Live Data Migration For Reducing SLA Violations In Multi-tiered Storage Systems

Jianzhe Tai
Northeastern University
jtai@ece.neu.edu

Bo Sheng
University of Massachusetts Boston
shengbo@cs.umb.edu

Yi Yao Ningfang Mi
Northeastern University
{yyao, ningfang}@ece.neu.edu

***Abstract*—Today, the volume of data in the world has been tremendously increased. Large-scaled and diverse data sets are raising new big challenges of storage, process, and query. Tiered storage architectures combining solid-state drives (SSDs) with hard disk drives (HDDs), become attractive in enterprise data centers for achieving high performance and large capacity simultaneously. However, how to best use these storage resources and efficiently manage massive data for providing high quality of service (QoS) is still a core and difficult problem. In this paper, we present a new approach for automated data movement in multi-tiered storage systems, which lively migrates the data across different tiers, aiming to support multiple service level agreements (SLAs) for applications with dynamic workloads at the minimal cost. Trace-driven simulations show that compared to the no migration policy, LMST significantly improves average I/O response times, I/O violation ratios and I/O violation times, with only slight degradation (e.g., up to 6% increase in SLA violation ratio) on the performance of high priority applications.**

***Index Terms*—multi-tiered storage systems; data migration; service level agreement (SLA); bursty workloads;**

I. Introduction

The volume of data in today's world has been tremendously increased. For example, Facebook revealed that its system each day processes 2.5 billion pieces of content and more than 500 terabytes of data, including 83 million pictures. Being one of the largest databases in the world, Google processes more than 25 petabytes of data per day. As more and more people use different types of devices such as smartphones and laptops, data comes from everywhere, including body sensors for collecting medical data and GPS devices used to gather traffic information. Such massive and diverse data sets will then lead to challenging issues for system designers to address. One of the most important issues is where to store these gigantic data sets and how to make them accessible.

Nowadays, flash-based solid-state drives (SSDs) have gained prominence in enterprise arrays and successfully been used as a replacement of HDDs because of significant performance improvement and low energy consumption. Yet, given the fact that SSDs are more expensive per gigabyte (GB) and have a limited number of writes over the life time, a multi-tiered storage platform, which combines SSDs with traditional HDDs (e.g., FC/SAS and/or SATA), has become an industrial standard building block in an enterprise data center. Nevertheless, how to best use of these hybrid storage resources for managing massive data and providing high quality of service (QoS) is still a core and difficult problem due to the following two issues.

First, modern enterprise data centers often provide shared storage resources to a large variety of applications which might demand for different performance goals such that different service level agreements (SLAs) have to be met. Hence, these data centers need to be SLA aware in the management of shared storage resources. Second, an effective resource manager needs to dynamically adjust its policy according to different application workloads. In practice, workloads may change over time. Bursty workloads and traffic surges are often found in enterprise data centers, which inevitably causes disastrous SLA violations, performance degradation and even service unavailability.

To address the above issues, we present a new approach, named LMST, for automated data movement in multi-tiered storage systems. LMST attempts to lively migrate the data across different storage tiers, aiming to guarantee multiple SLA requirements for applications under dynamic workloads. Via trace-driven simulations, we show that LMST efficiently utilizes the high-performance devices (e.g., SSDs) to improve the QoS for loose-SLA applications under bursty workloads and meanwhile provide the performance guarantees for applications with strict SLAs.

This paper is organized as follows. Section II demonstrates the architecture of a multi-tiered storage system. Section III presents the LMST algorithm for automated data migration in multi-tiered storage systems. Section IV evaluates the effectiveness and robustness of LMST using trace-driven simulations. Section V gives an overview of the related works. Finally, we draw conclusions in Section VI.

II. System Architecture

We first present an overview of a multi-tiered storage system which is considered in this paper. As shown in Fig. 1, the system consists of four main components: application, server, logical unit (LUN), and back-end storage pool.

Specifically, the *application* component in the top layer is used to represent the applications who can access the shared storage resources in data centers. We classify the applications into several categories according to their SLA requirements. Each application with its own I/O workload specifications is assigned to a virtual machine (VM) which provides a virtual disk to support the associated SLA requirement. The hypervisor, as a virtual machine monitor (VMM) in the *server* component, supports multiple VMs to access the shared *back-end storage pool* and allocates virtualized disk resources among VMs to achieve their different performance goals.

