Storage Sharing Optimization under Constraints of SLO Compliance and Performance Variability

Ning Li, Hong Jiang, *Fellow, IEEE*, Dan Feng *, and Zhan Shi
*Corresponding author

Abstract—SLO enforcement with the required strong SLO compliance and the desired low level of performance variability is necessary to ensure QoS for user applications with precisely differentiated service levels. However, for the cloud consolidating a large number of VMs rented by users, it is a great challenge to formulate an IO capacity allocation among consolidated VMs under the user-customized QoS constraints of SLO compliance and performance fluctuation for consolidated VMs. To address this challenge, we propose SASLO, an end-to-end VM-oriented control framework that supports users in customizing SLO targets and QoS constraints for each VM. SASLO can dynamically coordinate the throughput target and IO size limit for each VM adapting to the status of SLO enforcement so as to maximize the IO capacity allocation among consolidated VMs under QoS constraints. To accurately enforce time-varying throughput target, SASLO establishes a proportional-integral IO controller for each individual VM to converge the actual throughput to the target with an expected settling time. Our extensive evaluation driven by representative benchmarks demonstrates that SASLO is able to formulate a satisfactory IO capacity allocation plan for consolidated VMs under the constraints of SLO compliance and performance variability.

Index Terms—Quality of Services, Service-Oriented Enterprise Management, Solution-Level QoS Framework.

1 Introduction

A cloud system can be regarded as an Internet-scale and service-based software system since the number of VMs accommodated in the cloud can be scaled up and each VM will be specified by one or more service level objects (SLOs) according to the demands of users. However, it is with regret that the existing cloud systems are not able to provide the programmable interface supporting the storage IO sharing among consolidated VMs under the constraints of SLO compliance and performance fluctuation for each VM. Thus, SLO enforcement may be compromised, which in turn adversely affects user experiences.

For storage, the IO capacity allocation for consolidated VMs can be expressed in terms of throughput allocation and IO size limit for each VM, which are two key factors influencing the SLO compliance and performance fluctuation. The critical technique for optimizing the IO capacity allocation is to dynamically coordinate throughput allocation and IO size limit for each VM based on tracking the status of SLO enforcement. This control is built on a very simple assumption: we assume reducing IO throughput and IO size alleviates the IO contention for the storage device. This makes sense since the storage device load can be reduced when the IO requests arrive at a slow speed with decreased IO size that reduces request service time. Thus, performance variability can be alleviated and SLO compliance can be more easily met by the storage device with lower load. This scenario here involves a trade-off between the IO capacity allocation and the QoS constraints in terms of required SLO compliance and the desired level of performance variability. We can optimize this trade-off so as to let users receive IO capacity as high as possible under these QoS constraints.

However, to meet the QoS constraints, the storage IO capacity allocation among consolidated VMs is commonly unfixed since system throughput fluctuates [1]. The optimization procedure of the IO capacity allocation plan can be expressed as two time-varying series of throughput SLO target and IO size limit for each individual VM. Thus, there are two challenges we must face: 1) the dynamic enforcement for time-varying throughput SLO needs a type of SLO enforcement with the properties of stability and accuracy (i.e., *stable and accurate SLO enforcement*). *Stability* refers to the goals of the actual performance being able to converge to a given value (i.e., *steady-state value*) with an expected settling time while *accuracy* means that the steady-state value is adequately close to the target specified in SLO. If these properties can be satisfied in the SLO enforcement of each individual VM, cloud system can accurately and stably enforce multiple time-varying SLO series formulated by the IO capacity allocation plan for consolidated VMs. 2) it is difficult but necessary to bound the highly variable IO size for each individual VM that can significantly impact SLO enforcement.

For the first challenge, variable *IO characteristics* (i.e., request size, IO rate, etc.), the IO contention among VMs and unstable IO capacity provided by the storage back-end make it very difficult to support stable and accurate SLO enforcement of VM throughput performance. Existing IO control techniques adopted in the state of the arts [1–4]

- *This is the extension version of our manuscript that was accepted by International Conference on Cloud Computing and Big Data (CCBD), November 2015.*
- *N. Li, D. Feng and Z. Shi are associated with the School of Computer, Huazhong University of Science and Technology, Wuhan, China. H. Jiang is associated with the Department of Computer Science and Engineering, University of Texas at Arlington, USA.*
  E-mail: leebellwind@hust.edu.cn, hong.jiang@uta.edu, {dfeng, zshi}@hust.edu.cn

Manuscript received October 1, 2015; revised May 1, 2016.

basically lack a valid mechanism to ensure the properties of stability and accuracy under VM consolidation environment. In addition, it is nontrivial to discriminatively bound IO size fluctuation for each individual VM.

To address the above challenges, we propose a framework guaranteeing ***S**table and **A**ccurate **SLO** enforcement*, *SASLO*, to establish a *proportional-integral (PI)* controller, adopted from the control theory [5], to adjust the IO rate limit adapting to the performance error in an end-to-end VM-oriented fashion. PI control is a type of feedback control that combines the advantages of proportional control and integral control to ensure a zero deviation with an expectable convergence duration. SASLO can configure the PI controller automatically for each individual VM to ensure fast convergence for IO rate control. SASLO also constructs an *IO split controller* for each VM, which can be triggered to split a larger request into smaller ones by tracking the surge of IO latency. In this way, the probable penalty incurred by larger IO requests to SLO enforcement can be reduced. Based on the VM-oriented end-to-end PI controller and IO split controller, SASLO is able to automatically optimize the throughput allocation and IO size limit, which are the parameters configured for the PI controller and IO split controller of each consolidated VM respectively, under the constraints of SLO compliance and performance fluctuation. Thus, a satisfactory trade-off between high utilization of underlying storage devices and storage QoS can be made. This is desirable by both cloud service providers and users.

In contrast to the prior works in the context of SLO enforcement of VMs accessing shared storage resources, we believe that *SASLO is the first approach that is able to optimize storage IO capacity allocation under the user-customized QoS constraints in terms of required ranges of SLO compliance and performance variability*. We develop SASLO prototype in a small-scale cloud consisting of a cloud-computing server and a group of heterogeneous storage servers based on a 16-disk RAID0 and a SSD respectively. Our prototype evaluation shows that SASLO is able to make a satisfactory IO capacity allocation plan for consolidated VMs under user-customized QoS constraints.

The rest of the paper is organized as follows. Section 2 motivates the SASLO research and then review the related works. The architecture and design of SASLO are described in Section 3. Section 4 presents the implementation challenges. Detailed performance evaluation SASLO on a real system is presented in Section 5. We conclude the paper in Section 6.

2 Related Work

This paper focuses on SLO-Based storage sharing for the VM consolidation environment that is widely adopted in enterprise data centers or the cloud.

Storage Virtualization for VMs: Techniques for device emulation [6–8] can virtualize physical storage devices into multiple virtual disks of which each is dedicated to a specific VM. In addition, direct device access [9] can enable VM to achieve near-native performance by reducing the virtualization cost. MultiLanes [10] further improves the level of IO isolation among co-scheduled VMs by reducing the cost of contention on the data structure and locks in the kernel. In contrast, SASLO establishes IO controllers for each individual VM based on the device emulation of the Xen hypervisor [7, 11]. Our work is designed to optimize the IO capacity allocation for consolidated VMs under the constraints of SLO compliance and performance fluctuation, which is complementary and orthogonal to the previous works of storage virtualization for VMs.

IO Rate Control: IO rate control is critical for storage SLO enforcement. Façade [12], mClock [1] and SRP [2] use time-stamp based IO control; Pisces [3] adopts deficient round robin (DRR) [13] to perform IO scheduling; PARDA [4] uses the start-time fair queuing (SFQ) scheduler [14] to schedule VMs on the same host. SARC [15] is based on the leaky bucket technique [16], but allows the dispatched IO rate to be higher than SLO for high utilization.

Time-stamp based IO control refers to scheduling requests in the order of their time stamps, which has been adopted by many proposals [17–20]. mClock [1] conditionally alternates multiple time-stamp based IO controllers to enforce max-min fair share among concurrently running VMs. Leaky bucket [16] carries out IO throttling according to the SLO requirement. These two types of IO control algorithms are not able to eliminate the performance error caused by the IO rate fluctuation. While DRR [13] can reduce the performance deviation from the SLO target, it does not guarantee performance convergence to the target. In contrast, SASLO adopts proportional-integral (PI) control [21] in its SLO enforcement engine (SEE) that is established for each individual VM and carefully configures the PI controller in an end-to-end fashion to achieve the convergence and near-zero performance error in a feedback control loop.

SLO-Based Storage Sharing: The existing approaches to SLO-Based storage sharing can be broadly divided into three categories. The first is the class of approaches that support proportional sharing, such as PARDA [4], Argon [22], Aqua [23], Fahrrad [24] and SFQ(D) [14]. However, proportional-sharing based solutions cannot provide performance guarantees directly for a specific metric (e.g., the throughput in IOPS and the bandwidth in MB/s). The second is the class of algorithms supporting max-min fairness, such as mClock [1], SRP [2] and Pisces [3]. The third is the class of solutions that provide guarantees for latency-sensitive applications, such as Façade [12] and Avatar [15], which seek a trade-off between latency and throughput by IO queue size adjustment. SRP [2], based on mClock [1], can group related VMs into hierarchical pools and support IO throughput allocation at both the VM level and VM group level. Pisces [3] enables the sharing of key-value storage with max-min fairness on a per-tenant basis. IOFlow [25] supports high-level end-to-end policies at the VM or VM group level.

However, all these approaches lack a valid mechanism to guarantee the desired levels of SLO compliance and performance fluctuation for SLO enforcement under the consolidated VM environment. In contrast, SASLO is able to optimize storage IO sharing among consolidated VMs under the constraints of SLO compliance and performance fluctuation, which can potentially provide a SLO enforcement with strong compliance and stability.

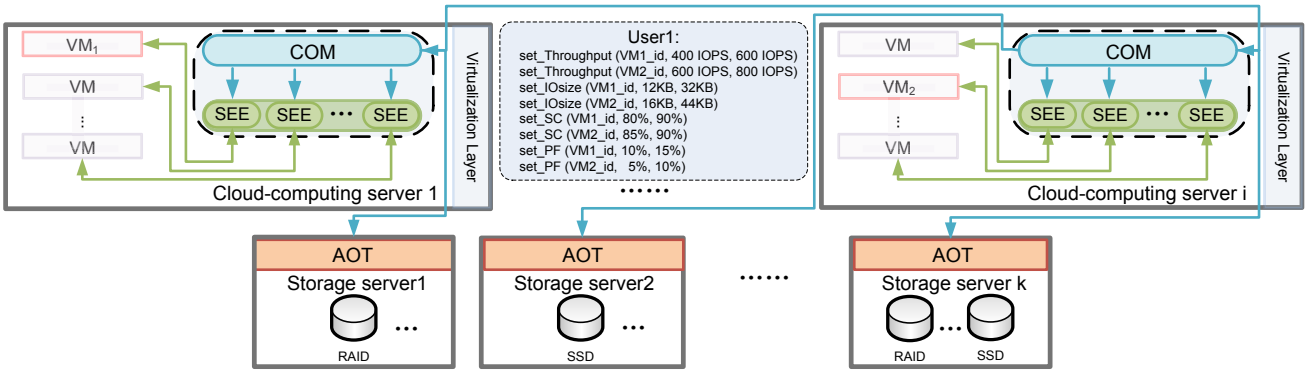

Fig. 1: The architecture of SASLO.

