Cluster Comput
DOI 10.1007/s10586-015-0461-9

SLA-aware data migration in a shared hybrid storage cluster

Jianzhe Tai[1] · **Bo Sheng**[2] · **Yi Yao**[1] · **Ningfang Mi**[1]

Received: 19 May 2014 / Revised: 4 February 2015 / Accepted: 14 May 2015

Abstract Data volume in today's world has been tremendously increased. Large-scaled and diverse data sets are raising new big challenges of storage, process, and query. Particularly, real-time data analysis becomes more and more frequently. Multi-tiered, hybrid storage architectures, which provide a solid way to combine solid-state drives with hard disk drives (HDDs), therefore become attractive in enterprise data centers for achieving high performance and large capacity simultaneously. However, from service provider's perspective, how to efficiently manage all the data hosted in data center in order to provide high quality of service (QoS) is still a core and difficult problem. The modern enterprise data centers often provide the shared storage resources to a large variety of applications which might demand for different service level agreements (SLAs). Furthermore, any user query from a data-intensive application could easily trigger a scan of a gigantic data set and inject a burst of disk I/Os to the back-end storage system, which will eventually cause disastrous performance degradation. Therefore, in the paper, we present a new approach for automated data movement in multi-tiered, hybrid storage clusters, which lively migrates the data among different storage media devices, aiming to support multiple SLAs for applications with dynamic workloads at the minimal cost. Detailed trace-driven simulations show that this new approach significantly improves the overall performance, providing higher QoS for applications and reducing the occurrence of SLA violations. Sensitivity analysis under different system environments further validates the effectiveness and robustness of the approach.

Keywords Data migration · Resource allocation · Service level agreement (SLA) · Bursty workloads · Hybrid storage clusters

1 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 TB of data, including 83 million pictures. Being one of the largest databases in the world, Google processes more than 25 PB 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.

In 1990s, flash-based solid-state drives (SSDs) were first introduced to maintain the data in the memory chips. Nowadays, SSDs have gained prominence in enterprise arrays and successfully been used as a replacement of HDDs because of significant performance improvement (e.g., high IOPS and low latency) 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 stor-

✉ Ningfang Mi
ningfang@ece.neu.edu

Jianzhe Tai
jtai@ece.neu.edu

Bo Sheng
shengbo@cs.umb.edu

Yi Yao
yyao@ece.neu.edu

[1] Northeastern University, Boston, MA, USA

[2] University of Massachusetts Boston, Boston, MA, USA

Published online: 27 May 2015

age 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 a cluster of hybrid storage devices 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. SLA is used to specify the quality of services that a service provider can offer to a customer. Usually, SLAs are in measurable terms, which can have a wide range of metrics such as throughput, availability, response times, etc.. In this paper, we focus on a response time criteria that must be met. These data centers need to be SLA aware in the management of shared storage resources in order to achieve different performance goals for applications. 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; for example, a user query from a data-intensive application might easily trigger a scan of a gigantic data set and then bring a burst of disk I/Os into the system. Bursty workloads inevitably cause 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, hybrid storage clusters. LMsT attempts to lively migrate the data across different storage tiers, aiming to guarantee SLA requirements on response times of IO requests for applications under dynamic workloads. We show that LMsT can efficiently utilize high-performance devices (e.g., SSDs) to improve performance for applications which are experiencing bursty traffic and more importantly to provide performance guarantees for latency-sensitive applications with strict SLAs. In detail, there are two phases in our new scheme: the selection phase chooses a set of potential migration candidates; and the validation phase examines each migration candidate, quantifies the benefits, and estimates the risk of SLA violations. Based on two sets of migration constraints, related to *SLA* and *performance*, LMsT dynamically determine *which* data blocks should be migrated, *where* to migrate the selected data blocks, and *when* to start a migration process.

The evaluation of LMsT is done via detailed trace-driven simulations. Experimental results show that by moving bursty workloads (i.e., hot data) to high-performance tiers (e.g., SSDs), our LMsT algorithm achieves significant performance improvement, providing higher QoS for applications under bursty workloads and reducing the occurrence of SLA violations in the low performance HDD-tiers. More importantly, under LMsT, neither additional migration I/Os nor newly migrated bursty workloads cause any delays to latency-sensitive applications that are already in SSD-tiers. Sensitivity analysis with respect to storage capacities, burstiness profiles, and parameter settings in two migration constraints further validates the effectiveness and robustness of LMsT.

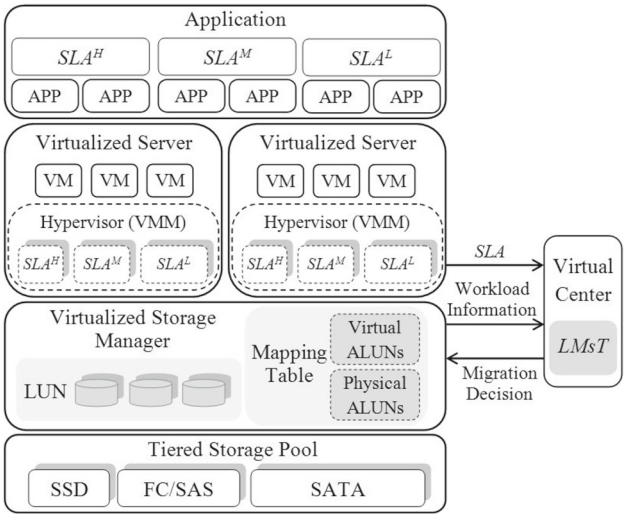

Fig. 1 The structure of a multi-tiered storage system

This paper is organized as follows. Section 2 demonstrates the architecture of a multi-tiered storage system. Section 3 presents the LMsT algorithm for automated data migration in multi-tiered, hybrid storage systems. Section 4 evaluates the effectiveness and robustness of LMsT using trace-driven simulations. Section 5 gives an overview of the related works. Finally, we draw conclusions in Sect. 6.