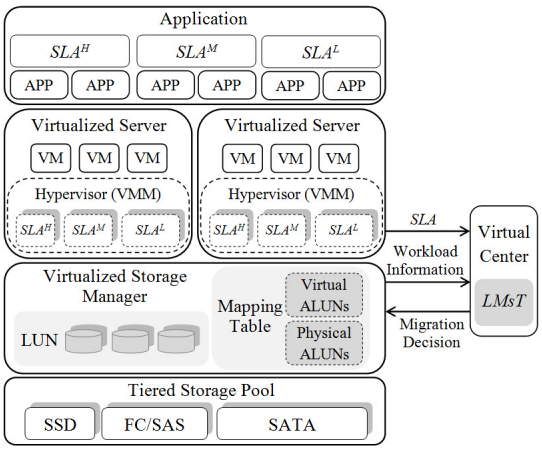

Fig. 1. The structure of a multi-tiered storage system.

The *LUN* component abstracts the fundamental storage pool and supports the storage virtualization by building a mapping table to connect the virtual disk resources with the physical disk resources. Therefore, the LUN component hides the information of the underlying hardware devices to applications while enables multiple applications to share virtualized storage resources without noticing the accesses and the contentions from the others.

Through storage virtualization in the LUN component, the storage pool can provide the fundamental disk resources as the module of allocation unit (ALUN) which is set to 1GB as the minimal capacity/migration unit for thin-provisioning in sub-LUN level. Via the mapping table, each virtual ALUN in the hypervisor is then dedicated to a physical ALUN in the storage pool. The virtual center (e.g., VMware vCenter [1]) is responsible to analyze the resource usage in virtualization layers and to deploy tools for resource management. We remark that our new migration method can be implemented as a new module in the virtual center, which is able to use all these information to make the decisions for data migration, and then send the decisions back to the virtualized storage manager which will execute the corresponding migration procedure.

III. Migration Algorithm LMsT

In this section, we present our new data migration algorithm LMsT. Our objective is to improve the system performance in terms of I/O response time while the application SLAs are still satisfied after the migration processes. In the rest of this section, we first present a formulation for data migration and then show how LMsT addresses the formulated problem in detail. Table I gives the notations that are used in this paper.

A. Overview and Problem Formulation

We use *data temperature* as an indicator to classify data into two categories according to their access frequency: *hot data* has a frequent access pattern and *cold data* is occasionally queried. We also consider a multi-tier storage structure consisting of two tiers, i.e., high performance tier equipped with SSDs and low performance tier using FCs. Because of the high hardware cost, high performance tier has a much smaller capacity than low performance tier. We note that our solution can be easily extended for data categories with more temperature levels and for storage systems with more than two tiers.

TABLE I
Notations in this paper.

$A_i,\ i \in [1,n]$	n ALUNs.
$D_j,\ j \in [1,m]$	m disks.
$x_{i,j} \in \{0,1\}$	indicator of association between A_i and D_j.
λ_{A_i} or λ_{D_j}	I/O arrival rates of A_i or D_j (KB/ms).
μ_{D_j}	average service rate of disk D_j (KB/ms).
s_{D_j}	average I/O size on disk D_j (KB).
SLA_k	the kth SLA requirement (ms).
$Q_{j,k}$	the kth logical buffer on disk D_j with SLA_k.
$y_{i,k} \in \{0,1\}$	indicator of association between A_i and SLA_k.
t_{win}	duration of a time window (ms).
t_{mgt}	time duration of the migration process (ms).

In practice, high performance tier is often reserved for applications which have strictly high SLA requirements. However, from the perspective of improving the overall system performance, high performance tier is also expected to host hot data regardless of the data owner's SLA. To best coordinate between the SLA-based and the performance-based resource allocations, LMsT automatically reallocates the data across multiple tiers of drives based on data temperature and SLA requirements. In designing this new algorithm, we define the following rules that allow LMsT to efficiently utilize the high performance SSD-tier.

