Machine Learning-based Adaptive Migration Algorithm for Hybrid Storage Systems

MILAN SHETTI, Hewlett Packard Enterprise, USA
BINGZHE LI, University of Minnesota, Twin Cities, USA
DAVID DU, University of Minnesota, Twin Cities, USA

Hybrid storage systems are popular in most large-scale enterprise storage systems since they balance storage performance, storage capacity and cost. The goal of such systems is to serve 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 K-Means clustering algorithm to dynamically select a proper training dataset such that the proposed algorithm can greatly 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 compared to other algorithms.

CCS Concepts: • **Information systems** → *Hierarchical storage management*.

Additional Key Words and Phrases: Hybrid storage, Machine learning, Migration, SVM

ACM Reference Format:
Milan Shetti, Bingzhe Li, and David Du. 2019. Machine Learning-based Adaptive Migration Algorithm for Hybrid Storage Systems. *ACM Trans. Storage* 37, 4, Article 111 (August 2019), 26 pages. https://doi.org/10.1145/1122445.1122456

1 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 [2, 3]. Recently, different types of emerging storage devices come out [35–37, 39], 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 [1]. Compared with different

This work was partially supported by NSF I/UCRC Center Research in Intelligent Storage and the following NSF awards 1439662, 1525617, 1536447, 1708886, 1763008, and 1812537.
Authors' addresses: Milan Shetti, Hewlett Packard Enterprise, Boston, USA, milan.shetti@hpe.com; Bingzhe Li, University of Minnesota, Twin Cities, Minneapolis, USA, lixx1743@umn.edu; David Du, University of Minnesota, Twin Cities, Minneapolis, USA, du@umn.edu.

1553-3077/2019/8-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

types of emerging devices, they can have 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 characteristics 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 high performance tier (like SSDs). In order to achieve this desired outcome, data have to be moved between tiers depending on the frequency of IO accesses (a process referred 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 [4, 12, 16, 29, 40]. They formulated the characteristics of workloads and the properties of devices based on statistical analysis. However, the migration optimization has the complexity of NP-hard [40]. To avoid 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 mis-expression 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 [9, 11, 23, 31]. 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 be resided.

In this paper, we focus on a hybrid storage system containing two types of storage devices (SSD and HDD) and propose an 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 current period, an 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, 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 system parameters on the performance of the migration algorithm including the time of periodical duration, slice size, capacity ratio between SSD and HDD and available back-end bandwidth.

The contribution of this paper is threefold: **I.** This work introduces a machine learning (ML) based migration algorithm for hybrid storage systems. The algorithm can greatly reduce the data migration volume. As a result, the method achieves lower latency than other algorithms. **II.** The selection algorithm of an adaptive training dataset using K-Means clustering improves the precision of migration classification and thus further improves the performance of either the overall migration size or the PT hit ratio. **III.** The effect of different system parameters of machine learning based migration algorithm is thoroughly investigated. The investigation results can be used to design and develop a more effective ML-based migration algorithm in different hybrid storage systems.

The structure of the paper is as follows. Section 2 gives a description of a basic SVM migration algorithm and the other two baseline algorithms. The preliminary comparison results and the issues of the basic SVM migration algorithm are provided in Section 3. Section 4 proposes an K-SVM migration algorithm and the results compared to the two baseline algorithms. Section 5 investigates the effects of different system parameters on the performance of migration algorithms. The results of a real large scale implementation on a large cloud system are provided in Section 6 and related work is introduced in Section 7. Finally, the conclusion and future work are described in Section 8.

Table 1. Terms and notations used in this paper

PT	Performance tier (default device is SSD)
CT	capacity tier (default device is HDD)
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	N_s: total number of slices in the system ($N_s = C/S_s$)
N_{PT}, N_{CT}	N_{PT}, N_{CT}: numbers of slices in PT and CT
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)

2 BASIC SVM MIGRATION ALGORITHM

In this section, we introduce a basic support vector machine (SVM) migration algorithm and also describe basic steps of classification and migration of this algorithm. After that, two baseline algorithms, popularity-based and least recently used (LRU) algorithms, are introduced as well. The terms and notations used in this paper are defined in Table 1.

2.1 Discussion between Hybrid Storage System and Caching System

First, in this subsection, we introduce the difference between the caching system and the hybrid storage system. These two systems have their similarity which intend to store frequently access data in fast devices and infrequently access data in slow devices. However, there are two fundamental difference. One is that they have different behaviors. The caching systems have one or two operations with one incoming request. If read/write cache hit happens, the request just reads/writes on the fast devices. If there is a read/write cache miss, at first one element should be evicted from the fast devices and written to the slow devices. Then, the new request is loaded/written to the fast devices. Data may have two copies on slow and fast devices in the cache systems. The hybrid storage systems basically just read/write requests on their target devices. Data only have one copy. For each period, the data might be swapped between slow and fast devices according to the migration algorithms. The second difference is how to collect information. The caching algorithm collects the request information based on the minimum request unit (page or block) which is a small granularity (KB). For a large scalded system (TB or PB level), the overhead of metadata from fast devices is tremendous and unacceptable. Moreover, the caching algorithms only collect/update data information which is located in the fast devices. For the hybrid storage systems, the monitoring granularity is a parameter that can be adjusted based on the requirement of the system. Moreover, the data information in both fast and slow devices are collected.

Moreover, we compared several caching algorithms (ARC [26], LeCaR [33], LRU and LFU) to the proposed SVM scheme described in Section 2. As indicated in Table 2, the proposed scheme achieves much better hit ratios for most of the traces. The reason is that the hybrid system schemes monitor trace information on both PT and CT and thus can achieve better performance than caching algorithms. Moreover, the write-back operation (element eviction to slow devices) is not considered

Table 2. Hit ratio comparisons between caching algorithms and the proposed scheme with 100GB PT capacity

	ARC	LeCaR	LRU	LFU	K-SVM
prn_1	61.86%	61.86%	61.86%	61.86%	74.87%
proj_1	9.22%	8.00%	8.00%	9.22%	44.74%
usr_1	66.60%	64.76%	64.76%	66.60%	72.53%
usr_2	12.73%	11.07%	10.89%	12.73%	55.59%
src1_0	52.17%	49.91%	49.91%	61.67%	88.88%
web_2	74.97%	74.97%	74.98%	74.98%	92.41%
stg_1	6.80%	6.80%	6.80%	6.80%	70.10%
mds_1	5.05%	5.05%	5.05%	5.05%	17.81%
proj_3	72.07%	72.07%	72.07%	72.07%	56.94%

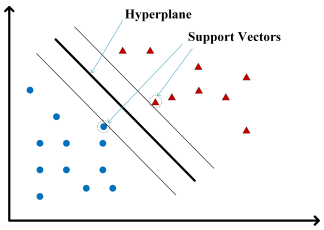


Fig. 1. Support Vector Machine (Theory of Operation)

for the caching algorithms, which makes the performance of caching algorithms even worse. To make fair comparisons, in the following sections, we mainly focus on the algorithms for hybrid storage systems.

2.2 Algorithm Description

SVM first proposed by Vapnik *et al.* [7] is a widely used supervised machine learning technique. SVM became popular because of its success in the handwritten digit recognition use case. As shown in Fig. 1, SVM is a two-class classifier based on the two vectors from the training dataset. It can provide a hyperplane which maximizes the distance between two closest vectors in each of two classes [15]. For the hybrid storage system, the maximum distance between two clusters 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 storage migration is called a basic SVM migration algorithm (**basic-SVM**) in order to distinguish with 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).

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 i^{th} slice in the training dataset and Y_i is the label of the i^{th} slice and can be either 1 (Performance Tier (PT)) or -1 (Capacity Tier (CT)) indicating the class which the i^{th} slice belongs to.