3 Architecture and Design

SASLO is designed for the cloud consisting of cloud-computing servers supported by storage servers. The cloud-computing servers provide a shared utility-computing infrastructure for tenants by encapsulating the user applications into VMs. Storage servers each involve one or more storage devices (e.g., disk array or SSD). Each storage device can be partitioned into several logical units (LUNs) of which each accommodates one or more the virtual disks (i.e., large image files) that are dedicated to VMs. Thus, all the VMs in the cloud-computing servers can share the storage devices connected over a storage area network (SAN). For brief, in this paper, the VMs supported by the same storage server are called as *consolidated VMs*.

SASLO views the underlying multi-layer IO stack (e.g., in network infrastructure and storage server) for a VM as a black box. The SLO enforcement engine (SEE) of SASLO implements VM-oriented feedback IO control adapting to the actual measured storage performance impacted by both network transfer and storage services. In so doing, SASLO can associate multi-resource (i.e., storage IO capacity and network bandwidth) with the real-time SLO enforcement for consolidated VMs. To facilitate our presentation of the design and architecture of SASLO, we first define and quantify the terms stability and accuracy in the context of SLO enforcement next.

3.1 Stability and Accuracy of SLO Enforcement

To quantify the accuracy of SLO enforcement, we firstly define performance error (E) as:

$$E = 100 * (P_{actual} - P_{SLO})/P_{SLO}\% \quad (1)$$

where P_{actual} is the actual performance and P_{SLO} refers to the target specified in SLO. To support accurate SLO enforcement, we define *SLO compliance* as the performance error (E) falling in the allowable range of $[-\delta_1\%, \delta_2\%]$ and the probability of SLO compliance is called the *SLO compliance rate (SC rate)*, which can be expressed by the following formula:

$$SC\ rate = Pr[-\delta_1\% \leq E \leq \delta_2\%] \quad (2)$$

where the values of δ_1 and δ_2 determine the extent of the allowable ranges for SLO compliance. Considering the amount of IO capacity can fluctuate widely for storage device (e.g., disk array) [1], we adopt the value of 10 for δ_1 and δ_2, which can be customized on demand.

In addition, we define *performance fluctuation ratio (PF ratio)* as the ratio of the standard deviation of the actual performance values measured during a period of time T to the target value in SLO, defined as:

$$PF\ ratio = 100 * \frac{SD(P_{actual}(k))}{P_{SLO}}\% \quad (3)$$

where $P_{actual}(k)$ is the k^{th} actual performance value measured in the period of T, P_{SLO} is the target value in SLO. Obviously, the smaller the value of the performance fluctuation ratio is, the less the performance fluctuation will be. We use the PF ratio to quantify the stability of SLO enforcement.

3.2 The Architecture of SASLO

As shown in Figure 1, SASLO consists of a *communication layer* (COM), an *SLO enforcement engine* (SEE) and an *automated optimization tool* (AOT), where COM and SEE are deployed on each cloud-computing server while AOT runs on each storage server. COM adds a layer of abstraction for the SLO enforcement protocol between the users and the service provider on each hypervisor. This abstraction simplifies the interface between the two in that each user simply sets their required SLO for their VM group and COM then places the SLO setting on the machines hosting the VM group. In addition, COM is responsible for transmitting the status of SLO enforcement in terms of SC rate and PF ratio obtained from the VMs required by AOT. Based on collecting the status of SLO enforcement from COM, AOT optimizes the IO capacity allocation among the VMs supported by the same storage server to guarantee that the current SLO targets of these VMs be feasible according to the required level of SLO compliance and performance fluctuation. Optimized SLO targets will be fed back by AOT to SEE that is in charge of SLO enforcement for each VM. Specifically, SASLO will construct one or more SEE for each individual VM to guarantee the convergence of SLO enforcement and near-zero performance error as well as bound the highly variable IO size for each VM.

SASLO provides a simple API for tenants to customize their SLO targets in terms of the allowable ranges of throughput and IO size limit as well as the QoS constraints in terms of the acceptable ranges of SC rate and PF ratio for each VM. As shown in Table 1, the first function is used

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2584057, IEEE Transactions on Services Computing

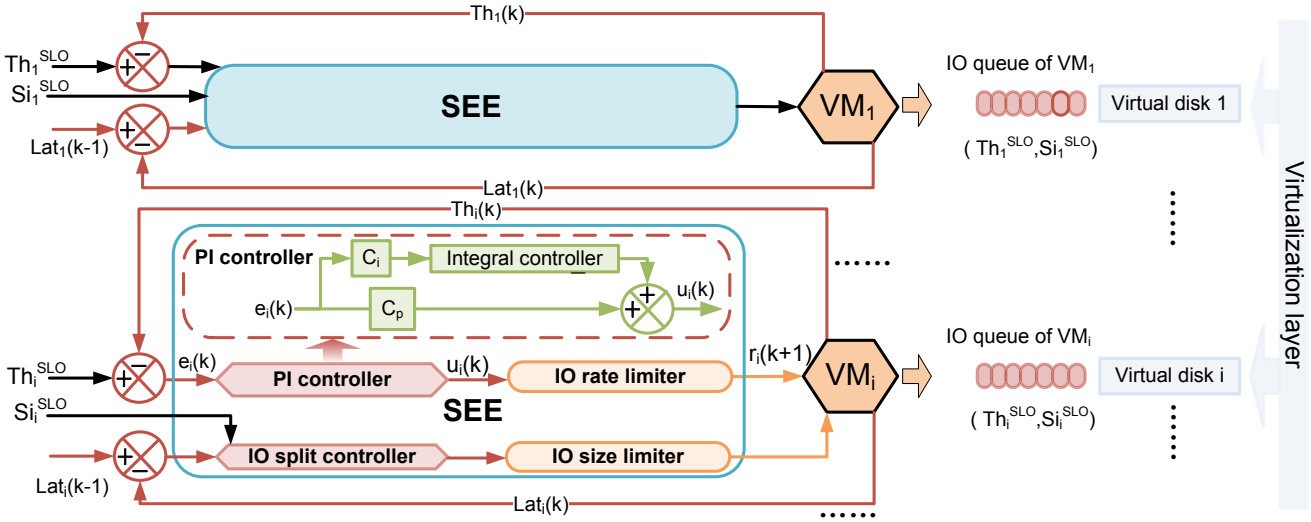

Fig. 2: An illustration of the SLO enforcement engine (SEE) in SASLO.

to set the allowable range of throughput SLO target for the VM tagged with d where the lower bound is the reservation and the upper bound is the limit. The second function is used to set the acceptable range of IO size limit for the VM tagged with d. *IO size limit* is the upper bound of the IO size exceeding which a request will be considered as oversize and liable to block the execution of other requests and thus be split into several smaller ones than the IO size limit. The third and fourth functions are used to set the desired range of SC rate and PF ratio for the VM tagged with d. Status of SLO enforcement can be obtained by invoking the fifth function listed in Table 1 that can transmit the newly updated SC rate and PF ratio measured for the VM tagged with d to the caller.

#	API
1	**set_Throughput (VM_id d, LB r, UB u)** set the lower and upper bounds of throughput at *r* and *u* respectively for the VM identified by *d*
2	**set_IOsize (VM_id d, LB r, UB u)** set the lower and upper bounds of IO size limit at *r* and *u* respectively for the VM identified by *d*
3	**set_SC (VM_id d, LB r, UB u)** set the lower and upper bounds of SC rate at *r* and *u* respectively for the VM identified by *d*
4	**set_PF (VM_id d, LB r, UB u)** set the lower and upper bounds of PF ratio at *r* and *u* respectively for the VM identified by *d*
5	**get_Status (VM_id d)** get the running status in terms of SC rate and PF ratio for the VM identified by *d*

TABLE 1: SASLO's API for users to customize SLO targets and QoS constraints

As illustrated in Figure 1, a user denoted by $User1$ rents two VMs (denoted by VM_1, VM_2), where VM_1 is deployed on the host $Cloud-computing\ server_1$ and VM_2 is running on the host $Cloud-computing\ server_i$. The user prescribes the SLO targets and QoS constraints for VM_1 and VM_2 by the API provided by SASLO. Specifically, the lower and upper bound of throughput SLO for VM_1 are 400 IOPS and 600 IOPS respectively while 600 IOPS and 800 IOPS are configured for VM_2. The lower and upper bound of IO size limit for VM_1 are 12 KB and 32 KB respectively while 16 KB and 44 KB are set for VM_2. The lower and upper bound of SC rate for VM_1 are 80 % and 90 % respectively while 85 % and 90 % are set for VM_2. The lower and upper bound of PF ratio for VM_1 are 10 % and 15 % respectively while 5 % and 10 % are set for VM_2. Once the user sets their required SLO targets and QoS constraints, COM will place these settings on the machines hosting the VM group, including the corresponding cloud-computing servers (e.g., $Cloud-computing\ server_1$ and $Cloud-computing\ server_i$) and storage servers (e.g., $Storage\ server_1$ and $Storage\ server_2$). And AOT deployed on these two storage servers will optimize the IO capacity allocation among consolidated VMs including VM_1 and VM_2 based on the SEE dedicated to each individual VM.

3.3 SLO Enforcement Engine

The SLO enforcement engine is in charge of SLO enforcement by an end-to-end IO rate and IO split control mechanism for each VM. As shown in Figures 2, SEE establishes two resource controllers and two resource limiters for each running VM. The two resource controllers include a proportional-integral (PI) controller and an IO split controller each of which drives the corresponding resource limiter (i.e., *IO rate limiter* or *IO size limiter*).

3.3.1 Resource Limiter

The storage resource allocation of IO rate and request size is facilitated by the corresponding resource limiters, i.e., IO rate limiter and IO size limiter.

SASLO adopts the strategy of the time-stamp assignment proposed in mClock [1], but only uses one time-stamp for each request. The time-stamp T_i^q assigned to a request q from VM_i, the larger of the sum of the previous time-stamp T_i^q-1 and $1/\hat{Rate}_i$ and the arrival time, can be expressed as:

$$T_i^q = max(T_i^{q-1} + 1/\hat{Rate}_i, \quad Arrival\ time) \tag{4}$$

The IO rate limiter for VM_i controls the interval between two consecutively scheduled requests by adjusting $\hat{Rate}_i$, where $\hat{Rate}_i$ is the target value of IO throughput. A higher value of $\hat{Rate}_i$ means that VM_i will have a higher IO throughput. Ideally, by keeping the arrival IO rate higher

than or equal to $\hat{Rate}_i$ and adequate capacity allocated for VM_i, VM_i achieves the IO throughput of $\hat{Rate}_i$.