2 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.

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. The *back-end storage pool* is modeled as a multi-tiered, hybrid paradigm, which consists of different storage devices such as SSD, FC/SAS, and SATA.

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 1 GB 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 [17]) 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.

3 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 1 gives the notations that are used in this paper.

Table 1 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)

3.1 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.

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 min 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.

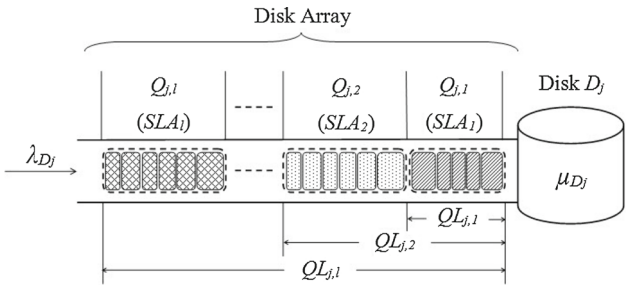

Fig. 2 The profile of logical buffers and disk array

ALGORITHM: The high level description of LMsT
for each time window t_{win}
a. Determine the migration candidates (i.e., ALUNs), see Sec. 3.2
I. *Selection Phase:* find the potential candidates (A_i, D_j) for forward and backward migration;
II. *Validation Phase:* select a subset of potential candidates using two sets of constraints, see Eq.s(8) and (16);
b. Determine the trigger time for a migration process, see Sec. 3.3
I. Estimate the migration duration t_{mgt} for each migration candidate using Eq.(17);
II. Schedule backward migrations;
III. Schedule forward migrations till the end of window or no more migration candidates;
end

Fig. 3 The high level description of LMsT

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

$$\lambda_{D_j} = \sum_i x_{i,j} \cdot \lambda_{A_i}, \tag{1}$$

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

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. A large α will increase the estimation of arrival rates (i.e., λ_{A_i} and λ_{D_j}) and thus reduce the number of ALUNs that can be validated for migration by the SLA constraint. That is, LMsT becomes more conservatively migrate ALUNs to SSDs, resulting lower migration ratios with a large α value, as shown in Sect. 4.2. On the other hand, with a very small value of α , LMsT will be too aggressive to move a large number of ALUNs to SSDs, which thus dramatically increases the load at SSDs and diminishes the benefits of using SSDs. A moderate α value should be considered to achieve the best performance improvement.

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,

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

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 Fig. 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 $x_{i,j} \cdot y_{i,k} = 1$. The high level idea of our new migration algorithm is also shown in Fig. 3. Upon every t_{win} time window, the migration algorithm LMsT bases on the monitored information during that time window to determine the migration candidates as well as the trigger time for a migration process. The duration of such a time window (i.e., t_{win}) should be properly set according to workload changes, i.e., how frequently the workloads shift from busy (idle) periods to idle (busy) periods. If the workloads change frequently, then a small time window is preferred in order to quickly adapt the changes by migrating peak IOs to SSDs. Vice versa.

3.2 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.

3.2.1 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.

3.2.2 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.

3.2.3 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 Fig. 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,

$$ML_{j,k} = SLA_k \cdot \mu_{D_j}. \tag{4}$$

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,

$$\overline{\lambda}_{j,k} = \sum_{t=1}^{k} \sum_{i=1}^{n} x_{i,j} \cdot y_{i,t} \cdot \lambda_{A_i}. \tag{5}$$

Thus, using Little's Law [10], $QL_{j,k}$ can be expressed as

$$QL_{j,k} = f(\overline{\lambda}_{j,k}) = \frac{\overline{\lambda}_{j,k}}{\mu_{D_j} - \overline{\lambda}_{j,k}} \cdot s_{D_j}. \tag{6}$$

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),

$$\lambda_{D_j} + \lambda_{A_i} < \mu_{D_j}, \tag{7}$$

$$ML_{j,k} \geq QL'_{j,k} = f\left(\overline{\lambda}_{j,k} + \lambda_{A_i}\right), \text{ for} y_{i,k} = 1. \tag{8}$$

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 $ML_{j,k}$.