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: $Z = AX + B$

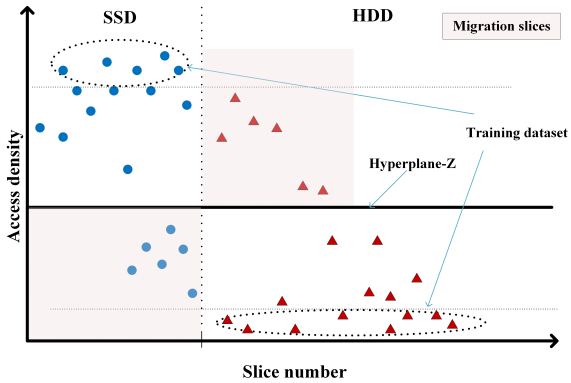


Fig. 2. The basic SVM migration algorithm.

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.

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 performance tier. CT indicates capacity tier. Therefore, if the output of Hyperplane-Z function with input i is 1, that means the i^{th} slice should be located in PT. Otherwise, the i^{th} slice should be located in CT.

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 and then the new hyperplane is re-defined. Finally, all migration candidates (storage slices) are classified. The migrating process happens in the next period $T + 1$.

Step II – Classifying and Migrating: After getting 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

Algorithm 2 Basic SVM Migration Algorithm: classifying and migrating

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**

Algorithm 3 Popularity-based Migration Algorithm

Input: T
Output: Migration candidates M_{PT}, M_{CT}

1: Collecting access density of N_s slices in one T period
2: Reversely sorting N_{PT} slices based on the access density for PT ($\mathbf{N}^i_{PT}$ indicates the i^{th} element in the sorted array)
3: Sorting N_{CT} slices based on the access density for CT ($\mathbf{N}^i_{CT}$ indicates the i^{th} element in the sorted array)
4: $i \leftarrow 0$
5: **while** $\mathbf{N}^i_{CT} > \mathbf{N}^i_{PT}$ **do**
6: $M_{CT} \leftarrow M_{CT} + \mathbf{N}^i_{CT}$
7: $M_{PT} \leftarrow M_{PT} + \mathbf{N}^i_{PT}$
8: $i \leftarrow i + 1$

indicates the procedures of the classifying and migrating. First, based on the Hyperplane-Z function, for all slices in PT or CT, if the classification result of the i^{th} slice is not equal to its original label Y_i, then the i^{th} 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 2 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 2, after determining the hyperplane-Z, the migration candidates are classified

Algorithm 4 HAT migration Algorithm

Input: T
Output: Migration candidates M_{PT}, M_{CT}

```
for each request (Req_i) in T do
    Computing slice number (S_Req) of the request (Req_i)
    if S_Req in LRU_Q then
        PT_LRUQ ← PT_LRUQ + S_Req
    Put S_Req in LRU_Q
for current slices (slice_i) in PT do
    if slice_i is not at first N_PT of PT_LRUQ then
        M_PT ← M_PT + slice_i
for current slices (slice_i) in CT do
    if slice_i is at first N_PT of PT_LRUQ then
        M_CT ← M_CT + slice_i
```

as shown in shaded red regions. Finally, those candidate slices will be scheduled to be migrated to the region that they 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 is to improve the performance (higher PT region hit ratio (SSD hit ratio)) or to reduce total migration overhead (lower amount of migration data).

2.3 Baseline Algorithm I: Popularity-based Algorithm

One of the baseline algorithms is popularity-based algorithm which is very popular with solutions from storage vendors. Some previous works [6][5] can be simplified to the popularity algorithm. The algorithm is defined in Algorithm 3. At period T, the popularity-based algorithm first collects the access density of each storage slice. Then, according to the access densities, the popularity-based algorithm is to exchange the slice of the highest access density in CT region with the slice of the lowest access density in PT region if the lowest value in PT region is smaller than the highest value in CT region. The migration process will continue until the access densities of slices in PT region are no longer smaller than the densities of any slices in CT region.

2.4 Baseline algorithm II: HAT Algorithm

The HAT algorithm [25] is a migration algorithm considering both frequency and recency. The basic idea of HAT in 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 PT region (N_{PT}). As shown in Algorithm 4, the slice at its first time access will be put into one LRU queue (LRU_Q). If the slice is accessed again and it is also located in LRU_Q, the slice will be put in PT_LRUQ. It means the slice is labeled as a PT region candidate. 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. With comparing the locations of current slices, the migration slices will be put into M_{CT} and M_{PT}.

2.5 Baseline algorithm III: LRU Algorithm

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

Table 3. Trace characteristics

	# of requests	Total request size (GB)	Trace length (h)	Maximum offset (GB)
MSR Cambridge traces [27]				
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 [21]				
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

accessed slices in a LRU queue for PT region and keeps the most recently accessed slices in the MRU (most recently used) queue for CT region. After period T, the algorithm exchanges the slices in LRU queue with the slices in 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 $N_s * 10\%$.

2.6 Baseline algorithm IV: ChewAnalyzer Algorithm

ChewAnalyzer algorithm [10] is another migration scheme for hybrid storage systems. The scheme is based on a hierarchical classifier [28] 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 (SSD) devices and others are scheduled to CT (HDD) devices.

3 PERFORMANCE OF BASIC-SVM ALGORITHM

3.1 Trace Characteristics and System Configuration

In the performance comparison, we use two types of traces, MSR Cambridge traces [27] and Systor'17 traces [21] to evaluate the performance of a hybrid system and migration overhead of these three 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 PT region (SSD) 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 responded to the first portion of the requests to PT region (SSD) until PT region is full. Then, the rest of storage slices are written to CT region (HDD). This precondition is practically used by

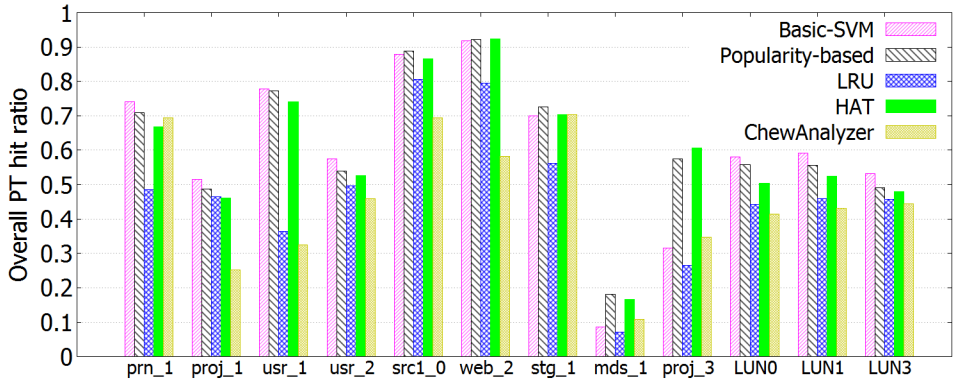


Fig. 3. PT hit ratio comparison between basic-SVM algorithm, popularity-based, HAT and LRU algorithms.

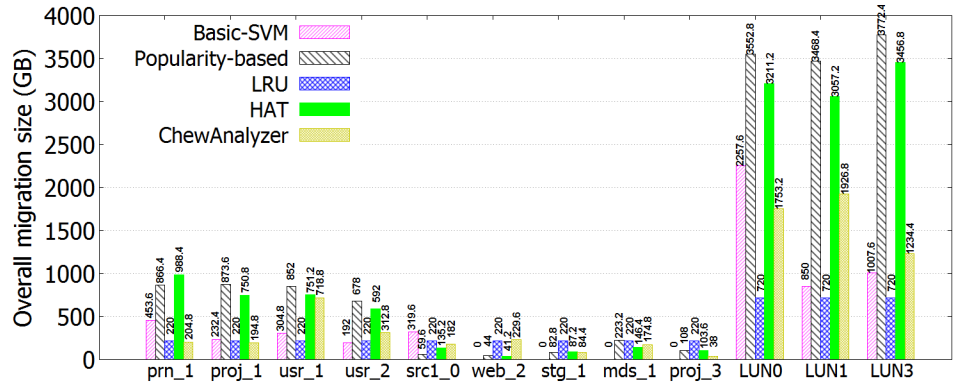


Fig. 4. Migration size comparison between basic SVM, SP, HAT and LRU migration algorithms. "0" indicates there are no migration data.

industries to simply initialize a hybrid storage system. This preconditioning process is applied to all algorithms and are used in all simulations and experiments in this paper.

3.2 Performance Comparisons

In this section, the performance comparisons between the basic-SVM algorithm, SP, 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 total 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 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 traces keeps similar.