The IO size limiter enforces the upper bound on the IO size seen by the device through setting *request-merge limit*, which is the upper bound on the IO size after requests are merged. If the sum of IO sizes of two merge-able requests is larger than the request-merge limit, the request merge will not be performed.

3.3.2 PI Control Model

We use proportional-integral (PI) control, adopted from the control theory [21], to make the actual IO throughput converge to the target specified in the SLO. PI control depends on a feedback control that needs to know the impact of the system input on the system output. As shown in Figure 2, the system output $r_i(k)$ is the function of the system input $u_i(k)$, i.e., $r_i(k) = f(u_i(k))$, and the arrival IO rate of VM_i is limited by $r_i(k)$. We adopt the autoregressive moving-average (ARMA) model [21] to capture the effect of the system input $u_i(k)$ on the system output $r_i(k)$, that is :

$$r_i(k+1) = a_i^k \times r_i(k) + b_i^k \times u_i(k) \tag{5}$$

in which a_i^k and b_i^k are parameters of the ARMA model [21] and $r_i(k)$ and $u_i(k)$ are the IO rate limit of VM_i at the k^{th} interval and the variation in $r_i(k)$ respectively. The values of the parameters a_i^k and b_i^k can be dynamically determined by the *least-squares* algorithm [21]. Ideally, the actual throughput at the k^{th} interval $Th_i(k)$ is equal to $r_i(k)$ and $Th_i(k+1)$ can be increased by the value of $u_i(k)$. Thus, a_i^k and b_i^k can be set at 1. In practice, the IO interference among co-scheduled VMs can make the actual throughput of a VM deviate from the control target set by the ARMA model, especially when the IO throughput target of the VM is infeasible under the available storage capacity provided by the storage utility.

The IO rate controller adopts PI control to dynamically coordinate the system input $u_i(k)$ so that the actual IO throughput can converge to the desired value in SLO. This is essential to guarantee the accuracy of SLO enforcement. As shown in Figure 2, the system input $u_i(k)$ consists of a proportional term $u_i^P(k)$ and a integral term $u_i^I(k)$ that are derived from the error $e_i(k)$ between the actual throughput $Th_i(k)$ and the SLO target Th_i^{SLO}. $u_i^P(k)$, $u_i^I(k)$ and $u_i(k)$ can be expressed as [21]:

$$u_i^P(k) = C_P \times e_i(k) \tag{6}$$

$$u_i^I(k) = u_i^I(k-1) + C_I \times e_i(k) \tag{7}$$

$$u_i(k) = u_i^P(k) + u_i^I(k) \tag{8}$$

To conveniently analyze the PI controller, we describe the system in the frequency domain (i.e., *z-domain*). As shown in Figure 2, we assume that $U_i(z)$ and $R_i(z)$ are the z-domain representations [21] of $u_i(k)$ and $r_i(k)$ respectively. As the time domain representations, $u_i(k_0)$ and $r_i(k_0)$ are the values of $u_i(k)$ and $r_i(k)$ at time $k = k_0$, which are the coefficients of the z term z^{-k_0} in $U_i(z)$ and $R_i(z)$ respectively. In this way, $u_i(k)$ and $r_i(k)$ can be encoded as the coefficients of the z terms in $U_i(z)$ and $R_i(z)$. To further study the PI controller in the control theory, we need to derive its *transfer function* that is defined as $G_i(z) = R_i(z)/U_i(z)$. Based on the formula 5, $G_i(z)$ can be derived as follows:

$$G_i(z) = \frac{b_i^k}{z - a_i^k} \tag{9}$$

Assuming $s_i(k)$ as the desired throughput in the SLO at the k^{th} interval and $S_i(z)$ as the z-domain representation of $s_i(k)$, the closed loop transfer function can be defined as $F_i(z) = R_i(z)/S_i(z)$, that is:

$$F_i(z) = \frac{[(C_P + C_I) \times z - C_P] \times G_i(z)}{(z-1) + [(C_P + C_I) \times z - C_P] \times G_i(z)} \tag{10}$$

Further, assuming that the system input $s_i(k)$ is a step change, the *steady-state error* [1] E_{ss} can be expressed as:

$$E_{ss} = \lim_{k \to \infty} (s_i(k) - r_i(k)) \tag{11}$$

where the steady-state value of $r_i(k)$ is the system output when the time is infinite. Based on Formula 10 and the final value theorem of Z-transforms [21], we have:

$$\begin{aligned} E_{ss} &= \lim_{z \to 1} (z-1)(S_i(z) - R_i(z)) \\ &= \lim_{z \to 1} \frac{(z-1)^2 \times (z - a^k) \times S_i(z)}{z^2 + [(C_P + C_I) \times b_i^k - (1 + a_i^k)] \times z + a_i^k - C_P \times b_i^k} \end{aligned} \tag{12}$$

According to Formula 12, PI control can have a zero steady-state error if the closed-loop system is stable.

Stability Consideration: The stability of a feed-back controller has a significant impact on the systems ability to converge to the target over time. According to the *stability theorem* of the control theory [21], the closed-loop system represented by $F_i(z)$ is stable if and only if the poles of $F_i(z)$ are inside the unit circle, which indicates the stability boundary in the complex coordinate plane. More specifically, for the PI controller of SASLO, the poles of $F_i(z)$ are the roots of the denominator of $F_i(z)$ that can be obtained by solving the following equation:

$$z^2 + [(C_P + C_I) \times b_i^k - (1 + a_i^k)] \times z + a_i^k - C_P \times b_i^k = 0 \tag{13}$$

Where C_P and C_I are determined according to the demands for settling time and maximum overshoot [2].

Settling Time: Settling time is the time it takes for the feedback control to reach the steady-state value for the purpose of convergence to the target. Shortening the settling time makes the controller more responsive.

In the PI controller of SASLO, we define the settling time as S_t and the maximum overshoot as M_o. To obtain the poles, we can change Formula 13 to $(z - p_1) \times (z - p_2) = 0$. If p_1 is a *dominant pole* [3] and expressed as a complex with the magnitude r and the angle θ, M_o is given as $r^{\Pi/|\theta|}$ [21]. In addition, to describe the settling time S_t, we first define a criterion of $c\%$ for convergence, which means that the feedback control reaches its steady-state value if the variation of the actual IO throughput lies within the $c\%$

1. Steady-state error refers to the difference between the steady-state value and the SLO target.

2. Maximum overshoot refers to the largest error between the actual performance and the steady-state value.

3. Dominant pole is the pole with the largest magnitude and determines the transient response of the feedback control.

(e.g., 2%) of the difference between the SLO target and the throughput obtained when the SLO is set. The settling time can thus be expressed as $\log(c/100)/\log r$ [21]. In this way, we can obtain the values of r and θ by setting the desired M_o (e.g., 10% of the throughput target) and S_t (e.g., 5 time epoches). And then, we have the value of p_1. Further, C_P and C_I can be resolved based on Formula 13 only if $|z| < 1$ is true.

3.3.3 IO Split Control

The IO split controller is responsible for reducing the negative impact of large requests on the accuracy and stability of the SLO enforcement. The reduction in IO size by the IO split controller can decrease the number of bytes read or written for the same number of requests. As shown in Figure 2, the requests issued by VM_i smaller than the IO size limit Si_i^{SLO} will not be split. Moreover, the IO split controller only performs the necessary IO split by tracking the change in IO latency measured over the last two consecutive intervals. If the increase in IO latency surpasses a prior determined allowable range, implying that requests may be severely queued or even blocked, then the IO size limiter will be triggered.

3.4 Automated Optimization Tool

The core idea of AOT is to recommend the appropriate values of SLO targets (i.e., Th_i^{SLO} and Si_i^{SLO}) for the consolidated VMs (i.e., $VM_i, 1 \leq i \leq N$) as the inputs to the SEE dedicated to each individual VM according to its user-customized QoS constraints.

For the parameter Th_i^{SLO}, we use the algorithm of throughput SLO coordination to seek appropriate values for all the consolidated VMs supported by the same storage server. As lineated in Algorithm 1, the inputs to the algorithm consist of the current throughput SLO targets (i.e., $Th_i^{SLO}, 1 \leq i \leq N$), SC rates and PF ratios derived for all the concurrent VMs over last fixed-length interval T_R (i.e., C_i and $F_i, 1 \leq i \leq N$) and the acceptable ranges of throughput SLO, SC rate and PF ratio specified in the SLO targets and QoS constraints for each VM. Throughput SLO coordination will try to improve the value Th_i^{SLO}, of course within the acceptable range of throughput SLO, on the premise that the SC rate of each consolidated VM (i.e., $C_i, 1 \leq i \leq N$) is larger than the upper bound of the acceptable range of SC rate (i.e., C_i'') and simultaneously the PF ratios of each consolidated VM (i.e., $F_i, 1 \leq i \leq N$) is smaller than the lower bound of the acceptable range of PF ratio (i.e., F_i')). However, if the minimum requirements for SC rate or PF ratio of one or more VM cannot be met (i.e., there are one or more consolidated VMs each of which obtains an actual SC rate smaller than the lower bound of the acceptable SC rate range or an actual PF ratio larger than the upper bound of the PF ratio range), the throughput SLO targets of consolidated VMs will be reduced as long as each newly reduced throughput SLO target is larger than or equal to the lower bound of the throughput SLO range. The decrement or increment of Th_i^{SLO} at a time is Δ_R that is set at 10% the range size of throughput SLO (i.e., $(Th_i'' - Th_i') * 10\%$) as default in the following experiments.

The algorithm of IO size limit coordination, as lineated in Algorithm 2, is used to make a better trade-off between IO size and the corresponding cost on the accuracy and stability of SLO enforcement. Similar to the logic of throughput SLO coordination, if the actual SC rate of each consolidated VM (i.e., $C_i, 1 \leq i \leq N$) is larger than the upper bound of the corresponding SC rate range (i.e., C_i'') and simultaneously the actual PF ratio of each consolidated VM (i.e., $F_i, 1 \leq i \leq N$) is smaller than the lower bound of the corresponding PF ratio (i.e., F_i')), the IO size limit (Si_i^{SLO}) will be increased at Δ_R (i.e., a memory page size, the default setting in Linux OS is 4KB) as long as the newly increased Si_i^{SLO} is smaller than or equal to the upper bound of the acceptable range of IO size limit. Si_i^{SLO} can be decreased at Δ_R if the minimum requirements for SC rate or PF ratio of one or more VMs cannot be met. It is noted that the IO size limit coordination carries out a main loop every T_S set at 5 minutes while the throughput SLO coordination runs every T_R set at 30 seconds. Thus, there is adequate time left for the throughput SLO coordination to seek appropriate throughput targets for consolidated VMs under the present settings of IO size limit that has been chosen by the IO size limit coordination.