3.2.4 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 λ'_{D_k}, λ'_{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,

$$RT_j = g(j, \lambda_{D_j}) = \frac{s_{D_j}}{\mu_{D_j} - \lambda_{D_j}}. \tag{9}$$

With Eq. (9), 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:

$$RT_k(A_i, D_j) = \left(g\left(k, \lambda_{D_k}\right) + g\left(k, \lambda_{D'_k}\right)\right) \cdot t_{win}, \tag{10}$$

$$\begin{aligned} RT'_k(A_i, D_j) = {} & g(k, \lambda_{D_k}) \cdot (t_{win} - t_{mgt}) \\ & + g(k, \lambda_{D_k} + \Delta\lambda) \cdot t_{mgt} \\ & + g\left(k, \lambda'_{D_k} - \lambda'_{A_i}\right) \cdot t_{win}, \end{aligned} \tag{11}$$

$$\overline{RT}_k(A_i, D_j) = \frac{RT'_k(A_i, D_j) - RT_k(A_i, D_j)}{RT_k(A_i, D_j)}, \tag{12}$$

$$RT_j(A_i, D_j) = \left(g\left(j, \lambda_{D_j}\right) + g\left(j, \lambda_{D'_j}\right)\right) \cdot t_{win}, \tag{13}$$

$$\begin{aligned} RT'_j(A_i, D_j) = {} & g(j, \lambda_{D_j}) \cdot (t_{win} - t_{mgt}) \\ & + g(j, \lambda_{D_j} + \Delta\lambda) \cdot t_{mgt} \\ & + g\left(j, \lambda'_{D_j} + \lambda'_{A_i}\right) \cdot t_{win}, \end{aligned} \tag{14}$$

$$\overline{RT}_j(A_i, D_j) = \frac{RT'_j(A_i, D_j) - RT_j(A_i, D_j)}{RT_j(A_i, D_j)}. \tag{15}$$

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. That is, the value of $e\,\%$ indicates the expected overall improvement (compared to NMsT) in terms of response times when an ALUN can be validated for migration. A large value of $e\,\%$ will decrease the number of ALUNs that can be validated for migration by the response time constraint and thus generate a conservative migration process with low migration ratios.

$$\frac{\overline{RT}_k(A_i, D_j) + \overline{RT}_j(A_i, D_j)}{2} > e\,\% \tag{16}$$

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.

3.3 Migration trigger time

Given validated migration candidates, we now turn to schedule them for actual migration. To fulfill this schedule, the first key issue is to find out when to trigger a migration process. Thus, we first estimate the migration duration for each candidate, and then present a migration trigger policy for both forward and backward migrations.

3.3.1 Estimation of migration duration

If a migration candidate (i.e., ALUN) meets the SLA constraint, then one can expect that both the source and the destination disks have extra capabilities to process migration I/Os which acquire additional transfer bandwidth.

Thus, we have the transfer rate $\Delta\lambda_{D_j} = \mu_{D_j} - \lambda_{D_j}$ for serving migration I/Os at the destination disk D_j, where μ_{D_j} and λ_{D_j} are D_j's service rate and arrival rate, respectively. Similarly, the transfer rate $\Delta\lambda_{D_k}$ for serving migration I/Os at the source disk D_k is equal to $\mu_{D_k} - \lambda_{D_k}$, where μ_{D_k} and λ_{D_k} are the service and the arrival rates of D_k, respectively. Conservatively, we choose the minimum one of $\{\Delta\lambda_{D_k}, \Delta\lambda_{D_j}\}$, as the mutual transfer rate for serving migration I/Os between the source and the destination disks.

Furthermore, the actual migration capacity sometimes can be larger than the size of an ALUN (i.e., 1 GB) because application write I/Os might arrive during migration process which can incur new migration I/Os for modified data. We thus adopt a tuning parameter β to assess the actual migration capacity and estimate the migration duration t_{mgt} for each candidate using the following equation:

$$t_{mgt} = \frac{\beta \times (\text{Capacity of ALUN})}{\min\{\mu_{D_j} - \lambda_{D_j}, \mu_{D_k} - \lambda_{D_k}\}}, \tag{17}$$

where μ_{D_j} and λ_{D_j} (resp. μ_{D_k} and λ_{D_k}) are the service and the arrival rates of D_j (resp. D_k), respectively. The parameter β can be tuned to adjust the total migration capacity for each ALUN. Even though an ALUN has 1 GB capacity, the real migration capacity might be larger than this size as some blocks might be rewritten during the migration process. Thus, the parameter β is proportional to the IO dirty ratio which can be calculated by the average dirty ratio.

3.3.2 Migration trigger policy

As backward migrations are used to release the space in high performance tiers, we need to execute all backward migrations in the beginning of each time window. Once all backward migrations are done, the policy starts to schedule the remaining forward migrations. However, it is possible that the entire migration process (i.e., backward plus forward migrations) exceeds a time window such that some migration candidates cannot be successfully executed. Our policy thus abandons all those migration candidates.

4 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.

Table 2 Device parameters of two tiers

Disk type	Disk number	Total capacity (GB)	Service rate (MB/s)
SSD	2	40	500
FC	5	100	160

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 $SLA^H = 1$ 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 2. The service rates of these two types of devices are roughly parameterized based on the specifications of actual devices. For example, Intel Solid-State Drive 530 Series with capacity 80 G has sequential read/write of 540/480 MB/s while Fujitsu Enterprise MAX3073FC with capacity of 73.5 GB has external transfer rate up to 200 MB/s. In general, we want to show that given multiple storage devices with different capacity and processing capability, our algorithm can leverage such kinds of difference to best utilize both of these devices through efficient data migration. 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. The e % parameter used as the threshold to determine the overall improvement in average response times is set to 6–10 %.