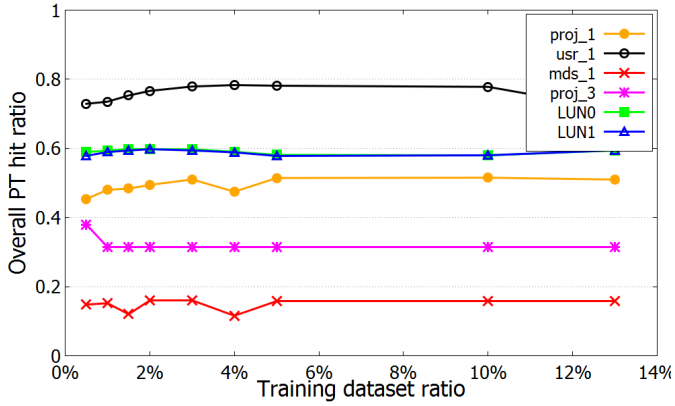


Fig. 5. Relationship between the overall PT hit ratio and the training dataset ratio.

As shown in Figures 3 and 4, 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 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 storage migration scheme. For the other four algorithms, the overall PT hit ratio for most of traces are similar while the basic SVM algorithm achieves much smaller overall migration size. This is because ChewAnalyzer, HAT and Popularity based schemes use the constant schemes to determine the access patterns. Therefore, they cannot dynamically follow the change of workloads and they achieve either lower PT ratio or higher migration overhead than 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 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 4.

3.3 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 3.2, 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 3.2.

As shown in Figure 5, the overall trend of PT hit ratio keeps roughly flat with the increasing training dataset ratio for all traces. However, in Figure 6 the overall migration sizes are changed tremendously and irregularly for different traces with increasing training dataset ratio. The maximum migration size can reach more than 10X than the minimum migration size in Figure 6. Therefore, the migration size is highly related to the training dataset ratio. Based on Figure 6 the

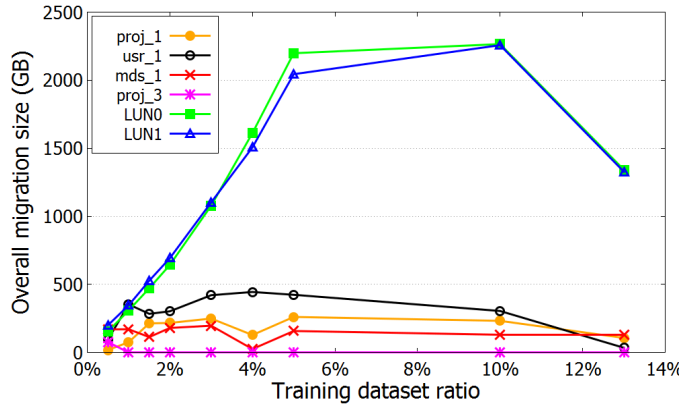


Fig. 6. Relationship between the overall migration size and the training dataset ratio.

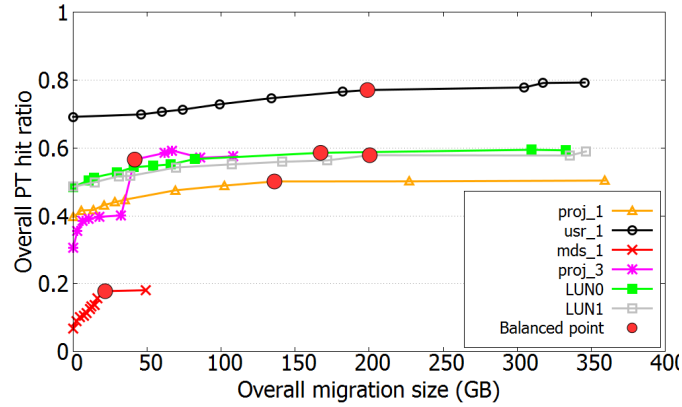


Fig. 7. Relationship between overall migration size and overall PT hit ratio for the SVM migration algorithm.

curves are so irregular and it seems hard to find a rule for picking up a proper training dataset ratio for a specific trace.

Moreover, 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 7, 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 7 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 3.2, 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 7. The reason of 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, at different iterations the request access patterns are changed and different. Therefore, training dataset ratio directly affects the performance of a migration algorithm (the PT hit ratio and total

Algorithm 5 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: $Z = B$

migration size). A proper training dataset ratio is useful for solving the issue of large migration sizes (investigated in Section 4.2).

In the following sections, we modify our proposed basic SVM migration algorithm to an K-SVM algorithm which is able to solve the issues mentioned above. Additionally, the target of the K-SVM algorithm is to get closer to the balanced point for each trace.

4 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 3.3.

4.1 Algorithm Description

The proposed K-SVM migration algorithm is shown in Algorithm 5. Compared to the basic-SVM algorithm in Algorithm 1, the main differences are the training dataset selecting (Lines 4-8 in Algorithm 5).

To remedy the improper training dataset issue, the basic idea is to include the most representative slices as many as possible into training dataset for SVM. For example, we want to include most of the relatively highly accessed slices in the training dataset of PT region. By doing that, those relatively high accessed slices can effectively represent the feature of 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 [17] 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 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 5. 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 4.2.

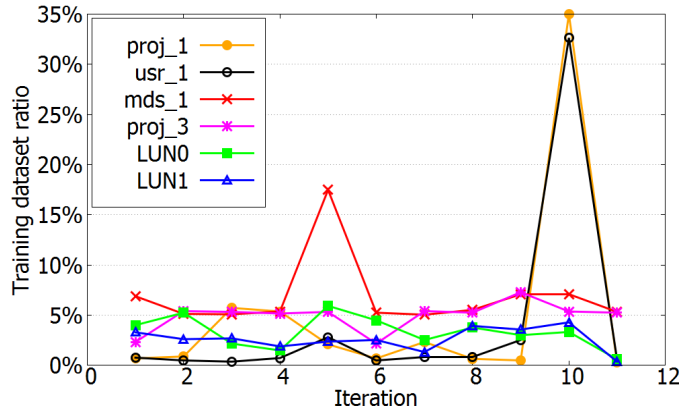


Fig. 8. Training dataset ratios of first eleven iterations for the K-SVM algorithm.

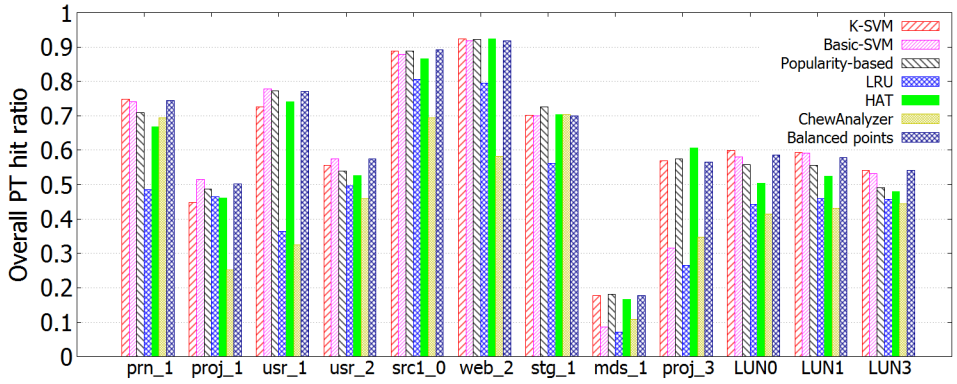


Fig. 9. The PT hit ratio comparisons between K-SVM and other algorithms.

4.2 K-SVM Results

To find how well the K-SVM algorithm is applied to the data migration problem of hybrid storage systems, we compare the performance of K-SVM algorithm with that of basic SVM, popularity-based, HAT and the performance of balanced points (discussed in Section 3.2). The system configurations are set to the same as the configuration in Section 3.2.