- **R1:** Latency-sensitive applications with strict SLAs should always been served in SSD-tier while the applications with loose SLAs should be initially served in HDD-tier.
- **R2:** Once an application with loose SLA suffers bursty workloads, its hot ALUNs should be migrated to SSD-tier in order to mitigate the burdens in HDD-tier and avoid SLA violations.
- **R3:** Extra I/Os caused by the migration process should not violate SLAs of any applications at both the source and the destination devices.
- **R4:** The newly migrated hot data in SSD-tier should not bring additional SLA violations to latency-sensitive applications with strict SLAs.

In particular, assume there are n ALUNs $\{A_1, A_2, \ldots, A_n\}$ across m disks $\{D_1, D_2, \ldots, D_m\}$. Let $x_{i,j} \in \{0,1\}$ indicate the association between A_i and D_j, i.e., $x_{i,j} = 1$ if ALUN A_i is hosted on disk D_j. Apparently, we have $\forall i, \sum_j x_{i,j} = 1$. In our solution, an ALUN is the minimum storage unit to be migrated. LMsT monitors the workload and the performance for each ALUN and each disk in a predefined time window t_{win} (e.g., 20 minutes in our experiments[1]), to assist our migration decision.

[1]We remark that the setting of t_{win} depends on how frequently the workload changes. If the workload changes fast, then a small t_{win} is preferred, vice versa.

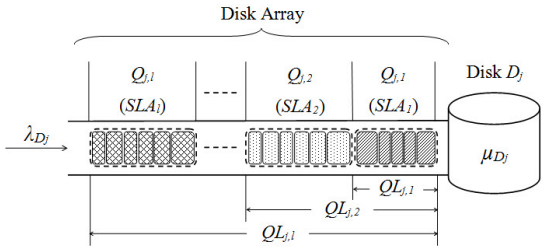

Fig. 2. The profile of logical buffers and disk array.

Let λ_{A_i} and λ_{D_j} represent the arrival rates (KB/ms) of ALUN A_i and disk D_j, respectively. Then, we have

$$\lambda_{D_j} = \sum_i x_{i,j} \cdot \lambda_{A_i}, \quad (1)$$

$$\lambda_{A_i} = m(\lambda_{A_i}) + \alpha \cdot \Delta(\lambda_{A_i}), \quad (2)$$

where $m(\lambda_{A_i})$ and $\Delta(\lambda_{A_i})$ represent the mean and the standard deviation of λ_{A_i}, and α is a tuning parameter for conservation. We further classify I/Os into four categories, i.e., sequential read (SR), random read (RR), sequential write (SW), and random write (RW) and let $\mu_{D_j}^{SR}$, $\mu_{D_j}^{RR}$, $\mu_{D_j}^{SW}$, and $\mu_{D_j}^{RW}$ denote the corresponding average service rates for these patterns, respectively. Then, the overall average service rate for disk D_j can be estimated as,

$$\mu_{D_j} = P_{SR} \cdot \mu_{D_j}^{SR} + P_{RR} \cdot \mu_{D_j}^{RR} + P_{SW} \cdot \mu_{D_j}^{SW} + P_{RW} \cdot \mu_{D_j}^{RW}, \quad (3)$$

where P_{SR}, P_{RR}, P_{SW}, and P_{RW} represent the fraction of each category. We also let s_{D_j} denote the average I/O size (KB) for each disk D_j.

In addition, assume each disk D_j has a single I/O queue consisting of l consecutive "logical" buffers $\{Q_{j,1}, Q_{j,2}, \ldots, Q_{j,l}\}$ and each $Q_{j,k}$ serves I/Os with a different SLA requirement, SLA_k (ms), see Figure 2. Without loss of generality, we assume $\forall i < k,\ SLA_i < SLA_k$. Let $y_{i,k} \in \{0, 1\}$ indicate if A_i is associated with SLA_k. Thus, A_i belongs to the buffer $Q_{j,k}$ if $x_{i,j} \cdot y_{i,k} = 1$.

