PSLO: Enforcing the X^{th} Percentile Latency and Throughput SLOs for Consolidated VM Storage

Ning Li
School of Computer
Huazhong University of Science and Technology
leebellwind@hust.edu.cn

Hong Jiang
Dept. of CSE
University of Texas at Arlington
hong.jiang@uta.edu

Dan Feng * Zhan Shi
WNLO, School of Computer
Huazhong University of Science and Technology
dfeng@hust.edu.cn/zshi@hust.edu.cn

Abstract

It is desirable but challenging to simultaneously support latency SLO at a pre-defined percentile, i.e., the X^{th} percentile latency SLO, and throughput SLO for consolidated VM storage. Ensuring the X^{th} percentile latency contributes to accurately differentiating service levels in the metric of the application-level latency SLO compliance, especially for the application built on multiple VMs. However, the X^{th} percentile latency SLO and throughput SLO enforcement are the opposite sides of the same coin due to the conflicting requirements for the level of IO concurrency. To address this challenge, this paper proposes PSLO, a framework supporting the X^{th} percentile latency and throughput SLOs under consolidated VM environment by precisely coordinating the level of IO concurrency and arrival rate for each VM issue queue. It is noted that PSLO can take full advantage of the available IO capacity allowed by SLO constraints to improve throughput or reduce latency with the best effort. We design and implement a PSLO prototype in the real VM consolidation environment created by Xen. Our extensive trace-driven prototype evaluation shows that our system is able to optimize the X^{th} percentile latency and throughput for consolidated VMs under SLO constraints.

1. Introduction

Cloud service (e.g., Amazon EC2 [1]) has been widely adopted as public utilities for users who pay for computing and storage capacity by renting *virtual machines (VMs)* on an as-needed basis. Many organizations or enterprises deploy enterprise-class applications in tens, hundreds or even thousands of VMs. Thus, the required latency SLO (service level objective) compliance ratio of a user application deployed on multiple VMs depends on the latency at a specific percentile, i.e., the X^{th} *percentile latency*, for these VMs.

For example, consider a user application built with 100 disjoint components that are each deployed in a different VM and a request of this application must collect responses from these 100 components in parallel. To guarantee a 90% request latency SLO compliance for this user application, we must ensure that the 99.9^{th} percentile latency SLO be met for all these 100 VMs (i.e., $99.9\%^{100} > 90\%$). Clearly, the higher the percentile of the latency SLO is guaranteed for each VM, the higher the latency SLO compliance can be obtained for the application deployed on a specific number of VMs. Thus, increasing attention has been drawn to the 99^{th}, 99.9^{th} or even 99.99^{th} percentile latency requirement in recent years [6, 15, 19, 31, 32], which can potentially support a high request latency SLO compliance for the application deployed in a large-scale VM cluster.

However, there is a conflict between the X^{th} percentile latency SLO and throughput SLO enforcements for VMs under shared storage. The attainment of the former usually and necessarily trades substantial IO concurrency for reduced IO waiting time while the latter requires an adequately high level of IO concurrency to meet the throughput SLO. This means that seeking a latency SLO in a higher percentile than required often leads to throughput SLO violation, and vice versa. Thus, it is desirable to enforce the X^{th} percentile latency or throughput SLO for each individual VM in a precise fashion. That is, the latency (throughput) SLO should be guaranteed for the user application, conditioned on a prespecified throughput (latency) SLO compliance.

Unfortunately, it is a great challenge to enforce the X^{th} percentile latency for consolidated VMs in a precise fashion. First, the X^{th} percentile latency is susceptible to many factors including contention for shared resources (e.g., network bandwidth and storage, etc.) [29, 34], queuing [22, 26] and highly variable workload IO behavior [34]. These factors can

* Corresponding author: dfeng@hust.edu.cn

EuroSys '16, April 18–21, 2016, London, United Kingdom
 ISBN 978-1-4503-4240-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2901318.2901330

exacerbate the variability of the VM issue queue [1] length and thus lead to an unpredictable IO waiting time. Second, the X^{th} percentile latency is a type of complicated statistical metric for multiple consolidated VMs that may each define latency SLOs at different percentiles, resulting in the difficulties of measurement and forecast. And lastly but most importantly, there is a lack of an effective control mechanism to precisely coordinate the X^{th} percentile latency.

Another key issue is how to enforce the fairness of throughput allocation among consolidated VMs under the X^{th} percentile latency SLO constraints. Different VMs often have different latency SLO violations even with the same SLO target, largely due to their distinctive IO characteristics (i.e., request size, IO rate, the degree of sequentiality, read & write ratio, etc.). For example, a VM VM_A running a sequential workload can very likely have more opportunities to increase the level of IO concurrency to enhance its throughput than a VM VM_B running a random workload if VM_A and VM_B have the same X^{th} percentile latency SLO, since sequential IOs usually have lower latency than random IOs. This difference must not be neglected particularly if VM_A encroaches on the reserved resources that should have been allocated to VM_B, which will very likely result in VM_B's throughput SLO violation.