Training dataset ratios of the K-SVM algorithm for different iterations are observed. As shown in Figure 8, the K-SVM is capable of dynamically selecting training datasets based on the request access patterns. The performance and overhead comparisons are shown in Figure 9 and Figure 10 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 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 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 of traces. For traces

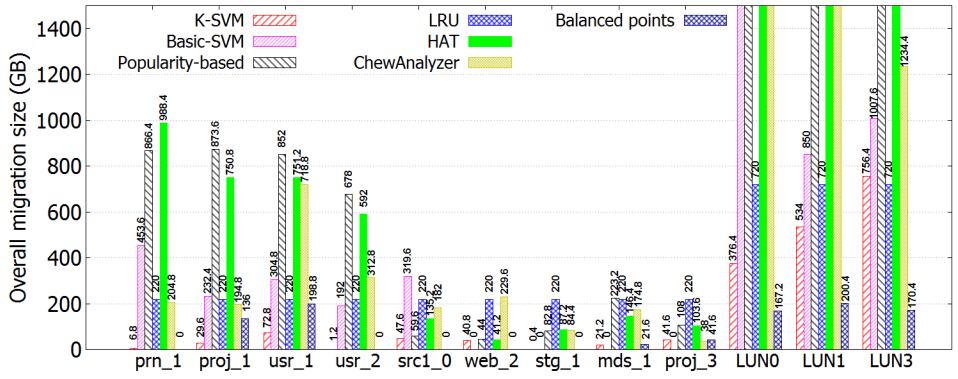


Fig. 10. The migration size comparisons between K-SVM and other algorithms.

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 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, K-SVM algorithm has slightly larger migration sizes compared with the results of balanced points, it remedies the issues described in Section 3.3. Moreover, the reduction of 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.

4.3 K-SVM Overhead Discussion

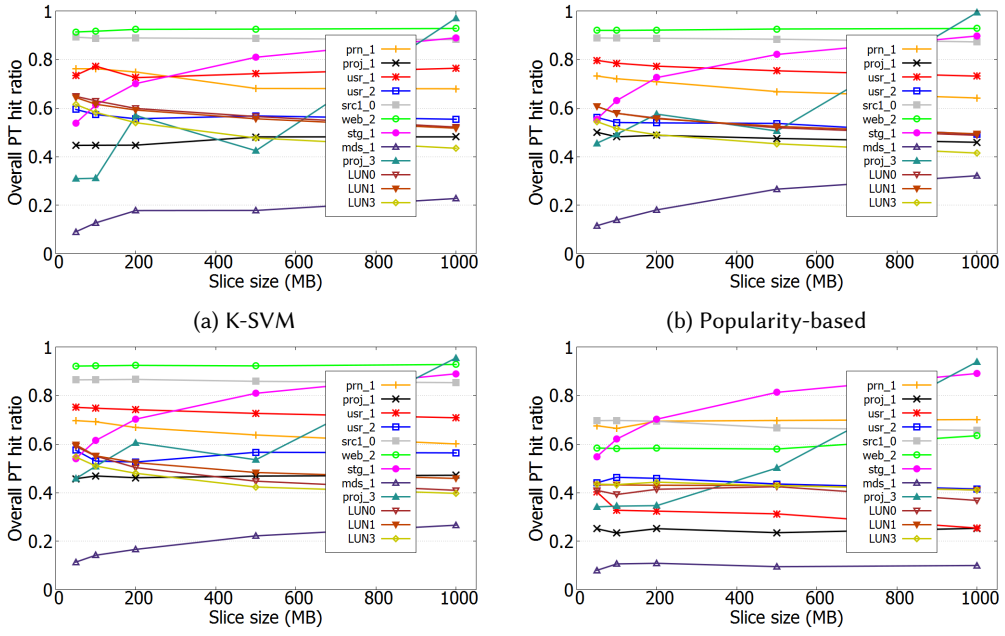
The overhead of the K-SVM scheme mainly comes from two aspects. One is the metadata overhead of recording collected trace information. The second one is the computation overhead of machine learning algorithms. Assume the total capacity of PT and CT tiers are 500 GB (PT: 100GB and CT: 400GB). The slice 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 training process. As seen in Table 4, 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 4. 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

5 EFFECT OF DIFFERENT PARAMETERS

After introducing the K-SVM algorithm and comparing it with other algorithms, we investigate the influence of various system parameters on the performance and overhead of these algorithms. The studied parameters include slice size (S_s), capacity ratio between SSD and the total capacity (r_{PT}), time interval (T) and available network/system bandwidth (BW). In the following subsections, we vary those parameters to investigate their relationships with the performance.



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

(c) HAT (d) ChewAnalyzer

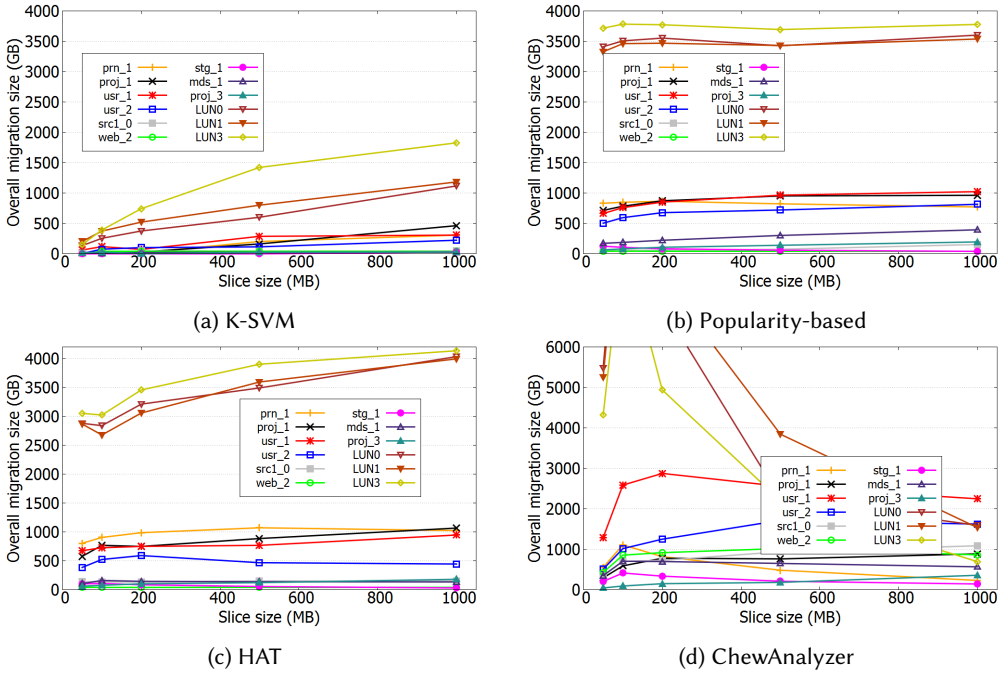
Fig. 11. The PT hit ratio with varying slice size.

5.1 Slice size

To investigate the effect of variable slice sizes on the PT hit ratio and total migration size, we make the other parameters fixed. The time intervals are set to one hour for the Systor'17 traces and 14 hours for the MSR traces. r_{PT} is set to 0.2 which is a typical ratio used in enterprise storage systems. The slice size is varied from 50MB to 1GB.

Figures 11 and 12 indicate the trends of PT hit ratio and total migration sizes for different migration algorithms with varying slice sizes respectively. Compared with the popularity-based, HAT and ChewAnalyzer algorithms, the K-SVM migration algorithm achieves similar PT hit ratios (better PT ratios than ChewAnalyzer) while obtains much smaller migration sizes. Compared between popularity-based, HAT and ChewAnalyzer algorithms, they achieve similar migration sizes and PT hit ratios in general. The ChewAnalyzer has much better performance on the LUN0, LUN1 and LUN3 traces with larger slice sizes and the HAT and Popularity-based schemes are better on smaller slice sizes in terms of migration size. The reason is that due to non-adaptive property, those three algorithms might be suitable for some specific access patterns and thus achieve much better performance for those access patterns.

Based on the performance results, we can categorize the traces into two groups. One group is that the PT hit ratio increases with the increase of slice size including traces, stg_1, mds_1 and proj_3. The traces in this group (in Table 3) have a typical feature which has a relatively small number of accesses for each slice. The small number of requests per slice causes a large variation during each period of time. As a result, the algorithms cannot precisely classify the characteristics of the traces. Thus, when increasing the slice size, the number of requests per slice is increased and then the algorithms can more precisely migrate the slices. Therefore, with increasing the slice size, the PT hit ratios of those traces increase as well. For the rest of the traces in the other group, they have



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

(c) HAT (d) ChewAnalyzer

Fig. 12. The migration size with varying slice size.