Algorithm1: Throughput SLO Coordination (30 seconds level)

Require: The SC rate (C_i) and PF rate (F_i) obtained for VM_i and the IO throughput SLO (Th_i^{SLO}) for VM_i; the acceptable SC rate range [C_i', C_i''], PF rate range [F_i', F_i''] and throughput SLO range [Th_i', Th_i''] customized for VM_i.

```
1.  /*Initialization*/
2.  for i in [1, N] do
3.      Th_i^SLO = Th_i'
4.  /*Main loop continues only if at least one VM running*/
5.  while N > 0 do
6.      /*The constraints for decreasing Th_i^SLO */
7.      if ∃i, C_i < C_i' or F_i > F_i'' and ∃i, Th_i^SLO ≥ Th_i' + Δ_R then
8.          for i in [1, N] do
9.              if Th_i^SLO ≥ Th_i' + Δ_R then
10.                 Th_i^SLO = Th_i^SLO − Δ_R
11.     /*The constraints for increasing Th_i^SLO */
12.     if ∀i, C_i > C_i'' and F_i < F_i' and ∃i, Th_i^SLO ≤ Th_i'' − Δ_R then
13.         for i in [1, N] do
14.             if Th_i^SLO ≤ Th_i'' − Δ_R then
15.                 Th_i^SLO = Th_i^SLO + Δ_R
16.     /*Warning triggered by the current status of SLO enforcement*/
17.     if ∃i, C_i < C_i' or F_i > F_i'' and ∀i, Th_i^SLO < Th_i' + Δ_R then
18.         Warning with unexpected C_i or F_i
```

Algorithm2: IO Size Limit Coordination (5 minutes level)

Require: The SC rate (C_i) and PF rate (F_i) obtained for VM_i and the IO size limit (Si_i^{SLO}) set for VM_i; The acceptable SC rate range [C_i', C_i''] and PF rate range [F_i', F_i''] and the range of IO size limit [Si_i', Si_i''] customized for VM_i.

```
1.  /*Initialization*/
2.  for i in [1, N] do
3.      Si_i^SLO = Si_i'
4.  /*Main loop continues only if at least one VM running*/
5.  while N > 0 do
6.      /*The constraints for decreasing Si_i^SLO */
7.      if ∃i, C_i < C_i' or F_i > F_i'' and ∃i, Si_i^SLO ≥ Si_i' + Δ_S then
8.          for i in [1, N] do
9.              if Si_i^SLO ≥ Si_i' + Δ_S then
10.                 /*Reducing the cost incurred by larger request size*/
11.                 Si_i^SLO = Si_i^SLO − Δ_S
12.     /*The constraints for increasing Si_i^SLO */
13.     if ∀i, C_i > C_i'' and F_i < F_i' and ∃i, Si_i^SLO ≤ Si_i'' − Δ_S then
14.         for i in [1, N] do
15.             if Si_i^SLO ≤ Si_i'' − Δ_S then
16.                 /*Increasing Si_i^SLO for a larger request size */
17.                 Si_i^SLO = Si_i^SLO + Δ_S
18.     /* Warning triggered by the current status of SLO enforcement */
19.     if ∃i, C_i < C_i' or F_i > F_i'' and ∀i, Si_i^SLO < Si_i' + Δ_S then
20.         Warning with unexpected C_i or F_i
```

Optimized SLO targets including throughput SLO targets and IO size limits of consolidated VMs will replace the original settings configured for the corresponding cloud-computing servers by the network communication between AOT and COM. And SEE running on these cloud-computing servers will enforce these newly updated SLO targets. If throughput SLO targets and IO size limits for all the consolidated VMs cannot be reduced (otherwise the IO capacity will be lower than user demands) and there are still one or more VMs whose QoS constraints cannot be met, it indicates that the SLO targets and QoS constraints for consolidated VMs cannot be simultaneously met under the given storage resources. In this case, VM storage migration [26] may be triggered to reduce the load of underlying storage devices.

4 Implementation Challenges

To better understand the design and implementation of the SASLO framework, in this section we will explain the end-to-end VM-oriented control mechanism in SASLO based on the para-virtualized Linux established by Xen [11]. To support the cloud system equipped with heterogeneous storage devices, SASLO does not make any assumption about the special support of the underlying storage subsystem. A virtual disk is actually a black box from the viewpoint of SASLO's SLO enforcement engine (SEE) and individual VMs, resulting from resource management. Thus, SASLO is actually insensitive to the internal organization in the underlying storage subsystem, and thus compatible with different kinds of storage devices including SSD or the disk array organized as RAID-0, RAID-5 or RAID-6, etc.

4.1 End-to-end VM-oriented Control

The split driver model [7] , adopted by the current version of Xen [11], is designed to provide IO access for VMs by performing real IO operations on behalf of the guest OS. Specifically, an IO request issued by a VM is first sent to the frontend driver (i.e., *blkfront*) and simultaneously an event is sent by blkfront to notify the backend driver (i.e., *blkback*) for the corresponding virtual disk of the VM to redirect the request to the block layer. Then, the request is handled by the hypervisor IO scheduler where the SLO enforcement engine (SEE) is integrated. In SEE's viewpoint, requests issued by VMs are sent by the processes of blkback of which each has a process name containing the serial number for a specific VM assigned by the hypervisor. Once a blkback process of a VM is forked and forwards IO requests to the hypervisor IO scheduler, the SEE will recognize the newly active VM and establish a resource controller and a resource limiters dedicated to the VM. If the blkback processes for a VM terminate, the controller and limiter of the VM will be reclaimed. Thus, SASLO can establish an end-to-end VM-oriented control for each VM.

The SLO enforcement engine is able to handle requests from a VM with the controller and resource limiter dedicated to that VM. If the VM is migrated to another host, its SEE on the source host will be revoked and another SEE will be allocated to the VM on the destination host. Thus, if the cloud scheduler detects the malfunction of a host and makes all the VMs consolidated on the host migrate to another host, SASLO can work well when facing VM dynamic deployment. In addition, operations such as snapshot and recovery on VMs executed by hypervisor cannot interfere with the feedback control loops of these VMs created by their SEEs in SASLO. This is because the inputs to each SEE dedicated to a specific VM are the pre-specified SLOs for and the IO statistics obtained from the VM and the outputs are the upper bound of throughput and IO size limit for the VM, resulting in a VM-oriented end-to-end control mode. Thus, the control of each SEE is not only isolated from the control loops created by SEEs for other consolidated VMs but also not affected by IO operations incurred by hypervisor. Furthermore, because of the VM-oriented end-to-end control mode under the SASLO design, the SASLO model accuracy is not sensitive at all to any increase in the number consolidated VMs, as long as the IO capacity is adequate.

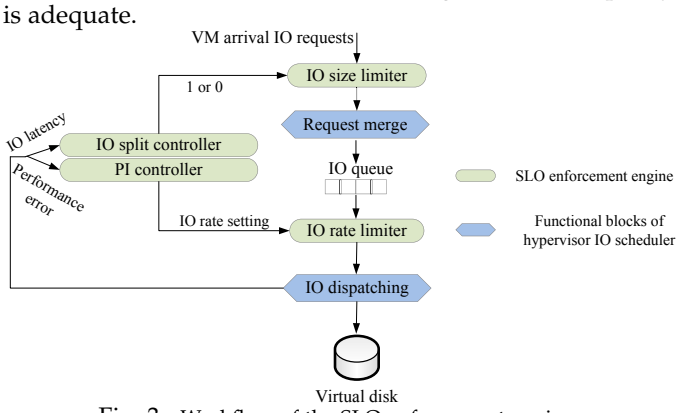

Fig. 3: Workflow of the SLO enforcement engine.

4.2 SLO Enforcement Engine

With the end-to-end VM-oriented control architecture, the SLO enforcement engine is able to handle the requests from a VM with the controllers and resource limiters dedicated to that VM. As shown in Figure 3, the IO size limiter carries out IO split by controlling the functional block of request merge in the hypervisor IO scheduler. Specifically, the IO size limiter adjusts the limit on the number of physical segments per request for the IO queue that is represented as the variable $max\_phys\_segments$ of the structure $request\_queue$ in the Linux kernel. When the total size of two mergeable requests exceeds the value of $max\_phys\_segments$, the request merge will not be performed. In this way, an originally large merged request is split into two smaller requests. The IO split controller can trigger the IO size limiter with a "1" signal while disable it with a "0" signal. To seek a better trade-off between the IO size and the corresponding cost of SLO enforcement, the IO split controller tracks the average IO latency obtained over the last two consecutive epoches, $latency(k-1)$ and $latency(k)$, for a VM (20 ms for each epoch). If the increase in the IO latency, expressed as the proportion of $(latency(k)-latency(k-1))/latency(k-1)$ (i.e., $latinc(k)$), goes outside the allowable range (e.g., $< 30\%$), the IO size limiter will be enabled. In this case, the value of $max\_phys\_segments$ at the $k+1^{th}$ epoch (i.e., $mm(k+1)$) will be set at $(mm(k))/(1+\alpha(k))$, where $\alpha(k)$ is the ratio of the SLO target to the actual IO throughput at the k^{th} epoch, only if the value of $mm(k+1)$ is larger than or equal to the lower bound of the user-customized range of IO size limit. Once the value of $latinc(k)$ falls within the allowable range

Storage subsystem	Type	Network	Configurations
Disk array	RAID0 on a SAN	10 Gbps	A 16-disk (7200RPM, 250GB) RAID 0 disk group
SSD	Direct-attached storage	—	Fusion-io ioScale 2, 825GB Multi Level Cell (MLC) Solid State Drive

TABLE 2: The configurations of storage subsystems for the experiments conducted in the SASLO evaluation.

and the value of the integral term $u_i^I(k)$ is zero, $mm(k+1)$ will be rolled back to the value of $mm(k-1)$ for a larger IO size. The IO rate limiter carries out IO throttling according to the IO rate setting determined by the PI controller.

5 Performance Evaluation

In this section, we present the results of a detailed evaluation of SASLO in a real cloud-computing server established by Xen 4.2 [11]. The cloud-computing server is supported by two storage servers each of which is equipped with different types of storage subsystems respectively, including a large-scale disk array over a SAN and a direst-attached SSD. This system enables us to verify the robustness and effectiveness of SASLO in the type of heterogeneous storage that is widely adopted in the enterprise data centers and clouds. The cloud-computing and two storage servers comprises three PowerLeader PR2760T servers of which each has 2 Intel Xeon E5620 quad-core processors, 12GB of RAM, and is equipped with a 10Gbps NIC (Intel 82598EB). The virtual disk attached to each VM for the SASLO evaluation has 80GB capacity, which is hosted on a LUN provided by the storage subsystems listed in Table 2. The cache mechanisms of the file system and the Xen hypervisor are enabled in the following experiments for evaluating SASLO in a typical cache configuration.