B. Migration Candidate Selector

Now, we present how to select candidate ALUNs for migration. In overall, there are two phases in our scheme. The first one is *Selection Phase* where we choose a set of potential migration candidates based on the workloads of each ALUN and the performance of each disk. Each potential candidate is represented by a pair value (A_i, D_j) indicating a migration of ALUN A_i to D_j ($x_{i,j} = 0$). In the second phase of *validation*, we carefully examine each migration candidate, quantify the benefits, and estimate the risk of SLA violations. A subset of feasible candidates will be selected for actual migration.

Selection Phase: There are two types of effective migrations that the system can benefit from. First, if an ALUN hosts hot data in low performance tier, it should be migrated to high performance tier for improving the performance. We call this migration as *forward migration*. Second, if the workload of an ALUN from loose-SLA application becomes cold in high performance tier, we may migrate that ALUN back to low performance tier in order to release the space in high performance tier. Such a migration is then called *backward migration*.

We define two thresholds of I/O workloads τ_h and τ_l ($\tau_l < \tau_h$) for selecting eligible ALUNs for migration as follows. For an ALUN A_i, if its average workload $\lambda_{A_i} > \tau_h$, we consider the data hosted on A_i is hot. If A_i resides in low performance tier, it would be beneficial for the system to migrate it to high performance tier. Similarly, if an ALUN's workload is less than the lower threshold, i.e., $\lambda_{A_i} < \tau_l$, the data stored on A_i is regarded as cold. We then move that particular ALUN A_i to low performance tier to release resources in high performance tier if A_i is now allocated in high performance tier but belongs to an application with loose SLA. By this way, we find a set of ALUNs that are eligible for either forward or backward migrations.

Furthermore, destination disk D_j for each eligible ALUN to migrate to is found such that D_j has the lowest load among those disks that can provide at least one available ALUN space. Finally, the selection phase yields a set of migration candidates (A_i, D_j) for the next validation phase.

Validation Phase: In validation phase, we quantify each migration candidate (A_i, D_j) through the following two conditions: (1.) SLAs have to be met; (2.) average I/O response time is expected to be decreased (for forward migration). A candidate is validated for migration only if both of these two conditions are satisfied. In the next, we quantify and analyze these performance metrics.

1) SLA Constraint: Recall that in our model, each disk array keeps multiple logical buffers and each buffer servers I/Os with a different SLA as shown in Figure 2. Upon the arrival of an I/O request, the I/O scheduler inserts it into a particular logical buffer which contains the requests having the same SLA requirement as the arriving one. While, within each buffer, all requests are scheduled based on First-In-First-Out (FIFO) discipline. Specifically, each buffer $Q_{j,k}$ can just hold a limited number of I/O requests in order to avoid introducing heavy loads to disk D_j and causing additional SLA violations.

Thus, for each logical buffer $Q_{j,k}$, we define $ML_{j,k}$ as the maximal queue length that the disk j can handle without causing any SLA violations,

$$ML_{j,k} = SLA_k \cdot \mu_{D_j}.$$

Additionally, we use $QL_{j,k}$ to denote the accumulated average queue length of logical buffers from $Q_{j,1}$ to $Q_{j,k}$. Let $\overline{\lambda}_{j,k}$ represent the overall arrival rates of the ALUNs whose SLAs are equal to or smaller than SLA_k in disk j,

$$\overline{\lambda}_{j,k} = \sum_{t=1}^{k} \sum_{i=1}^{n} x_{i,j} \cdot y_{i,t} \cdot \lambda_{A_i}.$$

Thus, using Little's Law, $QL_{j,k}$ can be expressed as

$$QL_{j,k} = f(\overline{\lambda}_{j,k}) = \frac{\overline{\lambda}_{j,k}}{\mu_{D_j} - \overline{\lambda}_{j,k}} \cdot s_{D_j}. \quad (4)$$

According to the definitions, $QL_{j,k} \leq ML_{j,k}$.