Existing approaches mainly focus on the reduction of the tail latency at a specific percentile (e.g., 99.9^{th}) by making use of duplicate requests [6, 27, 31], adaptive replica selection [26] or the combination of rate limiting and priority-based resource allocation [34]. However, these solutions do not address the issue of simultaneous SLO enforcement of the X^{th} percentile latency and throughput, and the aforementioned issue of unfair throughput sharing under the X^{th} percentile latency SLO constraints. As a result, they do not meet the storage QoS requirements for both the throughput target and latency SLO compliance for the application workload that depends on the parallel IO processing of multiple VMs under shared storage infrastructure.

In this paper, we propose PSLO, a framework that aims to optimize storage performance for consolidated VMs under the two-dimensional SLO constraints of the X^{th} percentile latency and throughput SLO targets for each VM. These SLO targets are determined by the application-level SLOs, i.e., latency SLO compliance and throughput SLO. The optimized storage performance for consolidated VMs leads to an improved performance of the applications deployed on these VMs. A key idea of PSLO is to precisely enforce the X^{th} percentile latency SLO by adaptively configuring the level of IO concurrency based on tracking the level of latency SLO violations for each and every individual VM. PSLO adopts integral control from the classic control theory [8, 14] to accurately enforce throughput SLOs for consolidated VMs. The integral control is able to guarantee the actual throughput converging to the SLO target. To our best knowledge, *PSLO is the first solution to enforce the VM-level X^{th} percentile latency SLO in the presence of another important SLO, the throughput SLO, especially for a typical application deployed across multiple VMs.*

We implement a prototype of PSLO in the Xen hypervisor [3] and evaluate our framework with real production [2, 18] and synthetic workload traces under a consolidated VM environment. Our extensive trace-driven evaluation shows that PSLO can support the X^{th} percentile latency and throughput SLOs for consolidated VMs and take full advantage of spare resources to optimize these two measures of storage performance for each VM under SLO constraints.

The rest of the paper is organized as follows. Section 2 first motivates our study of storage performance optimization under the SLO constraints, and then reviews the related works. The architecture and design of PSLO are described in Section 3. Section 4 presents the implementation issues. Detailed performance evaluation of PSLO on a real system is presented in Section 5. We conclude the paper with an overview for future work in Section 6.

2. Motivation and Related Work

We first investigate the impact of IO concurrency on the X^{th} percentile latency and throughput to motivate our study. We then present the related works of SLO-based storage sharing and review the limitations of the existing QoS techniques for the consolidated VM environment.

2.1 Motivation

A modern application deployed in a cloud environment can be built from multiple disjoint components, of which each can be deployed in a VM under shared storage infrastructure [15]. Thus, the variability in the latency distribution of the application can be magnified at the VM level since the application request latency is affected by the responsiveness of each corresponding VMs. The combination of the X^{th} percentile latency [2] of each VM and the number of VMs on which an application is deployed decides the application-level latency QoS.

To intuitively illustrate the impact of IO concurrency on the X^{th} percentile latency and throughput and thus motivate our study, we let four consolidated VMs each run one of the four different production trace workloads (i.e., TPC-E, MSN, Exchange and WebSearch) [2, 18] respectively and access their respective virtual disks deployed on a 16-disk RAID-0 disk array, with their IO capacity being limited below 150 IOPS. We continuously increase the level of IO concurrency for each VM by increasing the upper bound of the number of read or write requests in each VM issue queue (denoted by D) from 1 to 4. For the VM running the TPC-E

[1] A VM issue queue represents a set/array of pending IOs maintained by each individual VM at the device.

[2] The X^{th} percentile latency refers to the request latency seen by a VM at the X^{th} percentile, which is resulted from the underlying network and storage IO delay.

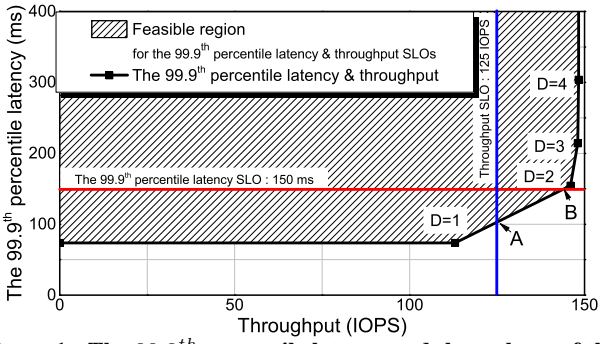


Figure 1: The 99.9^{th} percentile latency and throughput of the VM running the TPC-E workload as a function of the upper bound D of the number of read or write requests in the VM issue queue (i.e., the level of IO concurrency) ($D = 1, 2, 3, 4$).

workload, as shown in Figure 1, the curve depicting the 99.9^{th} percentile latency and throughput as a function of D (denoted by the performance & concurrency curve) actually defines the feasible region (i.e., the shaded region above the curve) in which the simultaneous enforcement of the 99.9^{th} percentile latency and throughput SLO targets (i.e., the two-dimensional SLO) is feasible. If the 99.9^{th} percentile latency and throughput SLO targets (e.g., 150 ms and 125 IOPS) can be simultaneously enforced accurately, they are represented by the two perpendicular lines of the red and blue colors respectively. Thus, we obtain two optimization settings (i.e., A and B). Setting A is located at the cross point between the performance & concurrency curve and the blue line, which represents the attainable minimum 99.9^{th} percentile latency under the throughput SLO constraint. Similarly, setting B is located at the cross point between the performance & concurrency curve and the red line, signifying the achievable maximum throughput under the 99.9^{th} percentile latency SLO constraint. The significance of this observation is that it is feasible to optimize the X^{th} percentile latency or throughput under their SLO constraints for each VM only if the red line and blue line cross each other within the feasible region. Otherwise, the X^{th} percentile latency and throughput SLOs cannot be met simultaneously. More importantly, the performance & concurrency curve in Figure 1 indicates the feasibility of optimizing the trade-off between the X^{th} percentile latency and throughput by controlling the level of IO concurrency.