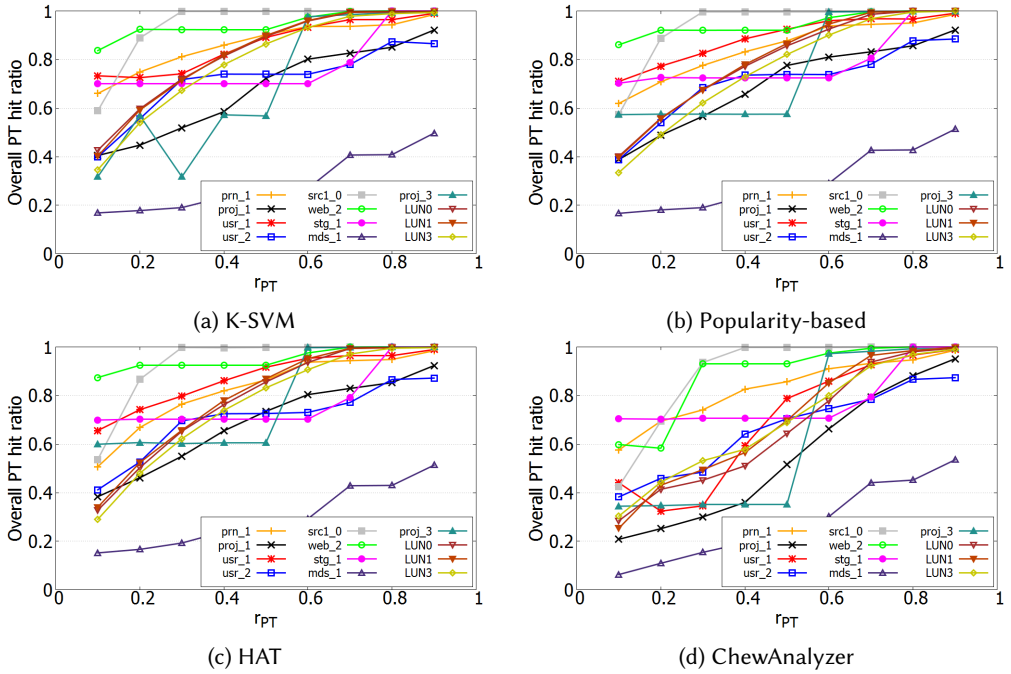
sufficient accesses in each slice. Therefore, the small slice sizes have the fine-granularity migration which helps increase the PT hit ratio. Moreover, with the increasing slice size, the migration size is also increased.

5.2 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 13 and 14, 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 PT capacity ratio on the migration size for each algorithm. For the popularity-based, HAT and ChewAnalyzer algorithms, they achieve similar trend of migration size, which is that the migration size first goes up and reaches to 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 deceases 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 mis-matched 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



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

(c) HAT (d) ChewAnalyzer

Fig. 13. The PT hit ratio with varying SSD (PT region) capacity.

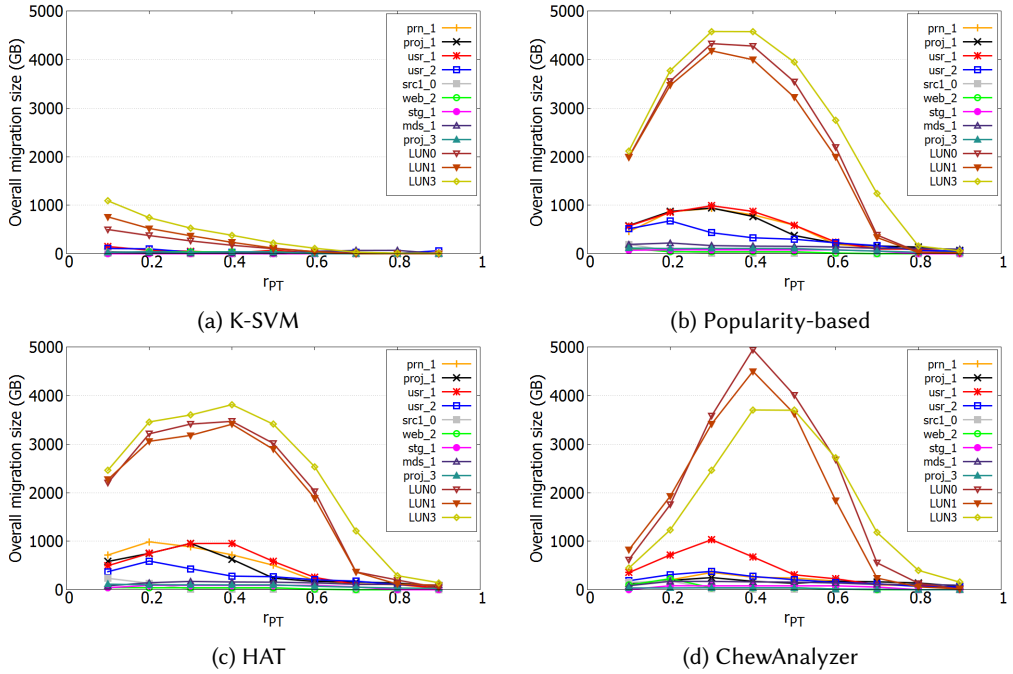
migration candidates and keeps the migration size small while maintaining similar PT hit ratios as other algorithms.

5.3 Time Interval T

We use some traces in Systor'17 and MSR to investigate the influence of the time interval T. In the Systor'17 traces the time interval T is varied from 0.5 to 4 hours. For MSR traces, the time interval T is varied from 3.5 to 56 hours which approximately matches the total number of requests in one time interval in the Systor'17 traces.

As shown in Figures 15 and 16, for the overall PT hit ratio, K-SVM, popularity-based, HAT and ChewAnalyzer algorithms are not sensitive to the time interval T and obtain almost the same PT hit ratios for Systor'17 traces with different time intervals. This is because the Systor'17 traces have similar access patterns with different time intervals. Thus the time interval does not have much effect on the PT hit ratio. Since a shorter time interval T means more times for classification and migration in the whole duration (36 hours), the migration sizes are increased with decreasing time interval T. Moreover, as seen in Figure 16a, there are obvious gaps between K-SVM and other algorithms. Thus, the K-SVM gets much less migration sizes than the popularity-based, HAT and ChewAnalyzer algorithms. In general, the overall migration size is decreased as increasing T since larger T results in less times of classifications/migrations. As a result, the total amount of migration overhead is reduced. For Systor'17 traces, the migration size has about 7- 9x difference between the smallest and the largest T. For MSR traces, the migration size has about 4 - 24x difference between the smallest and the largest T.

For the MSR traces, the usr_2 trace is sensitive to the time interval T sine the PT ratio variance can reach to about 25% for all those schemes. For the other two traces, they have flat trend with



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

(c) HAT (d) ChewAnalyzer

Fig. 14. The migration size with varying SSD (PT region) capacity.