Consider 7 applications (2 with SLA^H and 5 with SLA^L) to access a 70 GB data set. In average, each application requires 10 virtual ALUNs and the capacity of such a virtual ALUN is 1 GB. We then generate an I/O stream for each virtual ALUN such that there are totally 30 time periods and each lasts around 20 min. 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 100 KB.

To evaluate our new migration algorithm, we generate a set of synthetic I/O traces for each application, each of which is used as an I/O stream for one of ten virtual ALUNs of that application. These synthetic traces are parameterized based on the MSR Cambridge block I/O traces [13]. Each data entry of these traces describes an I/O request, including timestamp, disk number, logical block number (LBN), number of blocks and the type of I/O (i.e., read or write). Therefore, the specifications of an I/O request in our synthetic traces also include I/O arrival time (i.e., timestamp), I/O address (i.e., disk number and logical block number),

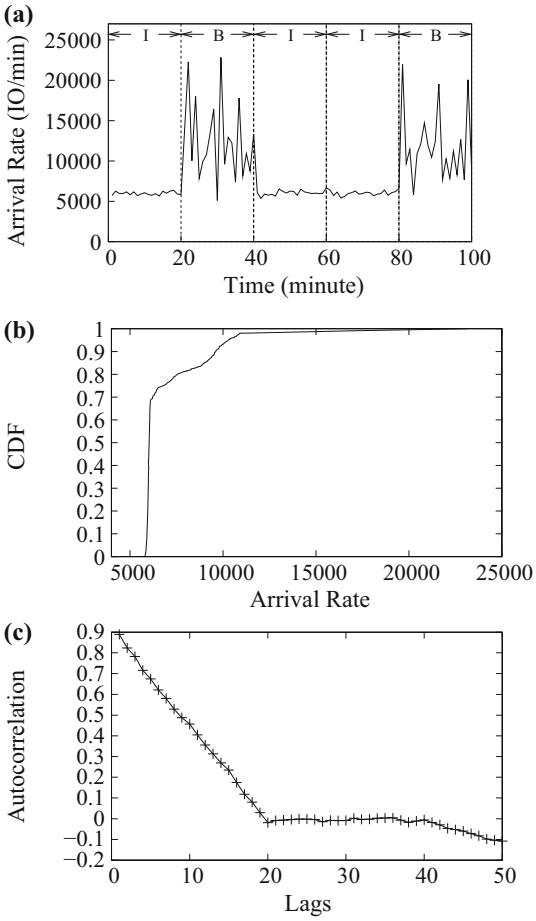

Fig. 4 Illustrate **a** the arrival rate per minute, **b** the CDF of arrival rate, and **c** the autocorrelation of arrival rate at different time lags of a high priority application in one virtual ALUN

I/O size (i.e., number of blocks) and the type of I/O (i.e., read or write). We found that arrival flows in enterprise data centers and storage systems often exhibit burstiness (temporal dependence) [14,18]; for example, a user query from a data-intensive application might easily trigger a scan of a gigantic data set and then bring a burst of disk I/Os into the system. Such burstiness profiles are also observed in arrival rates (e.g., number of accessed bins per second) of some MSR Cambridge traces, e.g., proj1 and web2. Therefore, we consider to imitate burstiness in the arrivals of our synthetic I/O traces by injecting 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. 4. I/O inter-arrival times in an idle period are generated 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. As shown in Fig. 4, while the arrival rates keep stable

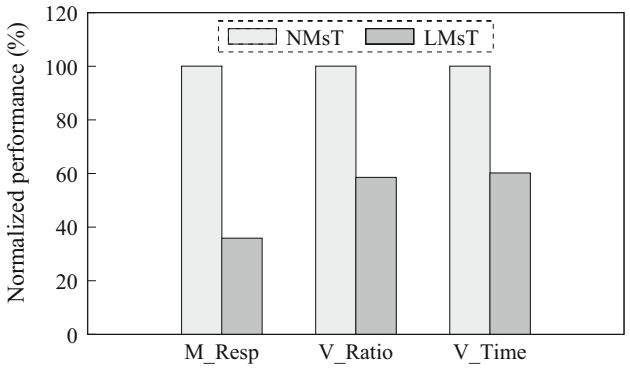

Fig. 5 Performance results under NMsT and LMsT

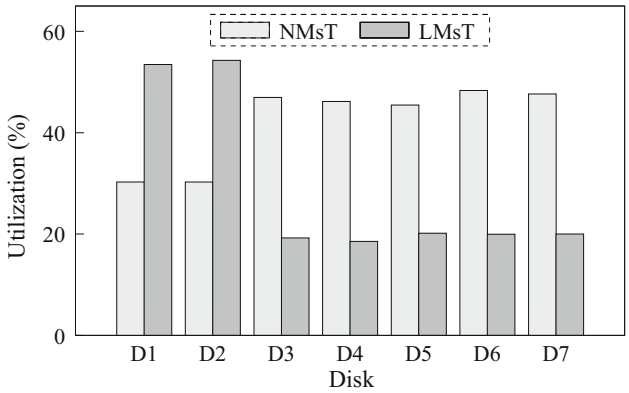

Fig. 6 Disk utilizations under NMsT and LMsT

(follow exponential distribution) in most of the time (70 % for this application), bursty periods introduce high autocorrelation to the arrival rate and long tail to the arrival rate distribution.