2.2 Related Work

In what follows we introduce the existing approaches to throughput-based storage sharing, which followed by an examination of the existing schemes supporting latency SLO under shared storage infrastructure.

Throughput-based Storage Sharing: The existing approaches of throughput-based storage sharing can be broadly divided into three categories. The first is the class of approaches that support proportional sharing, such as PARDA [11], Argon [28], Aqua [30], Fahrrad [23] and SFQ(D) [16]. Proportional-sharing based solutions commonly use techniques such as request queue length control or disk time utilization reservation to enforce proportional-share fairness among co-scheduled workloads. The second class of solutions aim for IO isolation and include SARC [33], PriorityMeister [34] and Triage [17], etc. To provide IO isolation, these solutions adopt IO throttling techniques, such as Leaky bucket [5], deficient round robin (DRR) [24] and start-time fair queuing (SFQ) [9], etc., to limit throughput among concurrently running workloads. The third is the class of algorithms supporting max-min fairness of throughput sharing among clients, such as mClock [12], SRP [13] and Pisces [25]. Our work uses a feedback control solution combining the IO throttling technique and integral control [8, 14] to precisely enforce throughput SLOs for consolidated VMs with near-zero deviation.

Latency SLO Support: Two main categories of solutions comprise the state-of-the-art schemes supporting latency SLO, namely, (1) schemes providing average latency SLO and (2) schemes focusing on reducing tail latency at a specific percentile (e.g., at the 99.9^{th} percentile).

The first is the class of approaches that support average latency SLO for consolidated workloads, such as Façade [21], Triage [17] and pClock [10]. In contrast, PSLO is designed to accurately bound the X^{th} percentile latency for each VM according to its SLO target, which enables the latency SLO compliance of a user application built on one or more VMs to be accurately guaranteed. The second category focusing on tail latency can also be divided into three groups according to different implementations and techniques.

The first is the group of approaches that make use of multiple replicas servicing the duplicated requests, including the studies of [6, 27], D-SPTF [20], Kwiken [15], CosTLO [31] and C3 [26]. An extensive verification is made in the study reported in [27] to obtain a specific server load threshold under which the redundancy can be used effectively to reduce the mean and tail latencies. The study of [6] combines the techniques of request reissues, load balancing of micro-partitions and selective replication to reduce tail latency. D-SPTF [20] uses the technique of request reissue to reduce tail latency by dynamically selecting the server with a high speed interconnect. Kwiken [15] focuses on tail latency of a large-scale service built on multiple disjoint components. It formulates the problem of tail latency reduction over a general processing DAG (directed acyclic graph) as a multi-layer optimization on the number of reissues over individual stages. CosTLO [31] addresses the trade-off between the control target of the request latency variability and the acceptable cost of redundant requests. And C3 [26] improves the mean and median latency and the 99.9^{th} percentile tail latency by the techniques of adaptive replica selection, distributed rate control and backpressure.

The second group refers to solutions such as PriorityMeister [34] and Cake [29] that enforce tail latency by multi-resource scheduling. PriorityMeister [34] adopts the

techniques of IO throttling and priority-based IO scheduling among consolidated workloads across multiple stages (i.e., a network transmission and a shared storage stage) to enforce tail latency for bursty workloads. Cake [29] can coordinate multi-resource allocation (CPU and storage resources) across a multi-layer software stack consisting of HBase and HDFS to enforce the 99^{th} percentile tail latency.

The third group of approaches, such as Stout [22], can cut down request latency by congestion control adapting to storage-layer performance variation.

Although the above solutions can reduce tail latency or alleviate latency variability to different degrees, they cannot support a more accurate and flexible latency SLO, i.e., the X^{th} percentile latency SLO for consolidated VMs under shared storage infrastructure, which is proposed in this paper. Our work aims at supporting consolidated VMs running with different X^{th} percentile latency and throughput SLOs so as to make a more accurate and effective performance differentiation for the user applications with distinct scales (i.e., the number of VMs on which an application is deployed). More importantly, our proposed system PSLO is able to take full advantage of the available IO capacity allowed by SLO constraints to increase throughput or reduce latency for each VM under different optimization policies.

