, to overcome those outliers, we force the top cluster containing at least 0.2% slices as shown in Lines 4-7 in Algorithm 3. Therefore, by using the modified K-Means clustering algorithm, the training dataset is adaptively selected by the algorithm itself. As a result, compared to other algorithms the K-SVM algorithm achieves smaller migration sizes and higher PT hit ratios in Section V.

V. Experimental Result

To find how well the K-SVM algorithm is applied to the data migration problem of hybrid storage systems, we compare the performance of the K-SVM algorithm with that of basic SVM, popularity-based, HAT and the performance of balanced points (discussed in Section III-B). A new type of traces (FIU trace [23] as shown in Table II) is added in this experiment. The system configurations are set to the same as the configuration in Section III-B.

A. Overall Performance Comparison

The performance and overhead comparisons are shown in Figure 5 and Figure 6 respectively. Among all algorithms, the LRU algorithm has the worst overall PT hit ratio. This is because the LRU algorithm always migrates the least recently used slices and cannot reflect the characteristics of workloads. Therefore, it causes a much low PT hit ratio. The migration only happens for each period. So, the LRU policy is capable of improving the cache hit ratio by immediately replacing the most recent accessed data but is not good at in the storage migration scheme. In the future experiment comparisons, we do not compare the LRU algorithm by varying system parameters. The K-SVM achieves similar or a little lower PT hit ratios as the balanced points. For the migration size, the balanced point results always have the lowest values among most traces. For traces proj_1 and usr_1, the balanced point has larger migration sizes but higher PT hit ratios than the proposed K-SVM algorithm. For the rest of the traces, the newly proposed K-SVM algorithm achieves close migration sizes to the balanced points and obtains much smaller migration sizes compared to the basic-SVM, popularity-based, LRU, HAT and ChewAnalyzer algorithms.

In summary, although for some traces, the K-SVM algorithm has slightly larger migration sizes compared with the results of balanced points, it remedies the issues described in Section III-C. Moreover, reducing the migration size is significant for all traces (2x - 8x on average). Therefore, the K-SVM migration algorithm effectively selects a proper training dataset for the SVM classifier and gains very close solutions to the balanced points which have the smallest migration size and the highest PT hit ratio.

B. K-SVM Overhead Discussion

The overhead of the K-SVM scheme mainly comes from two aspects. One is the metadata overhead of recording collected trace information. The second one is the computation overhead of machine learning algorithms. Assume the total capacity of PT and CT tiers are 500 GB (PT: 100GB and CT: 400GB). The slice size is 200MB. The metadata information only has about 16KB. Compared to the total 500GB capacity, the metadata overhead will have little influence on the systems. For the other overhead, we investigate the execution time of the training process. As seen in Table III, the training time is varied from 3.06ms to 211.79ms as varying the slice size. Compared to the period T (hours), the training time of the K-SVM scheme is acceptable.

TABLE III: Training time of the K-SVM scheme with varying slice size

slice size (MB)	50	100	200	500	1000
Training time (ms)	211.79	56.76	16.98	5.45	3.06

C. Effect of Space Capacity Ratio between PT and CT

In this subsection, we investigate the effect of different device capacity ratios. We keep the configuration the same as the previous subsection with the slice size as 200MB.

As seen in Figures 7 and 8, the PT hit ratios are increased with the increased PT capacity (SSD capacity). The K-SVM, popularity-based, HAT and ChewAnalyzer algorithms achieve similar PT hit ratios. For the migration size, our proposed K-SVM migration algorithm achieves about on average 32X smaller migration size when compared to the popularity-based, HAT and ChewAnalyzer algorithms.

Moreover, we investigate the influence of the PT capacity ratio on the migration size for each algorithm. For the popularity-based, HAT and ChewAnalyzer algorithms, they achieve a similar trend of migration size, which is that the migration size first goes up and reaches the peak values when r_{PT} is about 0.35 for Popularity based and about 0.4 for HAT and ChewAnalyzer. Then, the migration size decreases with the increased SSD (PT region) capacity. This is because at the around middle points, the numbers of migration candidates for PT and CT slices become similar and thus the migration size reaches the peak values. Before or after the middle points, either the number of PT candidates or the number of CT candidates is reduced. The mismatched number of candidates results in a smaller migration size. With increased PT capacity, the K-SVM algorithm migrates less amount of data. This is because a larger PT space can store more data and thus the number of migration candidates becomes less. The K-SVM algorithm efficiently selects the migration candidates and

Authorized licensed use limited to: Oklahoma State University. Downloaded on May 15,2023 at 03:23:21 UTC from IEEE Xplore. Restrictions apply.