Objectives and reference systems of the evaluation: To comprehensively evaluate SASLO, we must assess the robustness and effectiveness of its key functional component, i.e., AOT (Automated Optimization Tool) and SEE (SLO Enforcement Engine). First, to evaluate the optimization mechanism of AOT (i.e., optimizing throughput SLO target and IO size limit for consolidated VMs under user-customized QoS), it is important to examine the impact of throughput SLO target and IO size limit at various levels on SLO compliance, performance fluctuation and performance error. Second, we compare SC (SLO Compliance) rate, PF (Performance Fluctuation) ratio and performance error (Section 3.1) of SLO enforcement under the SEE adopting PI controller (denoted by SEE) against those under the reference algorithms adopted by the existing state-of-the-art schemes. Time-stamp based IO control, referred to as *RClock*, is chosen as a reference system because of its ability to guarantee the IO throughput according the SLO target, RClock is adopted in the state of the arts including mClock[1] and SRP [2]. DRR [13], adopted by Pisces [3], is the second reference system in this evaluation. Leaky bucket [16], as an important technique of IO throttling, is also chosen as a reference system (denoted by LB). Integral control [21] can guarantee zero steady-state error, adopted by Triage [27], we use it as another reference system (denoted by Integral). Finally and importantly, based on the stable and accurate SLO enforcement supported by SEE, our evaluation will focus on verifying how well and effectively a trade-off between SLO targets and QoS constraints can be made. Since we can modulate the throughput SLO target and IO size limit of an individual VM within the user-customized range, the QoS metrics in the following experiments refer to the averages among the measurements of SC rate and PF ratio obtained according to different SLO targets.

Workloads: We adopt Filebench [28] in each VM to generate macro-benchmarks. Filebench is used to mimic the workloads of a web server, a file server and a mail server by generating the IO stream with different IO characteristics controlled by pre-specified parameters (i.e., the number of files to be used, the number of concurrently running threads issuing IO requests, average file size and IO size). The parameters of each workload are listed in Table 3 and the detailed description of these workloads is as follows:

Web server emulates a web service by synchronizing 100 threads to issue concurrent IOs with an average IO size of 512KB, accessing 50000 files with an average file size of 16KB. The file operations of the web server workload are basically reads, consisting of IO sequences such as open, read and close. The operations of writing log files of the web server are emulated by executing an append operation after an open operation.

File server emulates an NFS file service by synchronizing 50 threads to issue concurrent IOs with an average IO size of 1MB, accessing 50000 files with an average file size of 128KB. The file operations of the file server workload consist of read, write, create, delete, append and attribute on files.

Mail server emulates an email service by synchronizing 16 threads to issue concurrent IOs with an average IO size of 16KB, accessing 50000 files with an average file size of 16KB. The file operations of the mail server workload consist of IO sequences, e.g., open, read and close, or open, append and close, or delete operation.

Services	Files	Threads	File size	IO size
Mail Server	50000	16	16KB	16KB
Web Server	50000	100	16KB	512KB
File Server	50000	50	128KB	1MB

TABLE 3: Parameters for Filebench workloads

5.1 The effect of SLO targets on SLO enforcement

In this subsection, we examine the impact of SLO targets (i.e., throughput SLO target and IO size limit) at various levels on SLO compliance, performance fluctuation and the absolute value of performance error (denoted by AVPE) under different storage subsystems including the disk array and SSD as listed in Table 2. We run three VMs, VM_1, VM_2 and VM_3, deployed with web server, file server and mail server simulated by Filebench [28] respectively. These VMs concurrently access the storage subsystem.

Specifically, we let the throughput targets specified in SLOs for the three VMs increase linearly every 10-minute. Thus, we can observe the effect of different levels of the throughput target on SLO enforcement under SASLO. If the SLO target surpasses the average throughput obtained in the original Xen hypervisor (denoted by without SASLO), we consider the SLO infeasible. As shown in Figure 4, the actual throughput averaged over 10-minute intervals of the three VMs can approach their SLO targets with near-zero absolute values of performance error ($< 0.4\%$) until the SLO targets of one or more VMs become infeasible for both

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2584057, IEEE Transactions on Services Computing

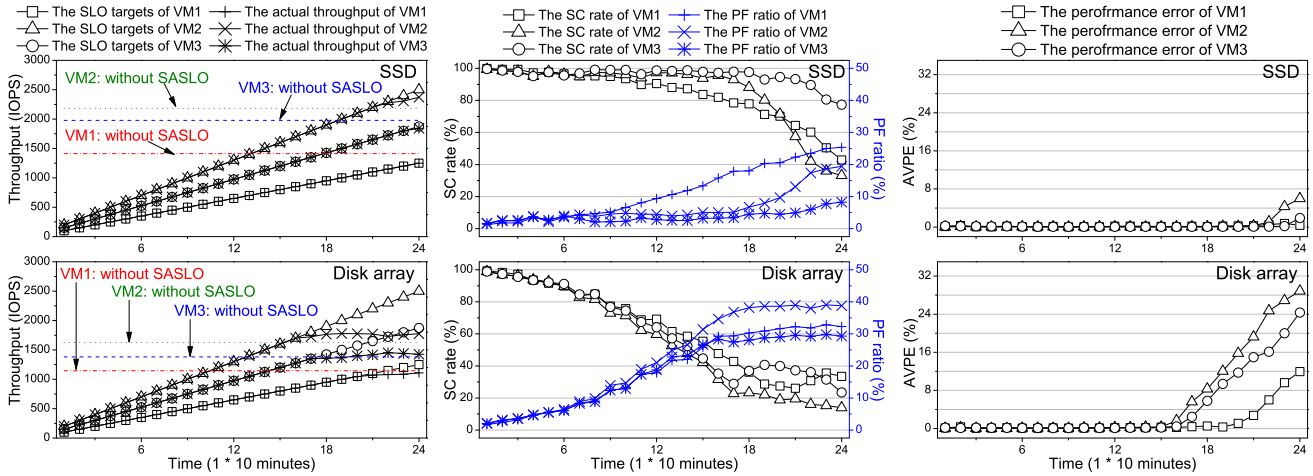

Fig. 4: The comparisons between the SLO targets and the actual IO throughput (figure on the left) and the SC rate (figure in the middle) and the AVPE (figure on the right) under SASLO for 3 concurrently running VMs as a function of the SLO target.

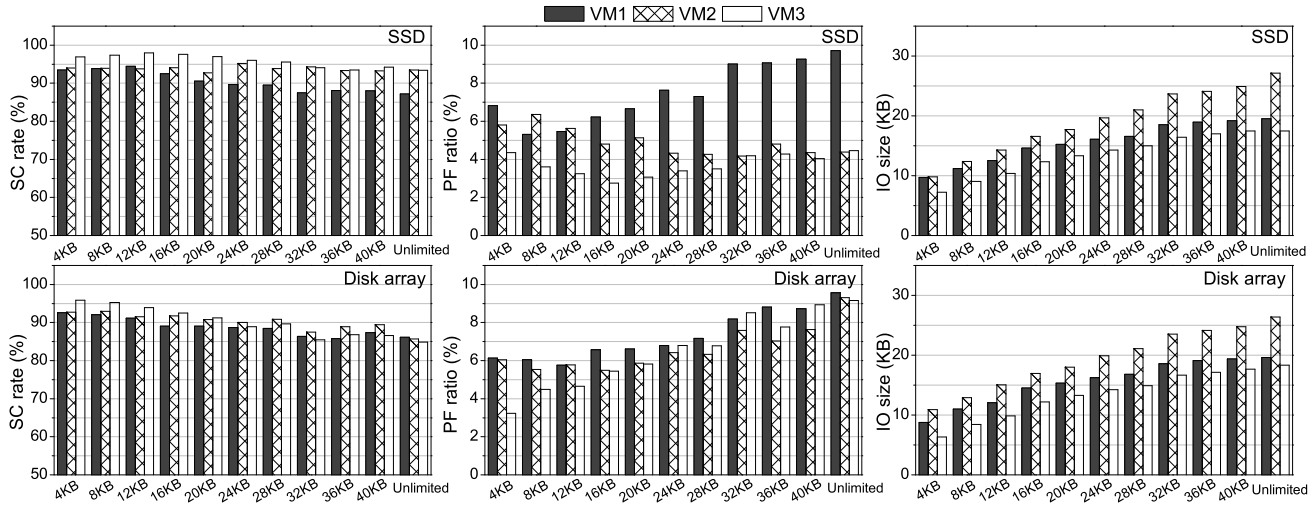

Fig. 5: The SC rate, PF ratio and actual IO size obtained by SASLO as a function of the IO size limit ranging from 4KB to unlimited with SSD and disk array respectively as the underlying storage subsystem.

the SSD and the disk array systems. In addition, the figure shows that the SC rate decreases and the PF ratio increases as the SLO target throughput increases. Second, we set the SLO targets of the three VMs at 400 IOPS, 600 IOPS and 800 IOPS respectively under the disk array system and set the SLO targets of the three VMs at 600 IOPS, 800 IOPS and 1000 IOPS respectively under the SSD system. As shown in Figure 5, the IO size limit for IO split ranges from 4KB to unlimited for each VM. And we show average values over a 20-minute period. Generally speaking, a smaller value of IO size limit is beneficial for guaranteeing a higher level of SLO compliance and a lower level performance fluctuation since a larger request is more likely to block the execution of other requests and thus induce more intensive IO interference. However, the value of performance error is maintained below 0.2% for all the values of IO size limit.

5.2 SLO Enforcement Engine

In this subsection, we provide a detailed evaluation of the SEE of SASLO under different storage subsystems including the disk array and SSD as listed in Table 2. Since the PI controller is the most important functional module that is responsible for fast converging the actual throughput to the dynamic throughput SLOs determined by AOT, we start by verifying the stability, accuracy and convergence of SLO enforcement dominated by PI controller. And then, we will assess the impact of VM scalability on the SASLO model accuracy. Finally, we will verify the contribution of stable and accurate throughput SLO enforcement provided by the SEE of SASLO to latency SLO enforcement.

5.2.1 *The Stability, Accuracy and Convergence of Throughput SLO Enforcement*

We run three VMs, VM_1, VM_2 and VM_3, deployed with web server, file server and mail server simulated by Filebench [28] respectively. These VMs concurrently access the storage subsystem.