3. Architecture and Design

PSLO is designed for the consolidated VM environment that consists of a large number of application servers with each consolidating multiple VMs under a shared storage infrastructure. There is a hypervisor (e.g., Xen hypervisor [3]) running on each application server to monitor and harness the VMs consolidated on the server. PSLO is integrated into each hypervisor to enforce the X^{th} percentile latency and throughput SLOs for each individual VM. It views the underlying service infrastructure (e.g., network transformation and storage service) for a host as a black box, with the assumption that IO isolation among application servers is provided by a host-level approach that focuses on fair IO sharing, such as PARDA [11], Argon [28] or SFQ(D) [16], which cannot support VM-level IO isolation.

3.1 The Architecture of PSLO

It is non-trivial to simultaneously enforce the X^{th} percentile latency and throughput SLOs for the consolidated VMs. Request execution can be delayed by network congestion or IO contention at the underlying storage devices (e.g., competing for resources such as controller, cache and disk heads in a disk array), resulting in an unpredictable request completion. Once the execution of a request is delayed, the request service time [3] will increase and the subsequent requests in flight will very likely suffer from queuing and waiting. Thus, a level of IO concurrency that specifies an upper bound on the number of requests in flight can correlate the degree of latency SLO violation risk with the metric of throughput. This essentially determines a trade-off between latency SLO enforcement and the enforcement cost in terms of reduced throughput. The key is to coordinate the level of IO concurrency adapting to the status of the X^{th} percentile latency SLO enforcement.

[3] Request service time refers to the time interval between the dispatching time of a request and its completion time.

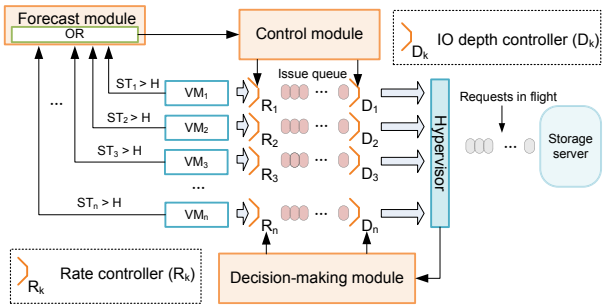


Figure 2: The architecture of PSLO.

To achieve its design goal, PSLO consists of three key functional modules, the *forecast module*, *control module* and *decision-making module*. As shown in Figure 2, the forecast module and the control module work together to minimize the latency SLO violations. The control module exports the interfaces for IO rate control and VM issue queue length control. The decision-making module uses these interfaces to coordinate the IO concurrency level and the arrival rate for each VM so as to gain a good trade-off between the X^{th} percentile latency enforcement and the throughput reduction.

3.2 Forecast Module

The forecast module is designed for the prediction of high percentile (e.g., 99.9^{th}) latency SLO violation, which must be able to forecast SLO violation timely and accurately. To achieve this, we take full advantage of the latency violation forecasts from each individual VMs based on an analysis suggesting that the latency SLO violated requests among consolidated VMs have a certain time-correlation, or temporal locality, as explained below.

For the consolidated VMs built on a shared storage infrastructure, the IO congestion taking place at the network transmission stage, the IO stack of the storage server or the underlying storage devices will very likely lengthen the request service time. Once the execution of a request from a VM is delayed, the subsequent requests issued by other concurrently running VMs will be blocked. This may result in possible SLO violations. As a result, there exits a certain time-correlation among requests that violate the latency SLO issued by different consolidated VMs, and the request service time is a critical indication for SLO violation. This implies that as soon as any VM detects a sign of imminent SLO violation of one of its requests, other consolidated VMs should be warned of potential SLO violations of their respective requests. Specifically, as shown in Figure 2, the forecast module collects the forecasts from all the consolidated VMs, where a state bit associated with each VM is set to ”0” if no

latency SLO violation is predicted in it, and "1" otherwise. The forecast module ensures the earliest forecast to take effect by ORing all the state bits for consolidated VMs. The forecast model can be represented as:

$$\zeta = 1 - \prod_{k=1}^{n}(1 - p_k) \quad (1)$$

where ζ represents the probability of issuing SLO violation forecasts for n consolidated VMs (denoted by $VM_k, 1 \leq k \leq n$), and p_k represents the probability of detecting the sign of an imminent SLO violation for VM_k. The forecast module determines the value of p_k by assessing the variability of request service time for each VM in terms of *normalized service time*. The normalized service time of a request issued by a VM is defined to be the ratio of the request service time to the average service time across all the requests issued by the VM over the last fixed-length interval (e.g., 20ms). Once the newly measured normalized service time of any request issued by VM_k (i.e., ST_k) exceeds a threshold (i.e., H), we consider that there is a risk of imminent latency SLO violation for consolidated VMs, which can be represented by the following formula:

$$p_k = Pr(ST_k \geq H) \quad (2)$$

Based on Formulas 1 and 2, we can conclude that the normalized service time threshold (H) can be used to adjust the probability of issuing SLO violation forecasts (ζ). In other words, the larger the value of H is, the smaller the value of ζ is. And a smaller ζ indicates a lower probability of limiting IO concurrency for consolidated VMs due to fewer SLO violation forecasts. This in turn results in a higher throughput, however, at the possible cost of more request latency SLO violations.