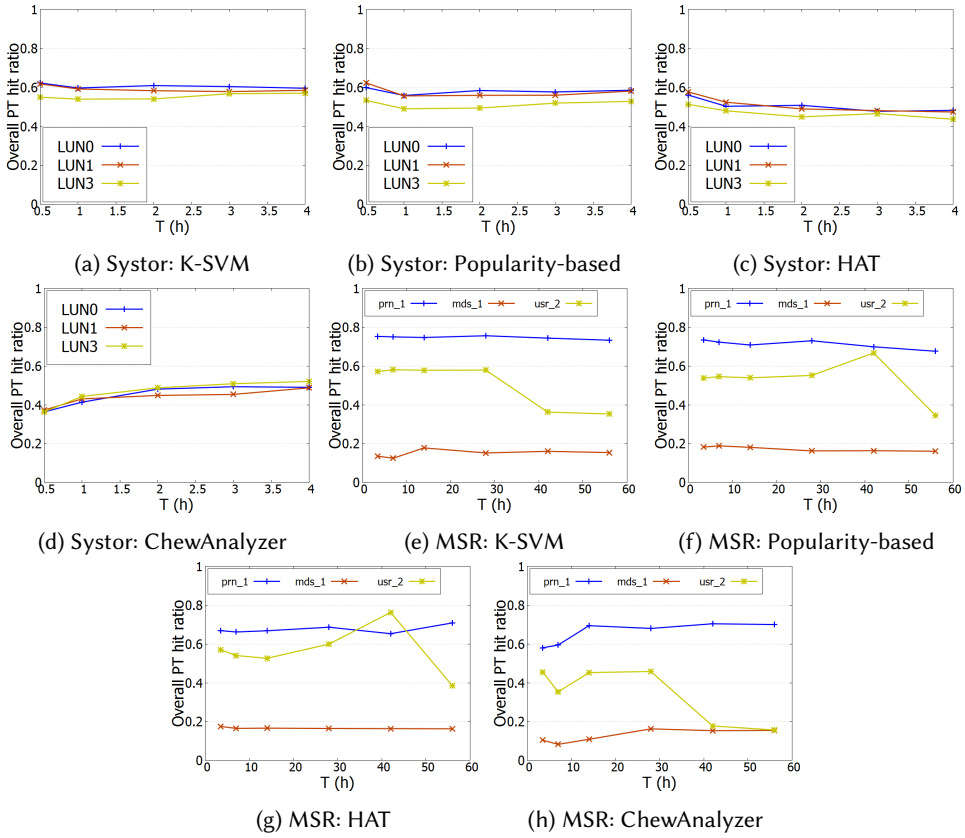
increasing T. The three algorithms achieve similar PT hit ratio except for the usr_2 at $T = 42$ with the K-SVM. The reason is that when $T = 42$ the total number of period is only 3 and the K-SVM might not be converged to a stable condition. Compared with the migration size, the conclusion is the same as the Systor'17 traces, the K-SVM algorithm always achieves smaller migration sizes than the popularity-based, HAT and ChewAnalyzer algorithms. On average, the K-SVM scheme obtains 4.22x, 6.14x and 3.49x less migration size than Popularity-based, HAT and ChewAnalyzer schemes, respectively.

5.4 Available Back-end Bandwidth

In a real environment, the applications with a high data access rate consume lots of available bandwidth. Therefore, due to limited available bandwidth, the system may not finish all migration classified by a migration algorithm. In this subsection, we investigate the available back-end bandwidth influence on the performance and overhead for different algorithms. The available back-end bandwidth can be regarded as the limited migration sizes in the system. The system configuration keeps the same as discussed in Section 5.1.

As seen in Figures 17 and 18, with very small available BW, the three algorithms have the same migration size. Also, three algorithms have similar PT hit ratio. This is because the K-SVM, popularity-based, HAT and ChewAnalyzer algorithms can always classify those candidates which are highly accessed in CT and less accessed slices in PT regions. Thus, with small available BW they only migrate those "obvious" migration candidates and obtain the same performance and overhead.

With increasing BW, the K-SVM, popularity-based, HAT and ChewAnalyzer algorithms start to classify different migration candidates and result in different PT hit ratios and migration sizes. As seen in Figure 18, the migration sizes of the popularity-based, HAT and ChewAnalyzer algorithms

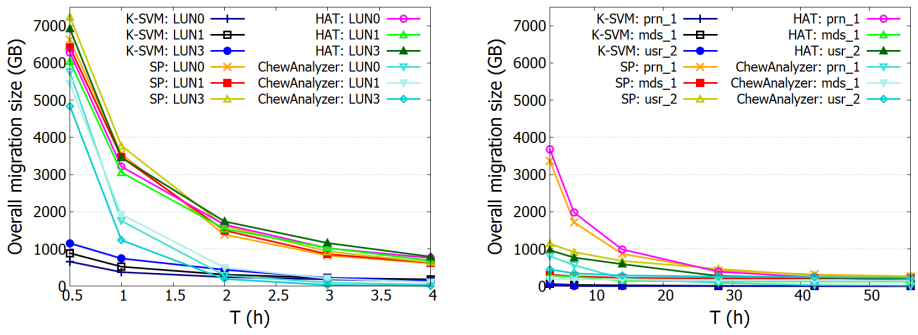


(a) Systor: K-SVM (b) Systor: Popularity-based (c) Systor: HAT

(d) Systor: ChewAnalyzer (e) MSR: K-SVM (f) MSR: Popularity-based

(g) MSR: HAT (h) MSR: ChewAnalyzer

Fig. 15. PT hit ratio with varying the time interval T.



(a) Systor'17 (b) MSR

Fig. 16. The migration size with varying the time interval T.

are tremendously increased. On the other hand the migration size of our proposed K-SVM only slightly increases while maintaining a similar PT hit ratio as the popularity-based algorithm. In

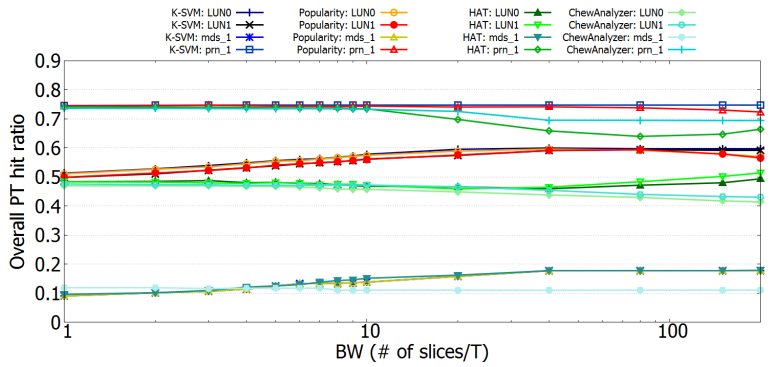


Fig. 17. The PT hit ratio with varying available back-end bandwidth.

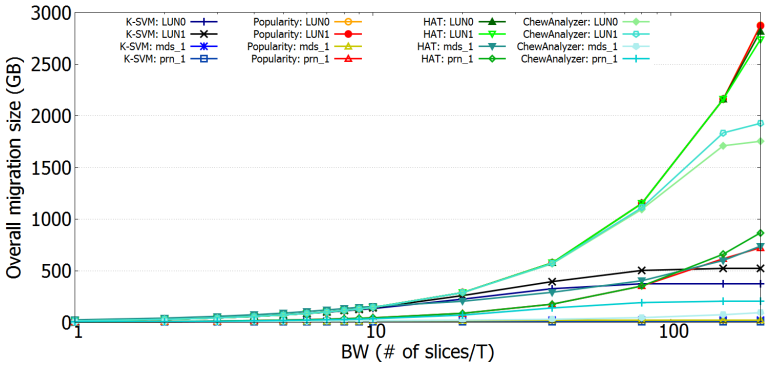


Fig. 18. The migration size with varying available back-end bandwidth.

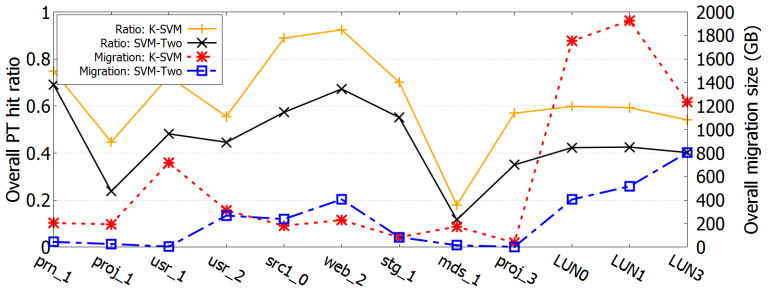


Fig. 19. Comparison between K-SVM and the SVM scheme with two factors.

summary, the lower available bandwidth causes a lower PT hit ratio for the K-SVM and popularity-based algorithms. With increasing available bandwidth, the K-SVM increases the PT hit ratios with only slightly increased total migration size.