The accuracy and stability in SLO enforcement: First, we compare the SEE of SASLO under PI controller with reference systems in SC rate, PF ratio and performance error obtained from statistics over a 15-minute period. We set the SLO targets of the three VMs at 400 IOPS, 600 IOPS and 800 IOPS respectively under the disk array system and set the SLO targets of the three VMs at 600 IOPS, 800 IOPS and 1000 IOPS respectively under the SSD system. As shown in Figure 6, SEE performs better than all other approaches under both the disk array and SSD systems. Specifically,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2584057, IEEE Transactions on Services Computing

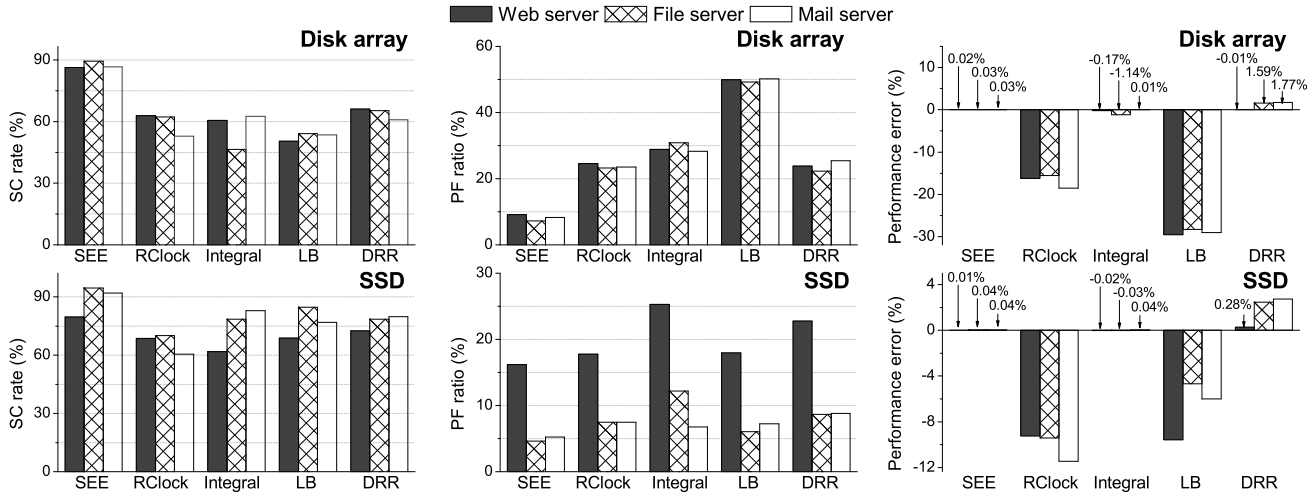

Fig. 6: Comparisons among SLO enforcement engine (SEE) and existing mainstream works in SC rate , PF ratio and performance error obtained over 15 minutes under disk array and SSD respectively.

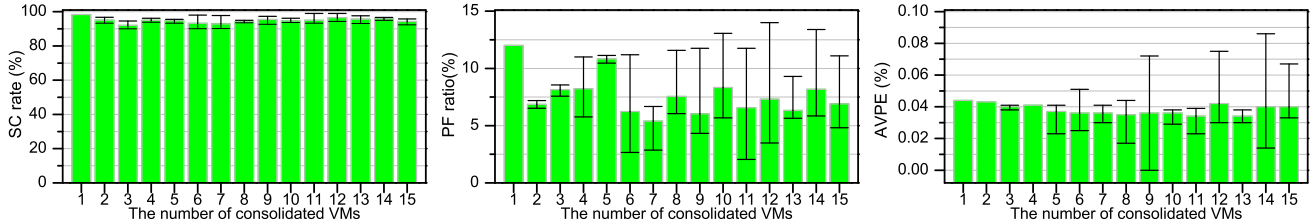

Fig. 7: SC rate, PF ratio and AVPE, obtained over 15 minutes, as a function of the number of consolidated VMs (1 through 15).

RClock and LB can't guarantee an adequately small absolute value of performance error. DRR is able to keep a smaller absolute value of performance error than 3%, however, resulting in a lower SC rate and higher PF ratio than SEE. In contrast, SEE and Integral are all able to maintain an almost zero performance error. However, Integral can produce a higher level of PF ratio than SEE. It is largely because that integral control can slow system response substantially in the cases of sudden changes in SLO target or disturbance inputs [21].

The settling time in convergence: We let the throughput targets for the three VMs increase linearly, starting from 100 IOPS, 200 IOPS and 150 IOPS respectively, by the increments of 50 IOPS, 100 IOPS and 75 IOPS for these VMs respectively every 3 seconds until the SLO targets of the three VMs reach 500 IOPS, 1000 IOPS and 750 IOPS respectively. We then continuously repeat the process over a 30-minute time period. We measure the settling time as the time period between the epoch of updating SLOs and the epoch of the first SLO compliance after last SLO update. We configure 5 time epochs in settling time for the PI control model of each VM. For the disk array system, the average numbers of the epochs required in convergence for the three VMs are 4.46, 3.83 and 4.45 respectively while 3.99, 3.53 and 3.67 are observed for the SSD system. The average length of an epoch for the three VMs is 30.70 ms, 29.02 ms and 31.42 ms under the disk array while 29.31 ms, 27.68 ms and 28.09 ms are obtained for the SSD. The surge of IO latency may delay the feedback control loop (run every 25 ms in the current design) especially when the storage device is overloaded.

5.2.2 *The impact of VM scalability*

In this section, we verify the impact of VM scalability (i.e., increasing the number of consolidated VMs) on the SASLO model accuracy, represented by two critical control parameters of the SLO compliance rate (SC rate) and the absolute value of performance error (AVPE). To this end, we conduct an experiment to measure the values of SC rate, AVPE and PF ratio for each VM playing the real production trace of WebSearch [29] as the number of consolidated VMs, supported by a storage server with an SSD, increases from 1 to 15. For the purpose of stressing IO contention, each VM plays the WebSearch trace as fast as possible and has the same throughput SLO of 1500 IOPS. This allows us to assess the impact of VM scalability on the SASLO model accuracy under severe IO contention. As shown in Figure 7, we mark the minimum, average and maximum values of SC rate, PF ratio and AVPE among all the consolidated VMs. It is observed from Figure 7 that the average values of SC rate fluctuate within a small range from 92.1% to 98.3% even when the number of consolidated VMs increases from 1 to 15. And the values of SC rate obtained for consolidated VMs in all the cases surpass 90%. Moreover, the average values of AVPE fluctuate within a very narrow range from 0.034% to 0.044% and the maximum AVPE falls below 0.087% when the number of consolidated VMs increases from 1 to 15. In addition, the average utilization of the storage device (i.e., the SSD) increases from 16.5% to 51.4% when the number of consolidated VMs increases from 1 to 15 and the maximum utilization of storage device ranges from 84.1% to 92.9%. These observations indicate that the SASLO model accuracy is rather insensitive to the VM scalability as long as the IO capacity is adequate (i.e., the maximum utilization $< 100\%$). Additionally, as shown in Figure 7, the maximum PF is smaller than 15% in all the cases and the VM scalability cannot significantly increase PF ratio. As a result, SASLO can also guarantee a low level of throughput fluctuation even when the number of consolidated VMs increases greatly.

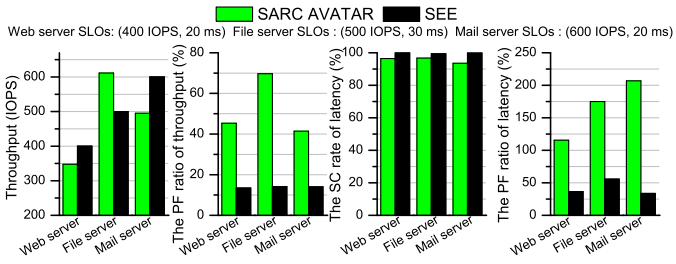

Fig. 8: Comparisons in the average throughput, the PF ratio of throughput and the SC rate and PF ratio of average latency per second obtained over 2 hours with 3 consolidated VMs deployed with the web server, file server and mail server respectively.

5.2.3 Throughput SLO & Latency SLO Enforcements

It is necessary to verify the effectiveness of the VM-oriented end-to-end proportional-integral IO control executed by SEE for the stability and accuracy of throughput SLO enforcement on the storage SLO guarantee in another important performance metric (e.g., IO latency). According to Little's law [30], the association among outstanding IOs, throughput and average latency can be expressed by:

$$Latency = Outstanding\ IOs/Throughput \quad (14)$$

Thus, stable and accurate throughput performance theoretically contributes to the stability and accuracy of latency SLO enforcement only if outstanding IOs can be well controlled. This is because unpredictable throughput fluctuation may intensify latency undulation based on Little's law, very likely resulting in latency SLO violation. The technique of latency SLO enforcement by outstanding IO controlling has been thoroughly studied by previous studies (e.g., Façade [12]). Hence, we adopt the Façade algorithm to guarantee the latency SLO for each individual VM by coordinating its IO queue length adapting to the deviation between the actually measured latency and the latency SLO.

We evaluate our approach based on the IO latency control executed by the Façade algorithm [12] (denoted by SEE) against the state of the art (i.e., SARC AVATAR [15]) on guaranteeing multi-metric storage SLOs (i.e., the throughput and latency SLOs for each workload), which does not focus on the stability and accuracy of throughput.

In more details, we run three VMs, VM_1, VM_2 and VM_3, deployed with web server, file server and mail server simulated by Filebench [28] respectively, concurrently accessing the disk array as listed in Table 2 under the control of SARC AVATAR [15] and SEE respectively. The throughput SLOs of the three VMs are 400 IOPS, 500 IOPS and 600 IOPS respectively and the latency SLOs of the three VMs are 20 ms, 30 ms and 20ms respectively. Thus, we can verify SEE's robustness and effectiveness in stable and accurate throughput SLO enforcement as well as alleviating the impact of throughput fluctuation on latency SLO enforcement.

As shown in Figure 8, the SC rate of average latency per second of the three VMs obtained during 2 hours under the control of SEE are 99.92%, 99.42% and 99.83%, which are higher than those obtained under SARC AVATAR [15], i.e., 96.42%, 96.75% and 93.58%. Moreover, as shown in Figure 8, the PF ratio values of average latency per second of the three VMs obtained during 2 hours under SASLO are also far lower than those obtained under SARC AVATAR. This means that SEE contributes significantly to improving the stability and accuracy of latency SLO enforcement. In addition, the actual throughput of the web server and the mail server under SARC AVATAR is smaller than the corresponding throughput SLOs. In contrast, SEE can accurately enforce throughput for each consolidated VM with negligible deviation (i.e., 400.16 IOPS, 500.02 IOPS and 600.25 IOPS for the three VMs). As a result, we believe that the SEE of SASLO can effectively limit the variability latency as well as support stable and accurate throughput SLO enforcement under the VM consolidation environment.

5.3 IO Capacity Optimization Under QoS Constraints

In this section, we will evaluate the robustness and effectiveness of SASLO in optimizing the IO capacity allocation decided by throughput SLO target and IO size limit under the user-customized QoS constraints in terms of SLO compliance and performance variability, which are measured by SC rate and PF ratio respectively.