To experimentally verify the above analysis, we conduct an experiment by running five consolidated VMs that are created by Xen [3] and accessing a 16-disk RAID-0 disk group. There are 5 workloads deployed on these five VMs, respectively, including four real production traces, MSN, Excange, TPC-E and WebSearch [2, 18], and a synthetic workload trace of a large file copy, File copy. We set the normalized service time threshold at 2, 4, 8, 16, 32 and 64 respectively for all the consolidated VMs. The value of ζ can be observed to decrease monotonously from 60.13% to 7.19% as the value of H increases from 2 to 64. In the meanwhile, the maximum throughput reduction, defined to be the ratio of the actual throughput to the baseline throughput obtained under the default Xen host OS, among all the consolidated VMs is observed to decrease from 21.92% to 4.45% while the latency SLO violation ratios for these VMs increase monotonously. Thus, it is necessary to obtain appropriate normalized service time thresholds for consolidated VMs based on their X^{th} percentile latency and throughput SLOs.

However, it is unavoidable for the model to have some false positive rate in latency SLO violation forecast. Moreover, it is very difficult to find the best combination of normalized service time thresholds to both meet the X^{th} percentile latency SLOs and have the lowest cost of throughput reduction for all the consolidated VMs by exhaustively searching for all the possible combinations. The search scope can be increased exponentially with the number of consolidated VMs, resulting in an unacceptable overhead. So we allow the forecast model to have a certain false positive rate to adequately prevent latency SLO violations. Thus, H can be empirically set at an appropriately small value to construct a forecast model adequately sensitive to the variability of request service time (e.g., set at 2 for HDD based storage and 1.5 for SSD based storage for our evaluation in Section 5). To avoid the consequent unnecessary throughput reduction due to an H setting that results in too high a probability of limiting IO concurrency for consolidated VMs, PSLO uses the decision-making module to coordinate the threshold that limits the IO concurrency level for each VM so that it adapts to the actual X^{th} percentile latency and throughput SLO compliance, as explained in Section 3.4.

3.3 Control Module

The control module is designed to enforce the X^{th} percentile latency SLO and throughput SLO for each VM by adjusting the level of IO concurrency and controlling the arrival rate at each VM's issue queue. As shown in Figure 2, the IO concurrency of a VM (VM_k) is decided by the upper bound on the number of read or write requests in the issue queue (D_k) and the arrival rate at VM_k is limited by the arrival rate limit (R_k). So a rate controller and an IO depth controller are established for each VM to work together based on these two parameters.

3.3.1 Rate Controller

The rate controller, equipped for each VM_k ($1 \leq k \leq n$ with n being the number of VMs), is responsible for coordinating the arrival rate at the issue queue of VM_k by a time-stamp based IO scheduling that underlies many previous studies [4, 7, 12]. The time-stamp T_k^q, assigned to a request q from VM_k, is the larger of the sum of the previous time-stamp T_k^{q-1} and $1/\hat{R}_k$ and the arrival time, as:

$$T_k^q = max(T_k^{q-1} + 1/\hat{R}_k, \quad Arrival\ time) \quad (3)$$

The rate controller for VM_k controls the interval between two consecutively scheduled requests by adjusting $\hat{R}_k$. Specifically, we use integral control, adopted from the control theory [8, 14], to make the measured arrival rate R_k^m converge to the pre-specified target R_k by coordinating $\hat{R}_k$ to adapt to the deviation between R_k^m and R_k.

3.3.2 IO Depth Controller

The IO depth controller, equipped for each VM_k, is employed to limit the length of VM_k's issue queue according to the setting of D_k. Once triggered by the forecast module and given the arrival rate of R_k at the issue queue of VM_k, the

IO depth controller will exact further admission control as follows. Read or write requests are admitted to into the VM issue queue if the number of read or write requests residing in the issue queue is now less than the upper bound of D_k. It is noted that the IO depth controllers for all the consolidated VMs are triggered simultaneously by the forecast module and remain active for a time interval τ. Thus, the risk of latency SLO violation can be reduced due to the reduced level of IO concurrency. The control module exports a control interface of IO concurrency level for all the consolidated VMs (i.e., $VM_k, 1 \leq k \leq n$), which can be called by inputting the parameter set of $[D_1, D_2, ..., D_n]$.

3.4 Decision-Making Module

The decision-making module is designed to strike a proper trade-off between the level of IO concurrency and latency SLO violation by specifying the parameters R_k and D_k for each VM VM_k. Based on the statistics on IO latency and completed requests collected by the IO scheduler of the hypervisor, the decision-making module can obtain the actual throughput and the accumulated number of SLO violated requests for each VM. To differentiate the X^{th} percentile latency SLO for each VM, PSLO first formulates the boundary curve of the X^{th} percentile SLO violation for each VM. As elaborated next in subsection 3.4.1, this curve marks the boundary below which the number of SLO violated requests for a given number of completed requests is considered acceptable without sacrificing too much IO concurrency. Thus, if the curve of the actual accumulated number of SLO violated requests lies below this boundary curve for a VM during a time interval, then it provides an opportunity to increase the throughput of the VM by increasing its IO concurrency during that interval.