Figure 5 depicts the performance results under NMsT and LMsT, where NMsT (i.e., no migration process, all ALUNs from each application will stay in the disks where those ALUNs are initially placed) is used as the base case to normalize LMsT's performance. Specifically, under the NMsT approach, none of ALUNs will be migrated from one disk to another, that is, all ALUNs from each application will stay in the disks where those ALUNs are initially placed. For the initial configuration of data placement, we map two applications with high SLA priority (i.e., SLA^H) to the high performance tier (i.e., two SSDs) and the remaining five applications which have low SLA riority (i.e., SLA^L) to the low performance tier (i.e., five FCs). 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.

Given this initial data placement and no migration process, NMsT cannot best utilize the high performance tier (i.e., SSDs) such that the utilizations of two SSDs are about 20 % lower than those of FCs. Moreover, the SLA violation ratios are quite high (around 40 %) for low priority applications under NMsT . While, LMsT significantly reduces the average I/O response time by 60 %. This can be interpreted by the fact that LMsT makes a better use of SSDs by migrating all validated bursty I/O traffic from FCs to SSDs. Furthermore, we observe that LMsT improves the SLA violation as well by reducing I/O violation ratio (V_Ratio) and I/O violation time (V_Time) by 42 and 40 %, respectively. We thus conclude that LMsT is able to significantly improve the overall performance in terms of I/O response time, I/O violation ratio and I/O violation time.

To further investigate the usage of storage resources, we depict the avarage utilizations of 7 storage devices, i.e., two SSDs and five FCs under both NMsT and LMsT in Fig. 6. Here, disk utilization is defined as the ratio of disk busy time over the duration of the whole simulation. We first observe that under NMsT , two high performance SSDs (i.e.,D_1 and D_2) are not well utilized since their disk utilizations are only about 30 %; while the other devices (i.e., five FCs) at low performance tier are busier (around 50 %) to serve all I/O requests from low priority applications. By migrating bursty I/Os of low priority applications from FCs to SSDs, LMsT improves the usage of high performance tier with SSD utilizations more than 54 %, and thus reduces the response times for these migrated I/Os. On the other hand, the utilizations of FCs are significantly reduced as the loads on these devices now become less, which further improves the performance (e.g., respose times) of the remaining I/Os of low priority applications served on these devices.

To further investigate the performance impacts of migrations on each application, we present their average I/O response times and I/O violation ratios under both NMsT and LMsT in Table 3. We observe that all low priority applications (i.e., $App3, \ldots, App7$) 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.

4.1 Sensitivity analysis on system workloads

Now, we turn to analyze the effectiveness and robustness of LMsT under various experimental conditions. In this subsection, we first focus on exploring the sensitivity of LMsT to different system workloads. Later, we investigate the sensitivity analysis of LMsT to our migration constraints.

[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.

Table 3 Each application's performance under NMsT and LMsT

Capacity 70 GB 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

Table 4 Configuration of applications and disks under different storage active capacities, where Num_H (Num_L) is the number of high (low) priority applications and C_SSD (C_FC) gives the initial active capacity of SSD (FC), and N_TS is the total number of time slots at the FC-tiers

Active capacity (GB)	Num_H	Num_L	C_SSD (×2) (GB)	C_FC (×5) (GB)	N_TS
40	1	3	5	6	900
70	2	5	10	10	1500
100	3	7	15	14	2100

In Table 4, we list the configuration of applications and storage devices which are considered in our sensitivity analysis on various system and workload parameters, such as active storage capacities and burstiness profiles in arrival flows. First, we vary burstiness profiles in arrival flows by clustering different amounts of I/O requests in bursty periods such that the percentages of I/O requests that arrive during bursty periods are euqal to 30, 50 and 70 %. As the arrivals become more bursty, performance degradation becomes more significant because not all of bursty I/Os at FCs can be migrated to SSDs. Secondly, we vary total active storage capacities by setting the amount of storage space that has been used to store data for applications to 40, 70, and 100 GB. Since each application requires 10 ALUNs and the capacity of each ALUN is set as 10 GB in our experiments, we get different numbers of applications (i.e., 4, 7 and 10) under active storage capacities of 40, 70, and 100 GB, respectively, as well as different numbers of high and low priority applications (denoted as Num_H and Num_L in Table 4). For example, under active capacity of 40 GB, we have totally 4 applications such that 1 application has high priority and the other 3 ones have low priority. Furthermore, in our experiments, the numbers of storage devices are fixed as 2 SSDs and 5 FCs and the data of all high and low priority applications are initially placed in SSDs and FCs, respectively. Consequently, as shown in Table 4, active storage capacities (denoted as C_SSD and C_FC) that are initially placed in SSDs and FCs are thus accordingly changed as the total active capacity increases. Finally, we remark that the combination of increasing active capacity and burst ratio further exacerbates the bursty load to the system. For example, when the active capacity increases to 100 GB and the burst ratio is 70 %, the total number of time slots (see N_TS in Table 4) at FCs reaches to 2100, i.e., 70 ALUNs at FCs times 30 slots/ALUN. Consequently, the total number of bursty slots at FCs then becomes larger then 1400, see the row of $Burst\_TS$ in Table 6.