Based on stable and accurate SLO enforcement supported by SEE, the AOT of SASLO can dynamically coordinate throughput SLO target and IO size limit for consolidated VMs according to the variation of SLO compliance and performance variability. Consequently, SASLO can achieve a satisfactory trade-off between high-capacity allocation (i.e., high throughput and large IO size) and user-customized QoS for consolidated VMs. To verify this ability of SASLO, we let three consolidated VMs (denoted by VM_1, VM_2 and VM_3) accessing the storage subsystems including the disk array and the SSD respectively. The three VMs are configured with the two configurations (denoted as *configuration 1* and *configuration 2*) each of which consists of the user-customized ranges of throughput SLO, IO size limit, SC rate and PF ratio for each VM as listed in Table 4 and Table 5. Configuration 1 adopts a higher level of QoS constraints than configuration 2. According to the logic of throughput SLO coordination and IO size limit coordination, the consolidated VMs under configuration 2 should gain a higher IO capacity than that gained under configuration 1 for given storage resources.

VM	Throughput SLO	IO size limit	SC rate	PF ratio
VM1	[400 IOPS, 600 IOPS]	[12KB, 32KB]	[80%, 90%]	[8%, 15%]
VM2	[600 IOPS, 800 IOPS]	[12KB, 44KB]	[85%, 90%]	[5%, 10%]
VM3	[800 IOPS, 1000 IOPS]	[12KB, 24KB]	[80%, 90%]	[5%, 10%]

TABLE 4: User-customized SLO targets and QoS constraints in the configuration 1.

VM	Throughput SLO	IO size limit	SC rate	PF ratio
VM1	[400 IOPS, 800 IOPS]	[12KB, 32KB]	[70%, 80%]	[15%, 20%]
VM2	[600 IOPS, 1200 IOPS]	[12KB, 44KB]	[75%, 80%]	[10%, 15%]
VM3	[800 IOPS, 1600 IOPS]	[12KB, 24KB]	[70%, 80%]	[10%, 15%]

TABLE 5: User-customized SLO targets and QoS constraints in the configuration 2.

As shown in Figure 9 and Figure 10, the actual throughput of the three consolidated VMs can be kept between the user-customized range under both configuration 1 and configuration 2. It is noted that SASLO can dynamically coordinate the throughput SLOs for all the consolidated VMs according to the variation of SC rate and PF ratio. Specifically, as shown in Figure 9, during the period from the time of $10*30$ seconds to the time of $15*30$ seconds, we can observe that the values of throughput SLO for all the three VMs are raised continuously by SASLO since the value of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2584057, IEEE Transactions on Services Computing

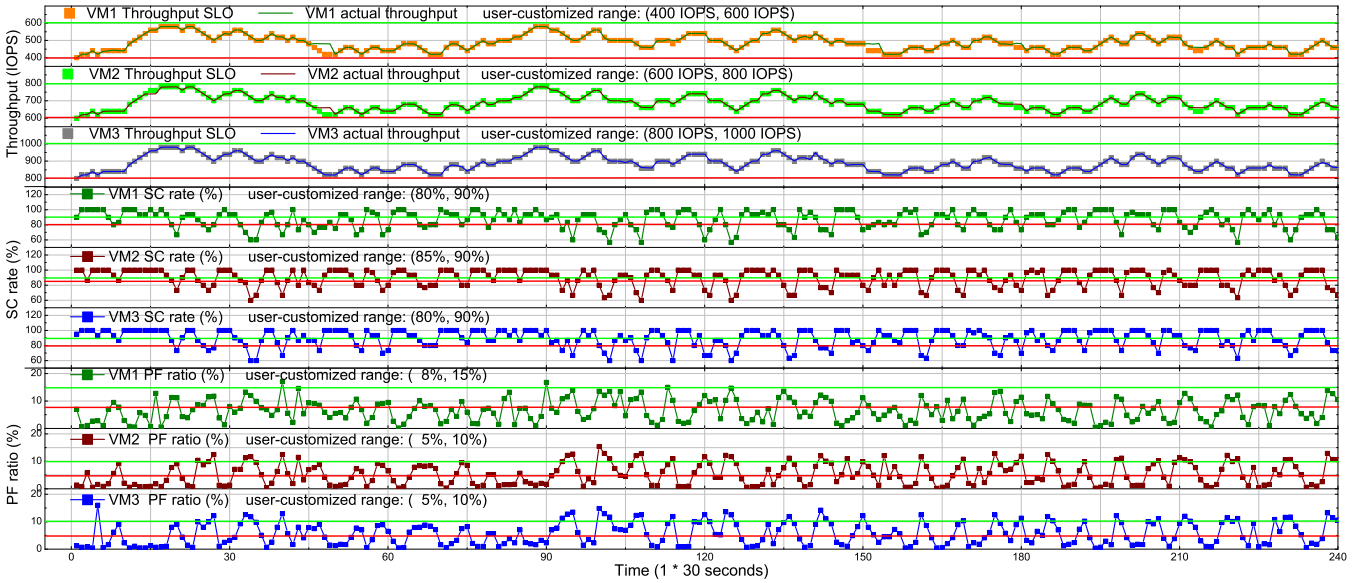

Fig. 9: The values of throughput SLO configured by throughput SLO coordination algorithm and the actual throughput, the variation of SC rate and PF ratio for the configuration 1 under the control of AOT. The lower bounds and upper bounds of user-customized ranges of throughput SLO, IO size limit, SC rate and PF ratio are marked with red lines and green lines respectively.

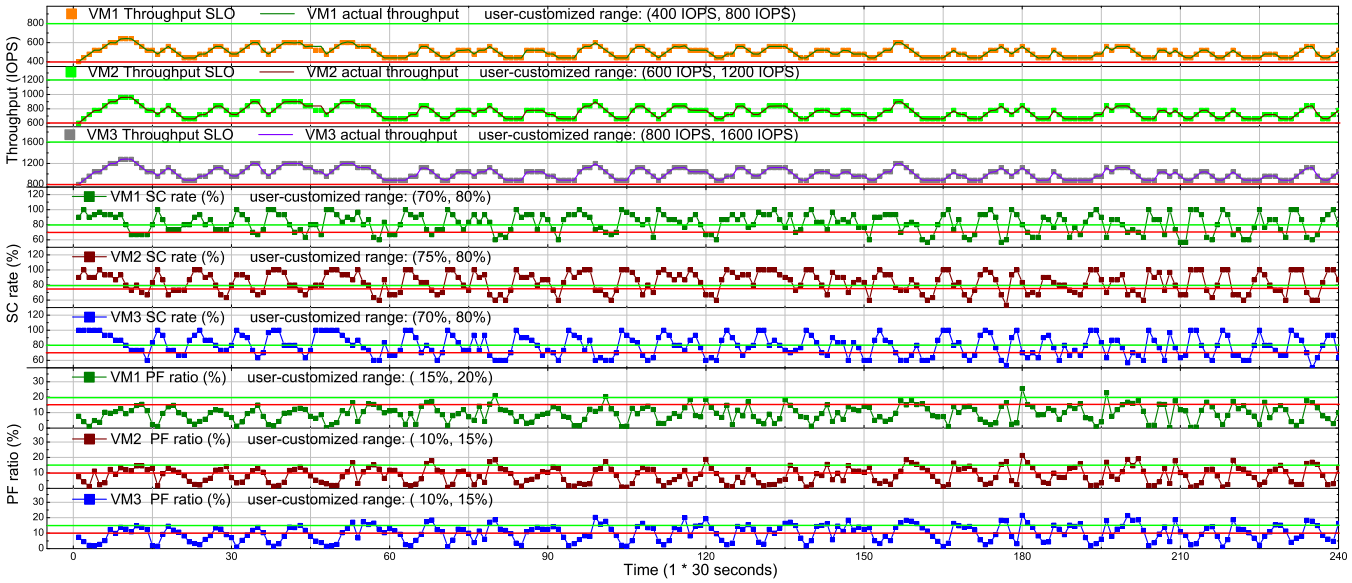

Fig. 10: The values of throughput SLO configured by throughput SLO coordination algorithm and the actual throughput, the variation of SC rate and PF ratio for the configuration 2 under the control of AOT. The lower bounds and upper bounds of user-customized ranges of throughput SLO, IO size limit, SC rate and PF ratio are marked with red lines and green lines respectively.

SC rate for each consolidated VM obtained during the time period is beyond the upper bound of the user-customized range and the value of PF ratio for each VM during the time period is below the lower bound of the user-customized range. At the time of $90 * 30$ seconds, the throughput SLOs for all the consolidated VMs can be observed decreasing in Figure 9 since the PF ratio for the VM_1 obtained during the 90^{th} 30-second interval is 16.8% which surpasses the upper bound of 15%. In this case, the IO capacity allocated to a VM can be influenced by another consolidated VM violating its requirements on SC rate or PF ratio. We can make a more reasonable combination of VMs consolidated on the same storage server in order to ensure a high utilization of the underlying storage devices as well as meet the user-customized QoS constraints. This is beyond the scope of our research and left for our future work.

Second, we reconfigure the configuration 2 for the three consolidated VMs accessing the disk array by relaxing the range of throughput SLO and degrading the requirements on SC rate and PF ratio. As shown in Figure 9 and Figure 10, the SC rate and PF ratio fluctuate more intensively under configuration 2 than those obtained under the configuration 1. Specifically, the variation ranges of SC rate for the three consolidated VMs under the configuration 1 are $[56.7\%, 100\%]$, $[60\%, 100\%]$ and $[60\%, 100\%]$ respectively while $[56.5\%, 100\%]$, $[53.3\%, 100\%]$ and $[50\%, 100\%]$ are obtained under the configuration 2. Moreover, the variation ranges of PF ratio for the three VMs are changed from $[0.2\%, 17.1\%]$, $[0.2\%, 15.4\%]$ and $[0.4\%, 16\%]$ to $[0.3\%, 25.6\%]$, $[0.8\%, 21.3\%]$ and $[1.1\%, 21.5\%]$. As a result, as listed in Table 6, the SC rates of VM_1, VM_2 and VM_3 obtained over the whole process of the experiment under

Storage	VM	Configuration 1				Configuration 2			
		Throughput (IOPS)	IO size (KB)	SC rate (%)	PF ratio (%)	Throughput (IOPS)	IO size (KB)	SC rate (%)	PF ratio (%)
Disk array	VM1	488.35	14.33	86.08	6.64	502.10	16.49	81.68	9.48
	VM2	687.23	16.81	89.47	5.40	753.00	21.15	83.07	8.50
	VM3	886.78	11.98	89.27	5.64	1003.23	13.31	79.44	10.02
SSD	VM1	733.28	13.72	88.64	8.49	876.56	16.80	79.45	14.16
	VM2	932.82	16.55	94.14	3.99	1168.56	21.63	90.89	4.58
	VM3	1133.08	11.84	96.00	2.82	1461.31	13.45	89.64	4.97

TABLE 6: The actual throughput and IO size averaged over a time period of 2 hours and the SC rate and PF ratio obtained over the time period under the configuration 1 and configuration 2 respectively.