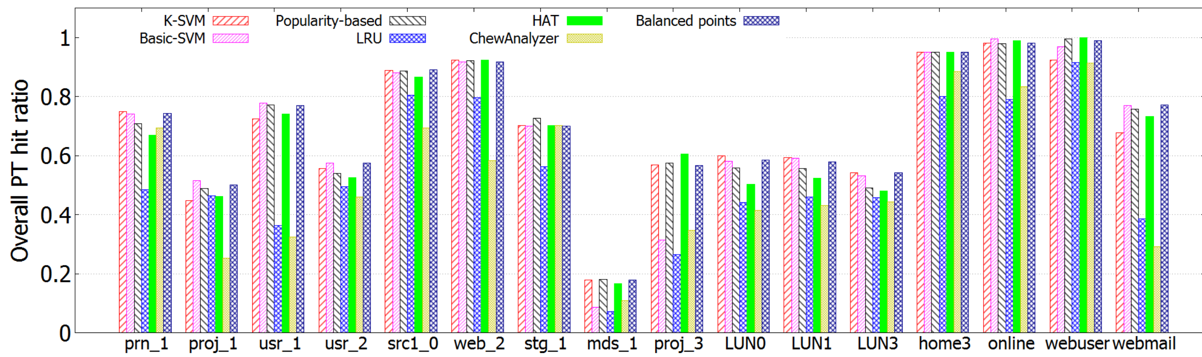


Fig. 5: The PT hit ratio comparisons between K-SVM and other algorithms.

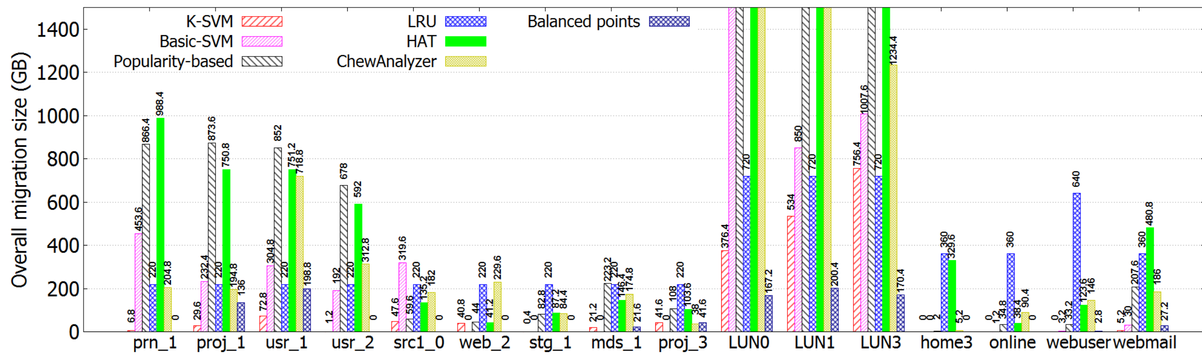
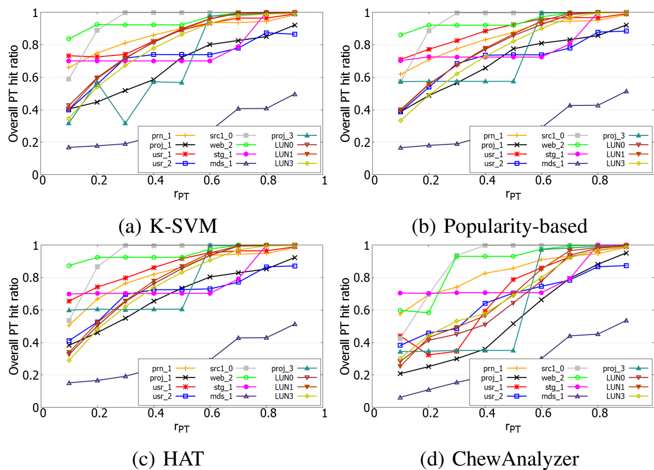


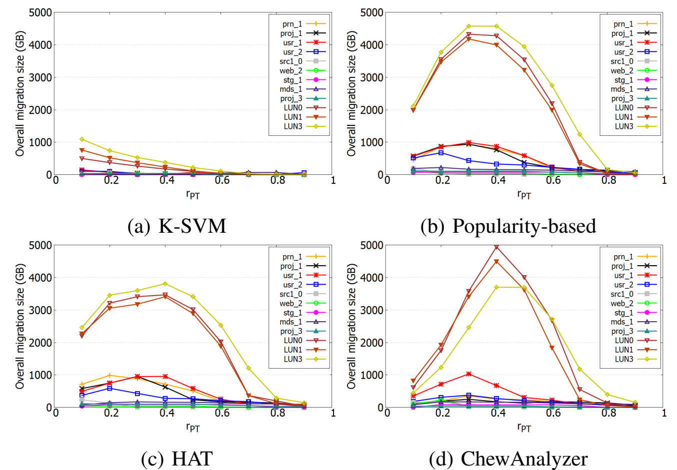
Fig. 6: The migration size comparisons between K-SVM and other algorithms.