5.5 Discussion of Different Design Factors

For block storage systems, limited information can be captured in the block layer including request size, I/O timestamp, write/read, and offset. The timestamp can be interpreted by the access density

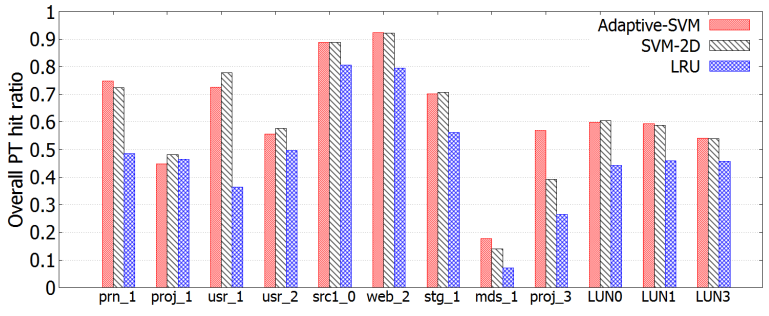


Fig. 20. The PT hit ratio comparison with K-SVM. SVM-2D and LRU algorithms.

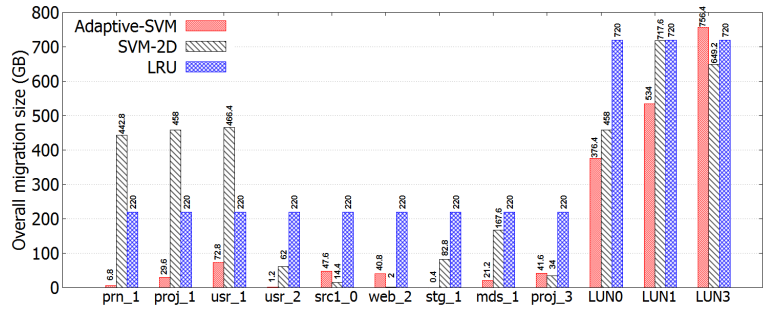


Fig. 21. The migration size comparison with K-SVM. SVM-2D and LRU algorithms.

which is used as one factor for the proposed scheme. For the issue presented in this paper, no matter read and write requests we always want to put highly frequently accessed data in the fast devices. So, we do not need to distinguish read and write in this work. The logical offset is used to compute the slice number in this work. The request size might have an influence on the overall performance. To investigate the influence of the request size, we make a comparison between one factor SVM (access density only) and two factor SVM (access density and request size). As shown in Figure 19, the K-SVM obtains much higher PT hit ratios than the SVM with two factors (18% higher on average). The K-SVM scheme achieves a little higher migration size for some traces. It means that the K-SVM can correctly migrate slices to achieve better hit ratios than the SVM with two factors. The reason is that the system uses the slice size with a larger value (hundred MBs to GBs) and thus the access density on the slice becomes more important than the request size. So, the K-SVM without considering the request size will achieve better high ratios than the SVM with two factors.

5.6 Effect on Frequency and Recency

In this subsection, we investigate the effect of frequency and recency on the performance of migration algorithm. Three algorithms are compared. The K-SVM algorithm considers the frequency. The LRU algorithm involves the recency. Based on the K-SVM algorithm, we extend the K-SVM with considering both frequency and recency, called SVM-2D. The basic idea of SVM-2D is similar to the K-SVM, which changes the x-axis to recency in Figure 2.

The results are shown in Figure 20 and Figure 21. According to the results, we can conclude that the LRU algorithm as discussed before has worst migration performance on both PT hit ratio and

Table 5. 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

migration size. Since the migration size is proportional to the number of periods as mentioned in Section 2, the LRU algorithm keeps the same amount of migration sizes for the MSR and Systor'17 traces. For the PT hit ratio, the LRU algorithm always migrates the least recently used slices and cannot reflect the characteristics of workloads. Therefore, it causes much lower PT hit ratios than the other two algorithms.

Comparing the K-SVM and SVM-2D, the results can be roughly categorized to three groups. For the first group, the K-SVM achieves similar PT hit ratio but much less migration sizes than that of SVM-2D algorithm among most of traces. The reason is that the recency degrades the impact of frequency on the classification. Thus, some un-necessary slices are migrated. For the second group, the K-SVM has higher or lower PT hit ratio and also obtains larger or smaller migration size than SVM-2D. As mentioned in Section 4.2, there is the trade-off between migration size and PT hit ratio and our K-SVM is much close to the "balanced points". For the third group, SVM-2D achieves similar PT hit ratios but smaller migration sizes than the K-SVM for traces src1_0, web_2 and LUN3. In these three traces, the access patterns might not be regular and make the K-SVM migrating larger size of slices. However, the SVM-2D with considering the recency mitigate the effect of irregular access patterns.

In summary, the K-SVM mostly achieves the best performance compared to SVM-2D and LRU algorithms. The analysis of the migration algorithms is that for the migration algorithms in the storage systems, unlike the cache policy, the migration and replacement do not happens immediately. The long period time can accumulate enough information to predict future access patterns and so the recency becomes not so important as the frequency.

6 A REAL 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 with 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 5. The total storage capacity in the test environment 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%. Four metrics are considered 1) the average latency as experienced by the host before migration (**pre-migration**); 2) latency spikes when data is actually migrated (**peak-migration**); 3) the average latency as experienced by the hosts after migration (**post-migration**); and 4) **Migration size**.

As seen in Figure 22, the system with the popularity-based algorithm moves about 0.95TB data per day and the system with the SVM algorithm only moves about 0.68TB data per day on average.

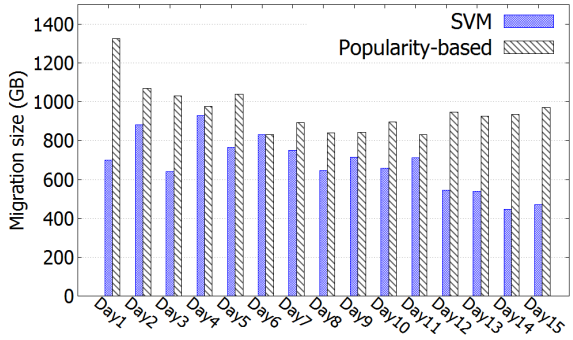
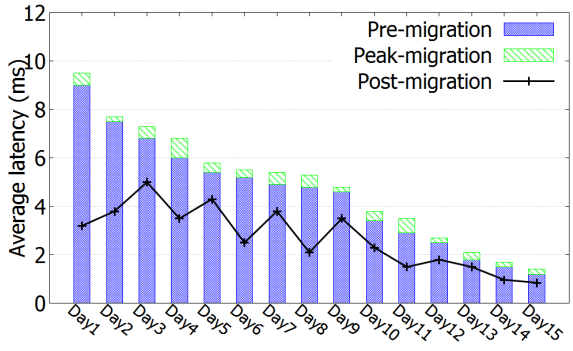
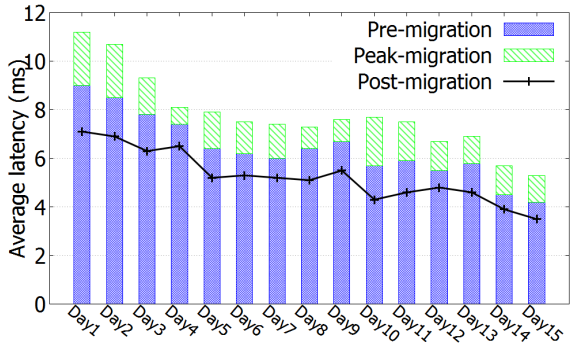


Fig. 22. Migration size comparison between SVM and popularity-based algorithms



(a) SVM



(b) Popularity-based

Fig. 23. Latency impact (prior, during and post migration)

Thus, the SVM algorithm achieves about 40% data migration reduction compared to the popularity-based algorithm. For the average latency in Figure 23, both algorithms are able to reduce the latency of the overall system after migrating. On average, the latency of the system with the SVM algorithm drops from 4.5ms to 2.7ms. While, the system with the SVM algorithm only achieves the drop from 6.4ms to 5.25ms. The SVM algorithm reduces the average latency about 70% compared to the

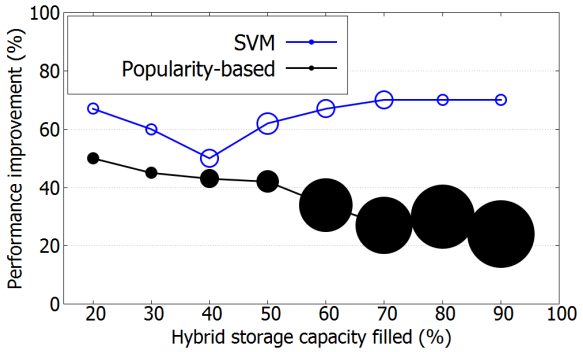


Fig. 24. Performance improvement and migration sizes with various storage capacities for SVM and popularity-based. (the sizes of bubbles indicate the migration size)

popularity-based algorithm. Moreover, the transient spikes in the popularity-based algorithm are as high as 27% on average, which is much larger than the spikes in the SVM algorithm (only 9.1%).