The experimental results of LMsT under 9 workload combinations are shown in Table 5. We also present the results of NMsT as well as the relative improvement with respect to NMsT in the table. First of all, we observe that under all the 9 workloads LMsT achieves non-negligible performance improvement in terms of the mean I/O response time (M_Resp), the fraction of I/Os that are SLA violated (V_Ratio), and the mean of SLA-violated times (V_Time). For example, under the case of 40 GB and 30 % burst, LMsT dramatically accelerates the average I/O response times by up to 83 % relative improvement over NMsT and decreases the number of SLA-violated I/Os with the relative improvement over NMsT up to 78 %. This indicates that LMsT provides the high QoS to low priority applications and meanwhile maintains the SLAs for high priority ones.

Also, we find that the burstiness in arrival flows does deteriorate the overall system performance under both LMsT and NMsT policies. Such performance degradation becomes more significant when the arrivals become more bursty, i.e., the bursty ratio increases. Similarly, the increasing active storage capacity degrades the system performance as well because the overall disk loads are increased. This further results in strict migration constraints, allowing fewer bursty ALUNs to be migrated. In addition, the combination of large active storage capacity and high bursty ratio makes the relative improvement over NMsT less visible, e.g., in the case of 100 GB and 70 % burst, the relative improvements with respect to all the three performance metrics diminish.

Table 5 Sensitive analysis of system workloads with active storage capacity of (a) 40 GB, (b) 70 GB, and (c) 100 GB. The burst ratio is set to 30, 50 and 70 %

	30 % Burst	50 % Burst	70 % Burst
(a) Capacity 40 GB			
NMsT			
M_Resp (ms)	29.45	66.93	127.43
V_Ratio (%)	7.16	13.72	21.80
V_Time (ms)	381.17	463.15	562.29
LMsT			
M_Resp (ms)	5.04 (82.89 %)	15.17 (77.33 %)	35.71 (71.98 %)
V_Ratio (%)	1.58 (77.94 %)	4.67 (65.98 %)	9.87 (54.70 %)
V_Time (ms)	271.16 (28.86 %)	295.45 (36.21 %)	337.15 (40.04 %)
(b) Capacity 70 GB			
NMsT			
M_Resp (ms)	67.11	205.20	433.90
V_Ratio (%)	12.00	25.78	39.68
V_Time (ms)	533.41	775.47	1075.62
LMsT			
M_Resp (ms)	24.65 (63.27 %)	73.51 (64.18 %)	140.44 (67.63 %)
V_Ratio (%)	6.21 (48.23 %)	15.09 (41.46 %)	28.35 (37.96 %)
V_Time (ms)	370.86 (30.47 %)	466.64 (39.82 %)	553.46 (48.55 %)
(c) Capacity 100 GB			
NMsT			
M_Resp (ms)	195.02	716.30	1675.73
V_Ratio (%)	30.00	47.18	61.74
V_Time (ms)	631.72	1502.26	2699.83
LMsT			
M_Resp (ms)	72.59 (62.78 %)	290.50 (59.44 %)	1324.17 (20.98 %)
V_Ratio (%)	18.85 (37.15 %)	36.49 (22.66 %)	56.67 (8.21 %)
V_Time (ms)	368.39 (41.68 %)	781.28 (47.99 %)	2322.53 (13.95 %)

4.2 Sensitivity analysis on parameters in migration constraints

Recall that the key idea of LMsT is to improve the QoS for low priority applications via on-the-fly moving their hot data to SSDs, and meanwhile without causing additional SLA violations to high priority applications. Therefore, it is critical to set the right parameters for the migration constraints in order to achieve the best performance of LMsT. To address this issue, we here conduct a set of experiments to investigate the sensitivity of LMsT to the key parameters in two sets of migration constraints, i.e., SLA constraint shown in Eq. (7) and response time constraint shown in Eq. (16).

We show the performance results measured from the experiments, where we vary the parameter α of SLA constraint in Fig. 7a and the parameter e % of response time constraint in Fig. 7b. All other parameters in these experiments are kept the same. Additionally, the total active storage capacity is set of 70 GB and 50 % of arrival flows to FCs are bursty. Then, we have 2 applications with high priority and 5 applications with low priority, see Table 4. In these experiments, we measure the migration ratio of bursty ALUNs in FC-tier and the response time ratio between LMsT and NMsT.

We first tune the parameter α in Eq. (2) to control the arrival rates to an ALUN and the associated disk. As the value of α increases, LMsT then conservatively migrates the ALUNs of low priority applications from FCs to SSDs due to the current heavy load (i.e., larger arrival rate) at SSDs. That is, the number of ALUNs which can be validated for migration by the SLA constraint in Eq. (7) becomes less, so that the migration ratio decreases, see Fig. 7a. However, the trend of the response time is not straightforward. As we discussed, LMsT with a large α conservatively migrates ALUNs and thus obtains less improvement in response times. On the other hand, when α is set to a small value, a large number of ALUNs may be aggressively moved to SSDs, which thus dramatically increases the load at SSDs and degrades the performance of the corresponding applications. We observe that the most benefits of LMsT are actually achieved when α is close to 0.3.