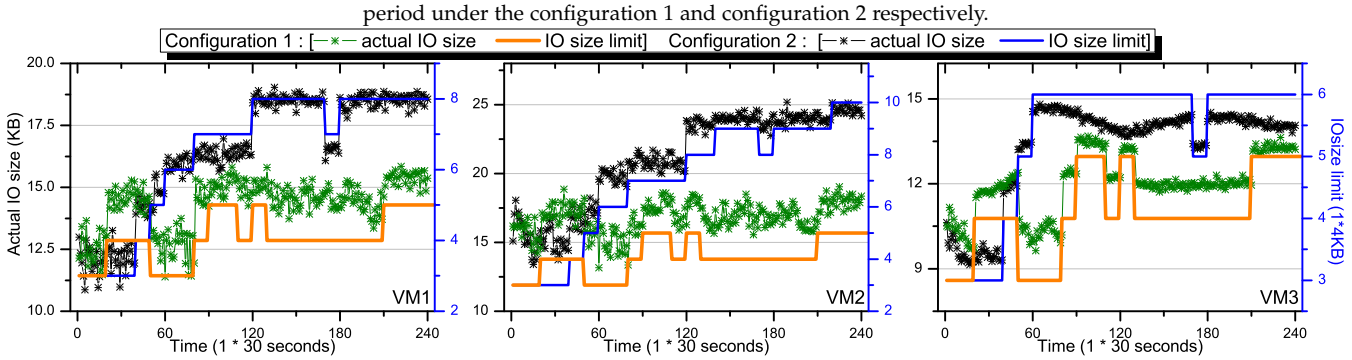

Fig. 11: The values of IO size limit configured by IO size limit coordination algorithm and the actual IO size.

the disk array (i.e., 2 hours) is 86.08%, 89.47% and 89.27% for the configuration 1 while 81.68%, 83.07% and 79.44% are obtained under the configuration 2. In addition, the PF ratios of the three VMs can be observed increasing from the values of 6.64%, 5.4% and 5.64% under the configuration 1 to the values of 9.48%, 8.50% and 10.02% obtained under the configuration 2. However, the actual throughput and IO size of consolidated VMs under the configuration 2 are larger than those obtained under the configuration 1. Specifically, the percentage increases of throughput for the three VMs are 2.82%, 9.57% and 13.13% respectively while the growth rates of IO size for these VMs are 15.07%, 25.82% and 11.1% respectively. The analogous phenomenon is also observed for the three consolidated VMs accessing the SSD. Thus, in spite of the lower level of QoS constraints under the configuration 2 than those under the configuration 1, higher IO capacity allocation can be gained instead under the configuration 2. More importantly, as listed in Table 6, the QoS constraints in terms of SC rates and PF ratios for consolidated VMs are met for both configure 1 and configure 2 under SASLO. This implies that SASLO is able to optimize IO capacity allocation for consolidated VMs under the user-customized QoS constraints.

In addition, it is observed in Figure 9 and Figure 10 that the SEE of SASLO can perfectly enforce throughput SLO configured by AOT since the PI controller dedicated to each individual VM can gain almost zero performance error under VM consolidation environment. In addition, as shown in Figure 11, the trend of actual IO size variation can also be controlled well under the setting of IO size limit coordinated by the IO size limit coordination algorithm.

6 Conclusion

In this paper, we focus on the issue of optimizing IO capacity allocation among consolidated VMs under the constraints of SLO compliance and performance variability. This is a great challenge since the IO capacity allocation among consolidated VMs must adapt to highly variable IO characteristics, IO interference among VMs and unstable IO capacity of storage devices. To address this challenge, we propose an end-to-end VM-oriented control framework that is able to dynamically coordinate IO capacity allocation according to the actual levels of SLO compliance and performance fluctuation so as to enhance storage utility utilization under the user-customized QoS constraints. The procedure of IO capacity allocation coordination can run as a throughput SLO series and an IO size limit series for each individual VM based on the stable, accurate and fast convergent SLO enforcement that is executed by VM-oriented PI controller and IO split controller. Our comprehensive prototype evaluation of SASLO, under different types of storage subsystems including disk array and SSD, demonstrates that SASLO can make a satisfactory trade-off between high IO capacity allocation and user-customized constraints of SLO compliance and performance variability.

Acknowledgments

This work is supported by the National High Technology Research and Development Program of China (863 Program) No.2013AA013203, No.2015AA015301, No.2015AA016701; NSFC No.61472153, No.61402189; US NSF grant: CNS-1116606 and CNS-1016609.

References

[1] A. Gulati, A. Merchant, and P. J. Varman, "mClock: handling throughput variability for hypervisor IO scheduling," in *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[2] A. Gulati, G. Shanmuganathan, X. Zhang, and P. Varman, "Demand based hierarchical QoS using storage resource pools," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2012.

[3] D. Shue, M. J. Freedman, and A. Shaikh, "Performance isolation and fairness for multi-tenant cloud storage," in *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[4] A. Gulati, I. Ahmad, and C. A. Waldspurger, "PARDA: proportional allocation of resources for distributed storage access," in *Proccedings of the conference on File and storage technologies (FAST)*, 2009.

[5] (2014) Windows azure sla. [Online]. Available: http://www.microsoft.com/windowsazure/zh/tw/sla/

[6] J. Sugerman, G. Venkitachalam, and B. H. Lim, "Virtualizing I/O devices on vmware workstation's hosted virtual machine monitor," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2001.
[7] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson, "Safe hardware access with the Xen virtual machine monitor," in *OASIS*, 2004.
[8] R. Russell, "virtio: towards a de-facto standard for virtual I/O devices," *SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.
[9] A. Gordon, N. Amit, N. HarEl, M. B. En-Yehuda, A. Landau, A. Schuster, and D. Tsafrir, "ELI: baremetal performance for I/O virtualization," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.
[10] J. Kang, B. Zhang, T. Wo, C. Hu, and J. Huai, "MultiLanes: providing virtualized storage for OS-level virtualization on many cores," in *Proccedings of the conference on File and storage technologies (FAST)*, 2014.
[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2003.
[12] C. R. Lumb, A. Merchant, and G. A. Alvarez, "Façade: Virtual storage devices with performance guarantees," in *Proccedings of the conference on File and storage technologies (FAST)*, 2003.
[13] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 375–385, 1996.
[14] W. Jin, J. S. Chase, and J. Kaur, "Interposed proportional sharing for a storage service utility," *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, pp. 37–48, 2004.
[15] J. Zhang, A. Riska, A. Sivasubramaniam, Q. Wang, and E. Riedel, "Storage performance virtualization via throughput and latency control," *ACM Transactions on Storage (TOS)*, vol. 2, no. 3, pp. 283–308, 2006.
[16] D. D. Chambliss, G. A. Alvarez, P. Pandey, D. Jadav, J. Xu, R. Menon, and T. P. Lee, "Performance virtualization for large-scale storage systems," in *Proceedings 22nd International Symposium on Reliable Distributed Systems (SRDS)*, 2003.
[17] J. C. R. Bennett and H. Zhang, "WF2Q: Worst-case fair weighted fair queueing," 1996.
[18] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," 1994.
[19] S. Suri, G. Varghese, and G. Chandramenon, "Leap forward virtual clock: A new fair queueing scheme with guaranteed delay and throughput fairness," 1997.
[20] P. Goyal, H. M. Vin, and H. Cheng, "Start-Time Fair Queuing: A scheduling algorithm for integrated services packet switching networks," *IEEE/ACM Transactions on Networking*, vol. 55, pp. 690–704, 1997.
[21] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2004.
[22] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger, "Argon: performance insulation for shared storage servers," in *Proccedings of the conference on File and storage technologies (FAST)*, 2007.
[23] J. C. Wu and S. A. Brandt, "The design and implementation of Aqua: an adaptive quality of service aware object-based storage device," in *Proceedings of the IEEE conference on Mass Storage Systems and Technologies (MSST)*, 2006.
[24] A. Povzner, T. Kaldewey, S.Brandt, R. Golding, T. M. Wong, and C. Maltzahn, "Guaranteed disk request scheduling with fahrrad," in *Proceedings of the 3th European conference on Computer systems (EuroSys)*, 2008.
[25] E. Thereska, H. Ballani, G. O'Shea, T. Karagiannis, A. Rowstron, T. Talpey, R. Black, and T. Zhu, "IOFlow: A software-defined storage architecture," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2013.
[26] A. Mashtizadeh, E. Celebi, T. Garfinkel, and M. Cai, "The design and evolution of live storage migration in vmware esx," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2011.
[27] M. Karlsson, C. Karamanolis, and X. Zhu, "Triage: Performance differentiation for storage systems using adaptive control," *ACM Transactions on Storage (TOS)*, vol. 1, no. 4, pp. 457–480, 2005.
[28] R. Mcdougall. (2014) A prototype modelbased workload for file systems, work in progress. [Online]. Available: http://solarisinternals.com/si/tools/filebench/filebenchnasconf.pdf
[29] (2015) Storagetrace, storage performance council (umass trace repository). [Online]. Available: http://traces.cs.umass.edu/index.php/Storage
[30] J. Little, "Na proof for the queuing formula: L= λw," *Operations research*, vol. 9, no. 3, pp. 383–387, 1961.

Ning Li received the B.S. degree in Computer Engineering from Nanjing University of Science and Technology (NUST, China) in 2004. He is currently a doctoral student in School of Computer, Huazhong University of Science and Technology (HUST) in China. His research interests include Virtualization, Quality of Service, and Cloud storage.

Hong Jiang received the B.Sc. degree in Computer Engineering in 1982 from Huazhong University of Science and Technology, Wuhan, China; the M.A.Sc. degree in Computer Engineering in 1987 from the University of Toronto, Toronto, Canada; and the PhD degree in Computer Science in 1991 from the Texas A&M University, College Station, Texas, USA. Since August 1991 he has been at the University of Nebraska-Lincoln, Lincoln, Nebraska, USA, where from 2001 to 2007 he served as Vice Department Chair of and is Willa Cather Professor of Computer Science and Engineering. He is concurrently in Department of Computer Science and Engineering, University of Texas at Arlington, USA. His present research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, performance evaluation.

Dan Feng received her B.E. degree and Master degree in Computer Science, and Ph.D. degree in computer engineering from Huazhong University of Science and Technology (HUST, China). After graduation, she has been working at the Huazhong University of Science and Technology (HUST) in China, and she is a professor of computer science and engineering currently. Her computer architecture, computer storage systems and parallel I/O, cluster, cloud storage, etc.

Zhan Shi received her B.E. degree and Master degree in Computer Science, and Ph.D. degree in computer engineering from Huazhong University of Science and Technology (HUST, China). He is an associate professor of computer science and engineering at the Huazhong University of Science and Technology (HUST) in China. Her interests include computer architecture, distribute file system, and parallel I/O storage systems, etc.