3.4.1 Upper Bound on the X^{th} Percentile Latency SLO Violations

The X^{th} percentile SLO is a statistical metric obtained at the end of the execution of a workload. For example, if a workload with a million requests runs for 5000 seconds and its 99.9^{th} percentile SLO is 200ms, we cannot affirm that the 99.9^{th} percentile SLO of the workload is met by knowing only the number of violations during the first 4000 seconds since there could be more than 1000 SLO violations (request latency exceeding 200ms) taking place during the final 1000 seconds, resulting in a 99.9^{th} percentile latency higher than the SLO of 200ms. So it is difficult to accurately bound the X^{th} percentile latency according to the pre-specified SLO during the run of the workload. The key is to let SLO violations spread evenly across the entire length of the workload execution according to the percentile specified by the latency SLO.

The boundary curve of the X^{th} percentile SLO violation for each VM is formulated based on the assumption that the number of SLO violations of a VM during a time interval is roughly proportional to the number of completed requests issued by the VM during that interval. In other words, the SLO violations of a VM are fairly evenly distributed among the time intervals where each has the same number of completed requests from the VM. Thus, if the latency SLO of a VM is specified at the 99.9^{th} percentile and 1000 requests are completed during an observed period, then the upper bound on the number of SLO violations for that period is 1. We obtain the boundary curve of the 99.9^{th} percentile SLO violation as a function of the accumulated number of completed requests (denoted by boundary curve), as shown in Figure 3, which can be represented as:

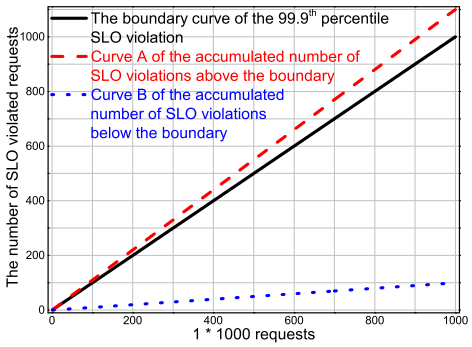


Figure 3: The illustration of the correlation among the three curves: the boundary curve of the 99.9^{th} percentile SLO violation, curve A of the accumulated number of SLO violations above the boundary, and curve B of the accumulated number of SLO violations below the boundary. Curve A indicates that every point on the curve violates in the 99.9^{th} percentile SLO. On the contrary, curve B suggests that there is ample room for throughput optimization. The ideal case happens when the curve of the number of SLO violations coincides with the boundary curve.

$$G(X_k, t) = \sum_{i=1}^{t} (1 - X_k) * N \quad (4)$$

where X_k is the percentile at which the latency SLO is defined for VM_k, t is the index of the time interval during which N requests issued by VM_k are completed and $G(X_k, t)$ is the upper bound on the accumulated number of SLO violations at the end of the t^{th} time interval. The value of t can be bounded by the number of intervals for the application execution or the maximum statistical cycle defined by the cloud service provider. If the actual accumulated number of SLO violations for VM_k at the end of the t^{th} time interval (denoted by $M_k(t)$) is larger than $G(X_k, t)$, we affirm that the X_k^{th} percentile latency SLO is violated during the period from the first time interval to the end of the t^{th} time interval. Otherwise, if $M_k(t)$ is smaller than $G(X_k, t)$, as curve B shown in Figure 3, we can increase the level of IO concurrency for a higher throughput at the cost of an increased $M_k(t)$ whose value is allowed to reach $G(X_k, t)$.

3.4.2 IO Concurrency & Arrival Rate Control

The decision-making module coordinates the level of IO concurrency and the limit on arrival rate for VM_k by adjusting the parameters R_k (arrival rate limit) and D_k (upper bound on the length of the request issue queue).

Based on the detailed evaluation of the impact of the D_k parameter on the latency SLO violation with real-world production traces [2, 18] (in Section 5.2), we find a positive correlation between D_k and the actual accumulated number of SLO violations for VM_k. Thus, we can increase D_k to improve throughput, subject to the following condition:

$$M_k(t) \leq G(X_k, t) \tag{5}$$

Formula 5 stipulates that the X^{th} percentile latency SLO must be complied with. The goal of IO concurrency control is to minimize the difference between $G(X_k, t)$ and $M_k(t)$, as expressed in Formula 6, when $G(X_k, t)$ is larger than $M_k(t)$. In this way, SLO violations can be evenly distributed among the time intervals as much as possible and IO concurrency can be maximized while still complying with the X^{th} percentile latency SLO.

$$e(k, t) = G(X_k, t) - M_k(t) \tag{6}$$

Note that the larger the value of $e(k, t)$ is, the fewer the latency SLO violations are. To convert $e(k, t)$ into a more intuitive metric and thus easier to control, we use $D_k(t)$ to denote the value of D_k set at the t^{th} interval for VM_k and divide $e(k, t)$ by $D_k(t)$ as follows:

$$E(k, t) = \frac{e(k, t)}{D_k(t)} \tag{7}$$

where $E(k, t)$ represents the difference between the actual latency SLO violations and the upper bound on the latency SLO violations, which makes it independent of D_k that is associated with workload IO characteristics and latency SLO.

In addition, the distinct IO characteristics and different requirements on the percentile latency SLO of user applications can lead to different latency SLO violations and unfair allocation of throughput among consolidated VMs under the different X^{th} percentile latency SLO constraints.