As shown in Eq. (16), the parameter e % is used as a threshold to determine the overall improvement in average response

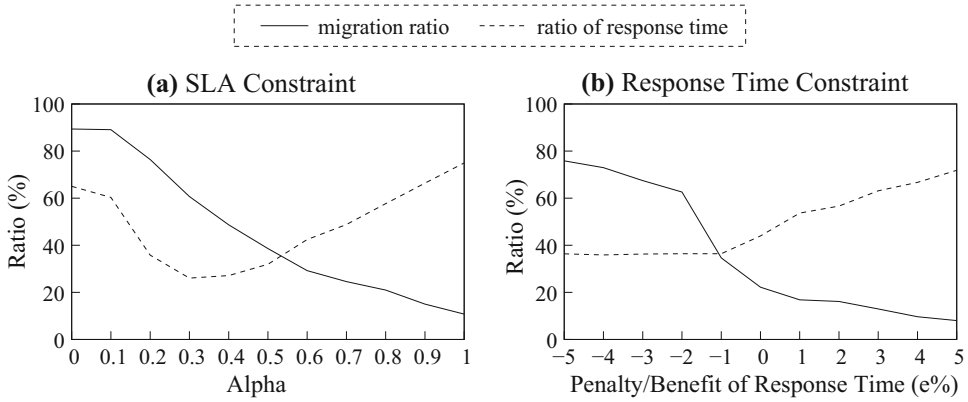

Fig. 7 Ratios of system performance between LMsT and NMsT using different parameters in **a** SLA constraint and **b** response time constraint

time, where the negative or positive value of $e\,\%$ (e.g., $-5\,\%$ or $5\,\%$) indicates the penalty or benefit in terms of response time. That is, when the relative benefit or penalty over NMsT is more or less than $e\,\%$, an ALUN will then be validated for migration. Consequently, a large value of $e\,\%$ indicates a conservative migration process, so that the migration ratio decreases and the ratio of response time between LMsT and NMsT increases, see Fig. 7b.

4.3 Performance comparisons with SMsT

In this section, we further validate the effectiveness of LMsT by comparing its performance with a migration algorithm, named SMsT, which adopts the existing approach (referred to as stop and copy) for migrating data from the source device to the destination one, which has been discussed and evaluated in [2]. The stop-and-copy technique for migrating is to stop serving all I/Os to access to-be-migrated data at the source, move data to the destination, and start serving those suspended I/Os at the destination. This approach gives higher priority to migration I/Os than I/Os requesting to access to-be-migrated data, which is simple to achieve safe migration yet results in considerably long down times for applications. Thus, we considered to use another set of performance metrics (i.e., Mgr_Resp, $Burst\_TS$ and Mgr_Ratio) to focus on evaluate the effectiveness of our LMsT algorithm during migration periods. For example, Mgr_Resp represents the mean response time of I/Os which would access to-be-migrated ALUNs during migration processes. This performance metric is used to evaluate the impact of migrations on regular application I/Os, i.e., how migration processes will add extra delay times on those I/Os. We use $Burst\_TS$ and Mgr_Ratio to further investigate the amount of migrated ALUNs under different workloads (in terms of burstiness profiles and active workload sizes).

Table 6 shows the resultant comparison between LMsT and SMsT. We remark that under SMsT, all the other application I/Os which do not access the to-be-migrated ALUNs still have higher priority than the migration I/Os during the migration process. In Table 6, the first important observation is that our new migration policy achieves much better performance (i.e., Mgr_Resp) under different workloads compared to SMsT. We interpret that using the synchronization mechanism, LMsT can effectively eliminate the negative impacts of migration on regular application I/Os, avoiding unnecessary delays to their execution.

Besides the results of mean response times, we also show the total number of time slots ($Burst\_TS$) that have bursty arrivals in all the ALUNs of 5 FCs, as well as the fraction of bursty time slots (Mgr_Ratio) which are validated to be migrated, see Table 6. We observe that as active storage capacity and bursty load increase, the total number of bursty time slots increases, whereas the migration ratio decreases. We interpret that as the heavy bursty loads reduce the capability of both SSDs and FCs to migrate data, few migration candidates can be validated by our migration constraints. This further verifies the experimental results in Table 5. Both large $Burst\_TS$ and low Mgr_Ratio incur non-negligible degradation in the performance improvement. For example, in the case of 100 GB and 70 % burst, LMsT achieves the smallest relative improvements over NMsT, compared to all the other cases.

As a final remark, it is interesting to observe that in the case of 40 GB active capacity, the migration ratios are similar across different bursty loads (i.e., 30, 50 and 70 % burst). This is because that the overall system load is very low under this case, therefore, most of the bursty ALUNs can be migrated to SSDs no matter how heavy the bursty load is.