Additionally, we examined the impact of the storage capacity on both popularity-based and SVM algorithms. As shown in Figure 24, the SVM algorithm achieve 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.

7 RELATED WORK

A hybrid storage system combining SSDs and HDDs has the advantages of cost effectiveness, higher performance and longer endurance. Thus, it has gaining popularity and attracting research interests since the beginning of this decade [20, 24, 32]. With the advent of flash memory based SSDs now and Non-Volatile Memory (NVM) in the near future, research on caching and tiering algorithms to improve and deliver Quality of Service (QoS) of storage systems has been extensively conducted as well [8, 13, 18, 19, 22, 34, 38].

The data tiering management normally can be classified into two categories: file-level and block-level. For the file-level migration, the storage manager has the information about application files and thus it can precisely migrate data based on the characteristics of applications [14, 30, 40]. However, compared to the block-level migration, the file-level migration is less efficient and has a larger migration overhead due to its migration granularity and the migration decision to be done by file manager. For block-level migration, Guerra *et al.* [12] proposed the Extent-based Dynamic Tiering (EDT) tool that contains two components: a configuration adviser (EDT-CA) and a dynamic tier management (EDT-DTM). The EDT-CA determines the extent placement based on a fixed utility function. The EDT-DTM manages the extent placement and migration via monitoring active workloads. ExaPlan [16] achieves a low mean response time by using a queuing model. HybridStore [20] provides a cost-efficient storage configuration for specific workloads and is able to reduce the response time for the random-write dominant workloads. These studies have two major difference with our work. One is that they used simple heuristic methods to solve the migration problem. The other one is that they focus on three or more types of devices and also consider the prices of devices. In this paper, we exploit the possibility of using machine learning approaches.

8 CONCLUSION AND FUTURE WORK

By applying known machine learning approaches to 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 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 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.

In future work, we plan to extend the system with two tiers to multiple tiers. Thus, the classification issue will be changed to a multi-class classification problem and the tiering problem becomes more complicated. Therefore, we plan to explore other AI and machine learning approaches like neural networks which may produce better results than SVM.

REFERENCES

[1] [n. d.]. 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.

[2] 2007. Storage Performance Council(University of Massa-chusetts trace repository). http://traces.cs.umass.edu/index.php/Storage.

[3] I. Ahmad. 2007. Easy and Efficient Disk I/O Workload Characterization in VMware ESX Server. In *2007 IEEE 10th International Symposium on Workload Characterization*. 149–158. https://doi.org/10.1109/IISWC.2007.4362191

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

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

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

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

[8] A Elnably and P Varman. 2012. Application-sensitive qos scheduling in storage servers. In *ACM Symposium on Parallelism in Algorithms and Architecture*.

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

[10] Xiongzi Ge, Xuchao Xie, David HC Du, Pradeep Ganesan, and Dennis Hahn. 2018. 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)*. IEEE, 94–101.

[11] Ross Girshick. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 1440–1448.

[12] Jorge Guerra, Himabindu Pucha, Joseph S Glider, Wendy Belluomini, and Raju Rangaswami. 2011. Cost Effective Storage using Extent Based Dynamic Tiering.. In *FAST*, Vol. 11. 20–20.

[13] Ajay Gulati, Arif Merchant, and Peter J Varman. 2007. pClock: an arrival curve based approach for QoS guarantees in shared storage systems. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 35. ACM, 13–24.

[14] Dingshan He, Xianbo Zhang, David HC Du, and Gary Grider. 2006. Coordinating parallel hierarchical storage management in object-base cluster file systems. In *Proceedings of the 23rd IEEE Conference on Mass Storage Systems and Technologies (MSST)*.

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

[16] Ilias Iliadis, Jens Jelitto, Yusik Kim, Slavisa Sarafijanovic, and Vinodh Venkatesan. 2017. ExaPlan: Efficient Queueing-Based Data Placement, Provisioning, and Load Balancing for Large Tiered Storage Systems. *ACM Transactions on Storage (TOS)* 13, 2 (2017), 17.

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

[18] Magnus Karlsson, Christos Karamanolis, and Xiaoyun Zhu. 2005. Triage: Performance differentiation for storage systems using adaptive control. *ACM Transactions on Storage (TOS)* 1, 4 (2005), 457–480.

[19] Hyojun Kim, Sangeetha Seshadri, Clement L Dickey, and Lawrence Chiu. 2014. Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches. *ACM Transactions on Storage (TOS)* 10, 4 (2014), 15.

[20] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramaniam. 2011. HybridStore: A cost-efficient, high-performance storage system combining SSDs and HDDs. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*. IEEE, 227–236.

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

[22] Bingzhe Li, Chunhua Deng, Jinfeng Yang, David Lilja, Bo Yuan, and David Du. 2019. HAML-SSD: A Hardware Accelerated Hotness-Aware Machine Learning based SSD Management. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[23] Bingzhe Li, Yaobin Qin, Bo Yuan, and David J Lilja. 2017. Neural Network Classifiers Using Stochastic Computing with a Hardware-Oriented Approximate Activation Function. In *2017 IEEE 35th International Conference on Computer Design (ICCD)*. IEEE, 97–104.

[24] Zhichao Li. 2014. *GreenDM: A versatile tiering hybrid drive for the trade-off evaluation of performance, energy, and endurance*. Ph.D. Dissertation. The Graduate School, Stony Brook University: Stony Brook, NY.

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

[26] Nimrod Megiddo and Dharmendra S Modha. 2003. ARC: A Self-Tuning, Low Overhead Replacement Cache.. In *FAST*, Vol. 3. 115–130.

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

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

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

[30] Tracy F Sienknecht, Richard J Friedrich, Joseph J Martinka, and Peter M Friedenbach. 1994. The implications of distributed data in a commercial environment on the design of hierarchical storage management. *Performance Evaluation* 20, 1-3 (1994), 3–25.

[31] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.

[32] John D Strunk. 2012. Hybrid Aggregates: Combining SSDs and HDDs in a single storage pool. *ACM SIGOPS Operating Systems Review* 46, 3 (2012), 50–56.

[33] Giuseppe Vietri, Liana V Rodriguez, Wendy A Martinez, Steven Lyons, Jason Liu, Raju Rangaswami, Ming Zhao, and Giri Narasimhan. 2018. Driving cache replacement with ml-based lecar. In *10th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*.

[34] Hui Wang and Peter J Varman. 2014. Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation.. In *FAST*, Vol. 14. 229–242.

[35] Fenggang Wu, Bingzhe Li, Zhichao Cao, Baoquan Zhang, Ming-Hong Yang, Hao Wen, and David HC Du. 2019. ZoneAlloy: Elastic Data and Space Management for Hybrid {SMR} Drives. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*.

[36] Fenggang Wu, Baoquan Zhang, Zhichao Cao, Hao Wen, Bingzhe Li, Jim Diehl, Guohua Wang, and David HC Du. 2018. Data management design for interlaced magnetic recording. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*.

[37] Kan Wu, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2019. 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*.

[38] Ji Xue, Feng Yan, Alma Riska, and Evgenia Smirni. 2014. Storage Workload Isolation via Tier Warming: How Models Can Help.. In *ICAC*. 1–11.

[39] Jinfeng Yang and David J Lilja. 2018. Reducing Relational Database Performance Bottlenecks Using 3D XPoint Storage Technology. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 1804–1808.

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