With the above analysis, we check the following two rules for each migration candidate (A_i, D_j),

$$\lambda_{D_j} + \lambda_{A_i} < \mu_{D_j}, \quad (5)$$

$$ML_{j,k} \geq QL'_{j,k} = f(\overline{\lambda}_{j,k} + \lambda_{A_i}), \text{ for } y_{i,k} = 1. \quad (6)$$

The first rule requires the total arrival rate on the destination disk D_j to be less than the processing rate μ_{D_j}. Similarly, in order to process migration, the arrival rate of the source disk to which A_i belongs should also be less than its processing rate. The second rule is for the particular logical buffer with the corresponding SLA that A_i belongs to. After migration, the new queue length of $Q_{j,k}$ should not exceed the maximal limit $ML_{j,k}$.

2) Response Time Constraint: Now, we turn to the performance constraint in terms of I/O response time for validating migration candidates. Basically, we estimate the I/O response time of both the source and the destination disks under the policies with and without migration and then evaluate the benefit (or the penalty) of each migration candidate.

For a migration candidate (A_i, D_j), assume A_i is currently hosted on disk D_k, i.e., $x_{i,k} = 1$. Let $\lambda'_{D_k}, \lambda'_{D_j}$ and λ'_{A_i} represent the workloads of D_k, D_j, and A_i in the next time window, respectively. Additionally, let t_{mgt} be the time duration to process a live migration ($t_{mgt} < t_{win}$) and $\Delta\lambda$ be the extra transfer rate for serving migration I/Os during the migration process. Assume if validated, the migration (A_i, D_j) will be launched at the current window. With this particular migration, for both the source disk D_k and the destination disk D_j, the workloads during t_{mgt} of the current window become $\lambda_{D_k} + \Delta\lambda$ and $\lambda_{D_j} + \Delta\lambda$, respectively. Additionally, in the next time window, their new workloads will be $\lambda'_{D_k} - \lambda'_{A_i}$ and $\lambda'_{D_j} + \lambda'_{A_i}$, respectively.

Based on the Little's Law, we can calculate the average response time RT_j of disk D_j as follows,

$$RT_j = g(j, \lambda_{D_j}) = \frac{s_{D_j}}{\mu_{D_j} - \lambda_{D_j}}. \quad (7)$$

With Eq.(7), we can evaluate the average I/O response time of both the source and the destination disks in three periods, i.e., before, during and after the migration process. Let $RT_{k/j}(A_i, D_j)$ and $RT'_{k/j}(A_i, D_j)$ be the average response times of the source disk D_k (or the destination disk D_j) under the policies with and without a particular migration (A_i, D_j), respectively, and $\overline{RT}_{k/j}(A_i, D_j)$ be the relative benefit (or penalty) in terms of response time. We then have the following equations:

$$RT_k(A_i, D_j) = (g(k, \lambda_{D_k}) + g(k, \lambda_{D'_k})) \cdot t_{win}, \quad (8)$$

$$RT'_k(A_i, D_j) = g(k, \lambda_{D_k}) \cdot (t_{win} - t_{mgt}) + g(k, \lambda_{D_k} + \Delta\lambda) \cdot t_{mgt} + g(k, \lambda'_{D_k} - \lambda'_{A_i}) \cdot t_{win}, \quad (9)$$

$$\overline{RT}_k(A_i, D_j) = \frac{RT'_k(A_i, D_j) - RT_k(A_i, D_j)}{RT_k(A_i, D_j)}, \quad (10)$$

$$RT_j(A_i, D_j) = (g(j, \lambda_{D_j}) + g(j, \lambda_{D'_j})) \cdot t_{win}, \quad (11)$$

$$RT'_j(A_i, D_j) = g(j, \lambda_{D_j}) \cdot (t_{win} - t_{mgt}) + g(j, \lambda_{D_j} + \Delta\lambda) \cdot t_{mgt} + g(j, \lambda'_{D_j} + \lambda'_{A_i}) \cdot t_{win}, \quad (12)$$

$$\overline{RT}_j(A_i, D_j) = \frac{RT'_j(A_i, D_j) - RT_j(A_i, D_j)}{RT_j(A_i, D_j)}. \quad (13)$$

The response time constraint is designed to compare the overall improvement in average response time to a threshold $e\%$. The migration candidate (A_i, D_j) is validated only if the following condition is satisfied.

$$\frac{\overline{RT}_k(A_i, D_j) + \overline{RT}_j(A_i, D_j)}{2} > e\% \quad (14)$$

In summary, we defined two sets of migration constraints, related to *SLA* and *performance* in our migration policy, LMsT, for evaluating each migration candidate. Once a candidate is validated, the corresponding forward or backward migration process can be actually performed by LMsT.