5 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 [9]. *Storage tiering* techniques are introduced to dynamically deliver appropriate resources to the business, targeting at performance

Table 6 Sensitive analysis of migration policies under system workloads with active storage capacity of (a) 40 GB, (b) 70 GB, and (c) 100 GB

	30 % Burst		50 % Burst		70 % Burst	
	LMsT	SMsT	LMsT	SMsT	LMsT	SMsT
(a) Capacity 40 GB						
Mgr_Resp (ms)	13.31	140.22	66.89	165.81	194.79	296.38
$Burst\_TS$	267		428		591	
Mgr_Ratio (%)	87.64		89.72		89.85	
(b) Capacity 70 GB						
Mgr_Resp (ms)	73.39	162.22	235.75	329.73	336.99	495.15
$Burst\_TS$	418		725		1038	
Mgr_Ratio (%)	89.71		76.41		49.62	
(c) Capacity 100 GB						
Mgr_Resp (ms)	79.60	188.55	266.88	371.58	323.67	418.77
$Burst\_TS$	644		1073		1464	
Mgr_Ratio (%)	53.73		25.82		12.98	

The burst ratio is set to 30, 50 and 70 %. Here, Mgr_Resp is the mean response time of the application I/Os which access the to-be-migrated ALUNs and $Burst\_TS$ is the number of bursty time slots in all the ALUNs of 5 FCs, and Mgr_Ratio is the migration ratio over $Burst\_TS$

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 [12], 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 [6], EMC Fully Automated Storage Tiering (FAST) for Celerra [3], and HP Adaptive Optimization for 3PAR [5].

A large literatures on storage management have been developed for the years. Recently, [1,4,7,8,11,15,16,19] proposed several new techniques (algorithmic or theoretical) to explore the effective data migration in storage systems. For example, [1,8] 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 [7] 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, [16] focused on minimizing the overhead of data migration by automatically detecting hotspots and reconfiguring the system based on the bandwidth-to-space ratio. [4] 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.

A cost model [15] has been developed to solve the problem of efficient disk replacement and data migration in a polynomial time. [11] implements the QoS guarantee of performance impact on foreground work by leveraging a control-theoretical approach to dynamically adapt migration speed. [19] 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 [19] assumes that the I/O profile exhibits a cyclic behavior and does not consider different application SLAs in their algorithm.

6 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.

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

References

1. Anderson, E., Hall, J., Hartline, J., Hobbs, M., Karlin, A.R., Saia, J., Swaminathan, R., Wilkes, J.: An experimental study of data migration algorithms. In: Proceedings of the Workshop on Algorithm Engineering, pp. 145–158. Springer, London (2001)
2. Elmore, A.J., Das, S., Agrawal, D., El Abbadi, A.: Zephyr: live migration in shared nothing databases for elastic cloud platforms. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pp. 301–312. ACM, New York (2011)
3. Emc, FAST. http://www.emc.com/products/launch/fast/
4. Guerra, J., Pucha, H., Glider, J., Belluomini, W., Rangaswami, R.: Cost effective storage using extent based dynamic tiering. In: Proceedings of the 9st USENIX Conference on FAST'11, pp. 20–20. ACM, San Jose, CA (2011)
5. HP 3PAR Adaptive Optimization Software. http://h18006.www1.hp.com/storage/software/3par/aos/index.html
6. IBM DS8000. http://www-03.ibm.com/systems/storage/disk/ds8000/
7. Karlsson, M., Karamanolis, C., Zhu, X.: Triage: performance isolation and differentiation for storage systems. In: Proceedings of the Twelfth IEEE International Workshop on Quality of Service, Palo Alto, CA, pp. 67–74. IEEE (2004)
8. Khuller, S., Kim, Y., Wan, Y.: Algorithms for data migration with cloning. In: Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 27–36. ACM, San Diego, CA (2003)
9. Laliberte, B.: Automate and Optimize a Tiered Storage Environment-FAST! White Paper (2009). http://www.emc.com/collateral/analyst-reports/esg-20091208-fast.pdf
10. Little, J.D.C.: A proof for the queuing formula: L = w. Oper. Res. **9**(3), 383–387 (1961)
11. Lu, C., Alvarez, G.A., Wilkes, J.: Aqueduct: online data migration with performance guarantees. In: Proceedings of the 1st USENIX Conference on FAST'02, pp. 219–230. ACM, Monterey, CA (2002)
12. Lundell, B., Gahm, J., McKnight, J.: 2011 IT Spending Intentions Survey. Research Report (2011). http://www.enterprisestrategygroup.com/2011/01/2011-it-spending-intentions-survey/
13. Narayanan, D., Donnelly, A., Rowstron, A.: Write off-loading: practical power management for enterprise storage. ACM Trans. Storage **4**(3), 10:1–10:23 (2008)
14. Riska, A., Riedel, E.: Long-range dependence at the disk drive level. In: Proceedings of theThird International Conference on Quantitative Evaluation of Systems, QEST 2006, pp. 41–50. IEEE (2006)
15. Seo, B., Zimmermann, R.: Efficient disk replacement and data migration algorithms for large disk subsystems. ACM Trans. Storage **1**(3), 316–345 (2005)
16. Sundaram, V., Shenoy, P.: Efficient data migration in self-managing storage systems. In: Proceedings of the IEEE International Conference on Autonomic Computing, Dublin, pp. 297–300 (2006)
17. VMware vCenter Server. http://www.vmware.com/products/vcenter-server/overview.html
18. Wang, K., Lin, M., Ciucu, F.: Characterizing the impact of the workload on the value of dynamic resizing in data centers. Perform. Eval. Rev. **40**(1), 405–406 (2014)
19. Zhang, G., Chiu, L., Liu, L.: Adaptive data migration in multi-tiered storage based cloud environment. In: Proceedings of the IEEE 3rd International Conference on Cloud Computing, Miami, FL, pp. 148–155. IEEE (2010)