(a) K-SVM (b) Popularity-based

(c) HAT (d) ChewAnalyzer

Fig. 7: The PT hit ratio with varying SSD (PT region) capacity.



(a) K-SVM (b) Popularity-based

(c) HAT (d) ChewAnalyzer

Fig. 8: The migration size with varying SSD (PT region) capacity.

keeps the migration size small while maintaining similar PT hit ratios as other algorithms.

TABLE IV: List of applications used on the single hybrid storage system test bed

Tenant	Application
A	Oracle, SAP, VMware
B	Home Directory
C	High Performance Computing
D	Virtual Desktop (VDI), Hyper V
E	SharePoint, Web Farm

VI. Large-scale System Implementation

The prototype of our proposed SVM and popularity-based algorithms are applied to a real enterprise hybrid system. We compared popularity-based and SVM algorithms since based on all previous discussions the popularity-based algorithm performs much similar to the HAT algorithm. The experiments were conducted first on the traces gathered from a live system with multi-tenant 26 different applications running at an enterprise lab system for 31 days. The application list is shown in Table IV. The total storage capacity in the test environment

Authorized licensed use limited to: Oklahoma State University. Downloaded on May 15,2023 at 03:23:21 UTC from IEEE Xplore. Restrictions apply.

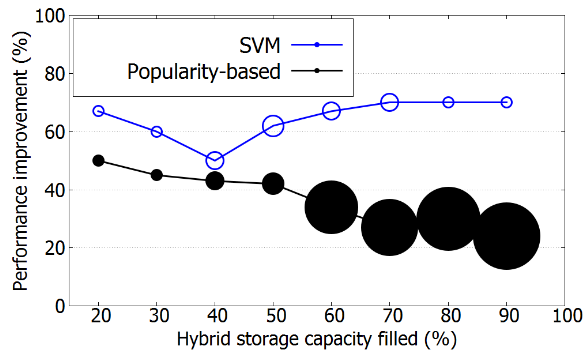


Fig. 9: Performance improvement and migration sizes with various storage capacities for SVM and popularity-based. (the sizes of bubbles indicate the migration size)

for Hybrid Storage System used was 1 PB with 100TB in SSD and 900TB in HDD.

The tests were performed with 47% storage consumed and at the end of the 15 days period and the total capacity used by the system was 49%. We examined the impact of the storage capacity on both popularity-based and SVM algorithms. As shown in Figure 9, the SVM algorithm achieves a higher performance improvement at all different available capacities than the popularity-based algorithm. Meanwhile, the SVM achieves less migration size than the popularity-based algorithm as well.

VII. Conclusion

By applying known machine learning approaches to the storage domain, an entire new set of tools can be applied to solve data tiering problems. In this paper, we propose a migration algorithm based on Support Vector Machine (SVM) and demonstrate the effectiveness of this algorithm to solve an optimization problem in the enterprise storage domain. Moreover, the proposed K-SVM migration algorithm involves K-Means clustering to dynamically select a proper training dataset. The proposed algorithm can tremendously reduce the size of the migration data. Finally, the results of a real implementation indicate that the ML-based algorithm reduces the volume of migration data by about 40% and achieves 70% lower latency compared to other algorithms.

VIII. Acknowledgement

This work was partially supported by NSF I/UCRC Center Research in Intelligent Storage and the following NSF awards 1439622, 1812537, 2204656, and 2204657.

References

[1] Storage performance council(university of massa-chusetts trace repository). http://traces.cs.umass.edu/index.php/Storage, 2007.

[2] Fenggang Wu, Bingzhe Li, and David HC Du. Fluidsmr: Adaptive management for hybrid smr drives. *ACM Transactions on Storage (TOS)*, 17(4):1–30, 2021.

[3] Kan Wu, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. Towards an unwritten contract of intel optane ssd. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19). USENIX Association, Renton, WA*, 2019.