IV. Performance Evaluation of LMsT

We use representative case studies to evaluate LMsT's effectiveness. A trace-driven simulation model has been built to emulate a multi-tier storage system as shown in Fig. 1. Without loss of generality, we assume that in our model the application components have two priority levels with different SLA requirements such that the SLAs of high and low priority applications are equal to $SLA^H = 1$ms and $SLA^L = 20$ms, respectively. We also assume two tiers of disk drives in the storage pool, i.e., SSD and FC. We remark that the number of disk drives in each tier is fixed in all the experiments. The device parameters of these two tiers are shown in Table II. Initially, the virtual ALUNs of applications with strict SLAs (i.e., SLA^H) are all mapped to SSDs, whereas FCs are initially assigned to low priority applications with SLA^L.

TABLE II
Device parameters of two tiers

Disk Type	Disk Number	Total Capacity	Service Rate
SSD	2	40GB	500MB/s
FC	5	100GB	160MB/s

Consider 7 applications (2 with SLA^H and 5 with SLA^L) to access a 70GB data set. In average, each application requires 10 virtual ALUNs and the capacity of such a virtual ALUN is 1GB. We then generate an I/O stream for each virtual ALUN such that there are totally 30 time periods and each lasts around 20 minutes. The specifications of a request in such an I/O stream include I/O arrival time, I/O address and I/O size, where I/O address is uniformly distributed within an ALUN while I/O size is drawn from an exponential distribution with mean of 100KB.

In order to evaluate LMsT under bursty workloads, we further inject bursty periods randomly into each I/O stream such that a time period can be marked as either "idle" or

"bursty", as shown in Fig. 3. We then generate I/O inter-arrival times in an idle period using an exponential distribution with mean rate of 10 KB/ms (resp. 5 KB/ms) for high (resp. low) priority applications; while I/O arrival process of a bursty period is drawn from a 2-state Markov-Modulated Poisson Process (MMPP) with mean arrival rate equal to 20 KB/ms (resp. 10 KB/ms) for high (resp. low) priority applications.

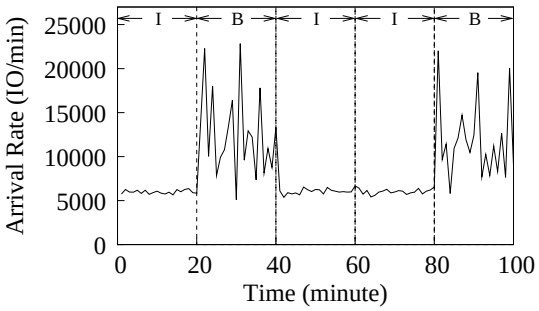

Fig. 3. Number of I/Os per minute in a virtual ALUN.

Fig. 4 depicts the performance results under NMsT and LMsT, where NMsT (i.e., no migration process) is used as the base case to normalize LMsT's performance. In this set of results, we measured the mean I/O response times[2] (M_Resp), the fraction of I/Os (V_Ratio) whose response times exceed the predefined SLAs, and the mean violation times (V_Time) that are the difference between the actual I/O response times and the predefined SLAs.

We observe that LMsT significantly improves the overall performance in terms of I/O response time, I/O violation ratio and I/O violation time. Faster I/O response time is achieved under LMsT, see Fig. 4. This is because LMsT makes a better use of SSDs by migrating all validated bursty traffic from FCs to SSDs. Consequently, LMsT significantly reduces the I/O violation ratio (V_Ratio) and I/O violation time (V_Time) by 42% and 40%, respectively. More importantly, LMsT always gives lower priority to migration I/Os as well as new application I/Os which are migrated from FCs. Thus, I/Os from high priority applications in SSDs are still guaranteed to meet the corresponding SLAs.