To address the above problem, each VM is reserved customized throughput as its throughput SLO target, and the decision-making module dynamically adjusts the arrival rates for the n consolidated VMs through the control interface of $[R_1, R_2, ..., R_n]$ exported by the control module. For a time period divided into t intervals, where the same number N requests are completed in each interval i whose actual length is denoted by $\lambda(i), 1 \leq i \leq t$, the average throughput $\overline{Th_k(t)}$ over this time period is defined as:

$$\overline{Th_k(t)} = \frac{t * N}{\sum_{i=1}^{t} \lambda(i)} \tag{8}$$

Thus, we can obtain the following recursive formula based on Formula 8:

$$R_k(t+1) = Max(\frac{Th_k^{SLO}}{\overline{Th_k(t)}} * R_k(t), Th_k^{SLO}) \tag{9}$$

where Th_k^{SLO} represents the throughput SLO target of VM_k. The constraints on the arrival rates based on Formula 9 help enforce the fairness of throughput allocation among consolidated VMs. However, the user-customized throughput SLO for a VM is very likely lower than the potential value the system is able to provide. Thus, another task of the decision-making module is to explore throughput targets for all the n consolidated VMs ($VM_k, 1 \leq k \leq n$) that are higher than their user-customized SLOs to exploit the spare resources on the condition that the X^{th} percentile latency SLO for each VM is respected.

Let $Th(k, i)$ denote the actual throughput of VM_k averaged over the i^{th} interval and assume that $Th(k, i)$ correlates positively with the number of latency SLO violations in the i^{th} interval. This is reasonable since a higher throughput of VM_k usually requires a higher level of IO concurrency, i.e., a larger value of D_k that is experimentally proven to correlate positively with the number of latency SLO violations in Section 5.2. Thus, we adjust the throughput target of VM_k by obtaining the values of $A(k, t)$ and $P(t)$ represented by the following formulas.

$$A(k, t) = Max(\sigma * \frac{e(k, t)}{G(X_k, t)}, 0) \tag{10}$$

$$P(t) = Min_{1 \leq k \leq n} A(k, t) + 1 \tag{11}$$

$A(k, t)$ depends on the ratio of $e(k, t)$ to $G(X_k, t)$, which reflects the extent to which the actual number of latency SLO violations deviates below the upper bound regulated by the X^{th} percentile latency SLO target. The larger the values of $A(k, t)$ are for all the consolidated VMs, the higher the throughput SLO target can be specified for each VM. σ is a parameter that reflects the trade-off between the cost of throughput reduction and latency SLO violations. A larger σ is more conducive to high throughput while a smaller σ is more beneficial to the X^{th} percentile latency SLO enforcement. By adding 1 to the minimum value of $A(k, t)$ among all the n consolidated VMs, $P(t)$ serves as a proportional coefficient to determine how much to increase the throughput targets of all the consolidated VMs in the $(t+1)^{th}$ interval, denoted by $Th_k^G(t+1), 1 \leq k \leq n$, and expressed as follows.

$$Th_k^G(t+1) = \begin{cases} P(t) * Th_k^G(t) & if\ \forall j, Th(j, t) \geq Th_k^G(t), \\ Max(\eta * Th_k^G(t), Th_k^{SLO}) & otherwise. \end{cases} \tag{12}$$

$$R_k(t+1) = Max(\frac{Th_k^G(t)}{\overline{Th_k(t)}} * R_k(t), Th_k^G(t)) \tag{13}$$

$$D_k(t+1) = \begin{cases} 1 & if\ \exists j, E(j, t) < 0, \\ Max(D_k(t) - 1, 1) & else\ if\ Th_k(t) > Th_k^G(t), \\ D_k(t) + 1 & else\ if\ E(k, t) \geq U, \\ D_k(t) & otherwise. \end{cases} \tag{14}$$

As expressed in Formula 12, if the actual throughput of each VM is larger than or equal to its throughput target in the

t^{th} interval, then the throughput target of VM_k for the next interval is set at the value of $P(t) * Th_k^G(t)$. Otherwise, all the throughput targets for the next interval for consolidated VMs will be the product of η (set at 0.95) and $Th_k^G(t)$. The initial value of $Th_k^G(t)$ is set at the value of the throughput SLO target of VM_k, Th_k^{SLO}. The corresponding value of the arrival rate limit $R_k(t+1)$ is expressed by Formula 13. Similarly, the value of the upper bound on the request issue queue depth, $D_k(t+1)$, is given by Formula 14, with its initial value (i.e., $D_k(1)$) being 1. The constant U in Formula 14 is the threshold for $E(k,t)$ to increase $D_k(t+1)$ and set at 5 by default. Specifically, if there exists $E(j,t)$ $(1 \leq j \leq n)$ that is smaller than zero, meaning that the X^{th} percentile latency SLOs of one or more VMs are violated, we set $D_k(t+1)$ at the smallest value (i.e., 1). Otherwise, if $E(k,t)$ is larger than or equal to U, $D_k(t+1)$ is set to be $D_k(t)$ plus one. If the measured throughput $Th_k(t)$ is larger than the throughput target $Th_k^G(t)$, indicating that the level of IO concurrency is higher than necessary, then $D_k(t+1)$ will be decreased by 1 provided that its value is greater than 1 (the smallest value). We keep $D_k(t+1)$ invariable for all the other cases.