[4] Bingzhe Li, Li Ou, and David Du. Img-dna: approximate dna storage for images. In *Proceedings of the 14th ACM International Conference on Systems and Storage*, pages 1–9, 2021.

[5] Fenggang Wu, Bingzhe Li, Baoquan Zhang, Zhichao Cao, Jim Diehl, Hao Wen, and David HC Du. Tracklace: Data management for interlaced magnetic recording. *IEEE Transactions on Computers*, 70(3):347–358, 2020.

[6] Snia solid state storage: The key to the next gen solid state storage technologies. http://www.snia.org/sites/default/files/AnilVasudeva_Solid_State_Storage_Key_NextGen.pdf.

[7] Zhengyu Yang, Morteza Hoseinzadeh, Allen Andrews, Clay Mayers, David Thomas Evans, Rory Thomas Bolt, Janki Bhimani, Ningfang Mi, and Steven Swanson. Autotiering: automatic data placement manager in multi-tier all-flash datacenter. In *Performance Computing and Communications Conference (IPCCC), 2017 IEEE 36th International*, pages 1–8. IEEE, 2017.

[8] Ilias Iliadis, Jens Jelitto, Yusik Kim, Slavisa Sarafijanovic, and Vinodh Venkatesan. Exaplan: Efficient queueing-based data placement, provisioning, and load balancing for large tiered storage systems. *ACM Transactions on Storage (TOS)*, 13(2):17, 2017.

[9] Jorge Guerra, Himabindu Pucha, Joseph S Glider, Wendy Belluomini, and Raju Rangaswami. Cost effective storage using extent based dynamic tiering. In *FAST*, volume 11, pages 20–20, 2011.

[10] Haixiang Shi, Rajesh Vellore Arumugam, Chuan Heng Foh, and Kyawt Kyawt Khaing. Optimal disk storage allocation for multi-tier storage system. In *APMRC, 2012 Digest*, pages 1–7. IEEE, 2012.

[11] Eric Anderson, Susan Spence, Ram Swaminathan, Mahesh Kallahalla, and Qian Wang. Quickly finding near-optimal storage designs. *ACM Transactions on Computer Systems (TOCS)*, 23(4):337–374, 2005.

[12] Bingzhe Li, Yaobin Qin, Bo Yuan, and David J Lilja. Neural network classifiers using stochastic computing with a hardware-oriented approximate activation function. In *2017 IEEE 35th International Conference on Computer Design (ICCD)*, pages 97–104. IEEE, 2017.

[13] Terrence S Furey, Nello Cristianini, Nigel Duffy, David W Bednarski, Michel Schummer, and David Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.

[14] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[15] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

[16] Feng Chen, David A Koufaty, and Xiaodong Zhang. Hystor: making the best use of solid state drives in high performance storage systems. In *Proceedings of the international conference on Supercomputing*, pages 22–32. ACM, 2011.

[17] Mustafa Canim, George A Mihaila, Bishwaranjan Bhattacharjee, Kenneth A Ross, and Christian A Lang. Ssd bufferpool extensions for database systems. *Proceedings of the VLDB Endowment*, 3(1-2):1435–1446, 2010.

[18] Yanfei Lv, Bin Cui, Xuexuan Chen, and Jing Li. Hat: an efficient buffer management method for flash-based hybrid storage systems. *Frontiers of Computer Science*, 8(3):440–455, 2014.

[19] Xiongzi Ge, Xuchao Xie, David HC Du, Pradeep Ganesan, and Dennis Hahn. Chewanalyzer: Workload-aware data management across differentiated storage pools. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 94–101. IEEE, 2018.

[20] Ishwar Krishnan Sethi and GPR Sarvarayudu. Hierarchical classifier design using mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, (4):441–445, 1982.

[21] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage (TOS)*, 4(3):10, 2008.

[22] Chunghan Lee, Tatsuo Kumano, Tatsuma Matsuki, Hiroshi Endo, Naoto Fukumoto, and Mariko Sugawara. Understanding storage traffic characteristics on enterprise virtual desktop infrastructure. In *Proceedings of the 10th ACM International Systems and Storage Conference*, page 13. ACM, 2017.

[23] Akshat Verma, Ricardo Koller, Luis Useche, and Raju Rangaswami. Srcmap: Energy proportional storage using dynamic consolidation. In *FAST*, volume 10, pages 267–280, 2010.

[24] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

Authorized licensed use limited to: Oklahoma State University. Downloaded on May 15,2023 at 03:23:21 UTC from IEEE Xplore. Restrictions apply.