To further investigate the performance impacts of migrations on each application, we preset their average I/O response times and I/O violation ratios under both NMsT and LMsT in Table III. We observe that all low priority applications (i.e., $App3, ..., App7$) obtain tremendous performance improvement, experiencing lower response times and less violation ratios, and thereby receiving high QoS. Moreover, the performance of high priority applications (i.e., $App1$ and $App2$) keeps almost the same despite a very slight degradation due to the extra migrated I/Os.

In our work, we have also done sensitivity analysis on different system workloads (e.g., system loads and bursty profiles) and investigate the impacts of key parameters in migration constraints such as α of the SLA constraint. Due to the limited space, we omit those results in this paper but refer the reader to [2] for the details. These results further validate the effectiveness and robustness of our LMsT policy.

[2]An I/O request response time is measured from the moment when an I/O request is submitted to the moment when that I/O request finishes.

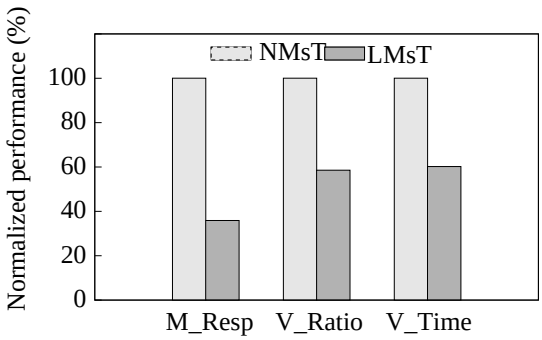

Fig. 4. Performance results under NMsT and LMsT.

V. Related Work

As enterprises consolidate a variety of applications that require different service levels, it becomes an urgent demand to build a multi-tiered storage platform for providing different levels of service in the storage domain [3]. *Storage tiering* techniques are introduced to dynamically deliver appropriate resources to the business, targeting at performance improvement, cost reduction and management simplification. Because of its significant importance, the technology of storage tiering has been recognized by ESG's 2011 Spending Intentions Survey [4], as one of the top 10 planned storage investments in the next couple years. Many industrial companies have already developed their own automatic tiering technologies and released the relative products, such as IBM Easy Tier for DS8000 [5], EMC Fully Automated Storage Tiering (FAST) for Celerra [6], and HP Adaptive Optimization for 3PAR [7].

A large literatures on storage management have been developed for the years. Recently, [8], [9], [10], [11], [12], [13], [14], [15] proposed several new techniques (algorithmic or theoretical) to explore the effective data migration in storage systems. For example, [8], [9] investigated the idea of using edge-coloring theory for data migration and achieved a near-optimal migration plan by using polynomial-time approximation algorithms. Triage, an adaptive controller, has been proposed in [11] to address the problem of performance isolation and differentiation in a consolidated data center. By throttling storage access requests, Triage ensures high system availability even under overload conditions. Later, [10] focused on minimizing the overhead of data migration by automatically detecting hotspots and reconfiguring the system based on the bandwidth-to-space ratio. [15] proposed a dynamic tier manager, named EDT-DTM, performing dynamic extent placement. However, we argue that none of the existing studies take account of both the on-the-fly migration penalties and the various application SLAs for data migration in multi-tiered storage systems.

TABLE III
EACH APPLICATION'S PERFORMANCE UNDER NMST AND LMST.

Capacity 70GB 50% Burst		High Priority		Low Priority				
		App1	App2	App3	App4	App5	App6	App7
NMST	M_Resp (ms)	1.30	1.27	382.29	366.60	368.52	350.12	380.55
	V_Ratio (%)	7.06	6.90	41.75	40.37	41.65	40.28	40.59
LMST	M_Resp (ms)	1.34	1.32	131.01	137.75	139.53	125.49	124.53
	V_Ratio (%)	7.73	7.53	21.55	22.36	22.28	19.94	19.34

A cost model [12] has been developed to solve the problem of efficient disk replacement and data migration in a polynomial time. [13] implements the QoS guarantee of performance impact on foreground work by leveraging a control-theoretical approach to dynamically adapt migration speed. [14] proposed a lookahead data migration algorithm for SSD-enabled multi-tiered storage systems, where the optimal lookahead window size is determined to meet the workload deadlines. However, the work [14] assumes that the I/O profile exhibits a cyclic behavior and does not consider different application SLAs in their algorithm.

VI. CONCLUSION

In this paper, we proposed LMST, a live data migration algorithm for efficiently utilizing the shared storage resources and meeting various application SLAs in a multi-tiered storage system. We have shown that bursty workloads in storage systems can deteriorate system performance, causing high I/O latency and large numbers of SLA violations in low performance tiers. In order to mitigate such negative effects, hot data that are associated with those bursty workloads should be migrated to high performance tiers. However, extra I/Os due to data migration as well as the newly migrated bursty workloads can incur additional SLA violations to high priority applications in high performance tiers.

Therefore, we designed LMST to counteract the impacts of burstiness by efficiently utilizing the high-performance devices, and to minimize the potential delays to latency-sensitive applications. Trace-driven simulations have been conducted to evaluate the performance of our new LMST policy. Compared to the no migration policy, LMST significantly improves average I/O response times, I/O violation ratios and I/O violation times by migrating all validated bursty traffic from FCs to SSDs. More importantly, under LMST, the extra-migrated I/Os only cause a very slight degradation (e.g., up to 6% increase in SLA violation ratio) on the performance of high priority applications.

ACKNOWLEDGMENT

This work was partially supported by NSF grant CNS-1251129 and IBM Faculty Award.

REFERENCES

[1] "VMware vCenter Server," http://www.vmware.com/products/vcenter-server/overview.html.

[2] J. Tai, B. Sheng, Y. Yao, and N. Mi, "Live data migration for reducing sla violations in multi-tiered storage systems," Tech Report, 2013.

[3] B. Laliberte, "Automate and Optimize a Tiered Storage Environment-FAST!" White Paper, 2009, http://www.emc.com/collateral/analyst-reports/esg-20091208-fast.pdf.

[4] B. Lundell, J. Gahm, and J. McKnight, "2011 IT Spending Intentions Survey," Research Report, 2011, http://www.enterprisestrategygroup.com/2011/01/2011-it-spending-intentions-survey/.

[5] "IBM DS8000," http://www-03.ibm.com/systems/storage/disk/ds8000/.

[6] "EMC FAST," http://www.emc.com/products/launch/fast/.

[7] "HP 3PAR Adaptive Optimization Software," http://h18006.www1.hp.com/storage/software/3par/aos/index.html.

[8] S. Khuller, Y. Kim, and Y. Wan, "Algorithms for data migration with cloning," in *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. San Diego, California: ACM, 2003, pp. 27–36.

[9] E. Anderson, J. Hall, J. Hartline, M. Hobbs, A. R. Karlin, J. Saia, R. Swaminathan, and J. Wilkes, "An experimental study of data migration algorithms," in *Workshop on Algorithm Engineering*. London, UK: Springer, 2001, pp. 145–158.

[10] V. Sundaram and P. Shenoy, "Efficient data migration in self-managing storage systems," in *IEEE International Conference on Autonomic Computing*, Dublin, Ireland, 2006, pp. 297–300.

[11] M. Karlsson, C. Karamanolis, and X. Zhu, "Triage: performance isolation and differentiation for storage systems," in *Twelfth IEEE International Workshop on Quality of Service*, Palo Alto, CA, 2004, pp. 67–74.

[12] B. Seo and R. Zimmermann, "Efficient disk replacement and data migration algorithms for large disk subsystems," *ACM Transactions on Storage*, vol. 1, no. 3, pp. 316–345, 2005.

[13] C. Lu, G. A. Alvarez, and J. Wilkes, "Aqueduct: Online data migration with performance guarantees," in *Proceedings of the 1st USENIX Conference on FAST'02*. Monterey, CA: ACM, 2002, pp. 219–230.

[14] G. Zhang, L. Chiu, and L. Liu, "Adaptive data migration in multi-tiered storage based cloud environment," in *IEEE 3rd International Conference on Cloud Computing*, Miami, FL, 2010, pp. 148–155.

[15] J. Guerra, H. Pucha, J. Glider, W. Belluomini, and R. Rangaswami, "Cost effective storage using extent based dynamic tiering," in *Proceedings of the 9st USENIX Conference on FAST'11*. San Jose, CA: ACM, 2011, pp. 20–20.