3.4.3 Feasibility of Simultaneous Enforcement of the X^{th} Percentile Latency and Throughput SLOs

Based on the experimental analysis in Section 2.1 and the IO isolation among consolidated VMs by VM-oriented arrival rate control, it is possible to assess the feasibility of simultaneously enforcing the X^{th} percentile latency and throughput SLOs for a VM. Specifically, we first obtain the performance & concurrency curve by sampling the X^{th} percentile latency and throughput for the VM under different levels of IO concurrency. And then, we determine if there exists a sampling point with the measured X^{th} percentile latency that is lower than the X^{th} percentile latency SLO and the measured throughput that is higher than the throughput SLO. If yes, it is considered feasible to simultaneously enforce these two SLOs for the VM. Otherwise, these two SLOs cannot be simultaneously met. In fact, we believe that it is possible to reduce the sampling points for the performance & concurrency curve so as to decrease the cost of SLO feasibility estimation. This, however, is beyond the scope of this paper and thus a topic of our future research.

4. Implementation Issues

In this section, we present the implementation details of PSLO's critical functionalities, latency SLO violation forecast and end-to-end VM-oriented control.

4.1 Forecast Latency SLO Violation

The forecast module monitors the service time of each request issued by each individual VM in a VM-oriented end-to-end fashion. The request service time is measured between the time when a request is dispatched to the underlying storage device and the time when the request is completed, obtained based on the Linux block layer statistics and collected by the IO statistics sub-module. In all the following experiments, we obtain the value of normalized service time by dividing the measured request service time by the average service time over the latest 20ms time interval for each VM. Moreover, whenever the currently measured normalized service time of a VM exceeds a pre-determined threshold H (empirically set at 2 for our evaluation), the forecast module will issue a latency SLO violation forecast. In this way, the forecast is proven to be adequately sensitive to the variability of request service time that is key to predicting the latency SLO violation at a high percentile (e.g., 99.9^{th}).

4.2 End-to-End VM-oriented Control

In the current version of Xen [3], an IO request issued by a VM is first sent to the frontend driver while an event is simultaneously sent by blkfront to notify the backend driver for the corresponding virtual disk of the VM to redirect the request to the block layer. The request is then handled by the hypervisor IO scheduler where the control module of PSLO resides. Each request will be tagged with the identification of the issuing VM in the hypervisor, which is used by the control module to implement VM-oriented end-to-end IO control. Specifically, once the first request of a VM arrives, the control module will recognize the newly activated VM and establish a rate controller and an IO depth controller dedicated to that VM.

The rate controller carries out IO throttling according to the arrival rate target determined by the decision-making module and adopts the integral control [8, 14] to reduce the deviation of the actual IO rate from the target value. The IO depth controller for a given VM VM_k limits its issue queue length according to D_k, the optimized upper bound on the number of the read or write requests in the issue queue of the VM that is determined by the decision-making module. Thus, the value of $E(k,t)$ (Formula 7) that determines the value of $D_k(t+1)$ is key to striking a good trade-off between the level of IO concurrency and the latency SLO violations. In practice, we collect IO statistics for each individual VM in every time interval with 1000 completed requests.

5. Performance Evaluation

In this section, we present the evaluation results of a PSLO prototype implemented in a real consolidated VM environment. All the following evaluation experiments are conducted on a dedicated rack of servers. The storage server is a PowerLeader PR2760T machine configured with 2 Intel Xeon E5620 quad-core processors, 12GB of RAM, a Mellanox MT26428 ConnectX VPI Infiniband NIC and a storage array. The storage volume is hosted on a 16-disk RAID-0 disk group on the array, consolidating all the virtual disks, where each VM is associated with a virtual disk of 80GB. The application server is a PowerLeader PR2760T machine

Policy	Throughput SLO compliance	The X^{th} percentile latency SLO compliance	Throughput allocation fairness	Throughput optimization	Latency optimization
PSLO(TS+L)	**Y**	**Y**	**Y**	**N**	**Y**
PSLO(LS+T)	**Y**	**Y**	**Y**	**Y**	**N**
PSLO(L)	**N**	**Y**	**N**	**N**	**Y**
PSLO(LS+U)	**N**	**Y**	**N**	**Y**	**N**
PSLO(T)	**Y**	**N**	**Y**	**N**	**N**
No PSLO	**N**	**N**	**N**	**N**	**N**

Table 1: Comparison of different optimization policies under PSLO.

that uses CentOS 6.3 with 64-bit Linux kernel 3.9.9 for the host OS and supports VM consolidation based on the hypervisor of Xen 4.2 [3].

We evaluate PSLO with a collection of real production traces [2, 18] (i.e., MSN, Excange, TPC-E and WebSearch) and a synthetic workload trace of a large file copy (i.e., File copy). There are five consolidated VMs running these five traces respectively, concurrently accessing the storage volume hosted on the large-scale RAID-0 disk array. The VM running the File copy workload only requires a throughput SLO of 150 IOPS for all the following experiments.

5.1 Evaluation Objectives and Policies

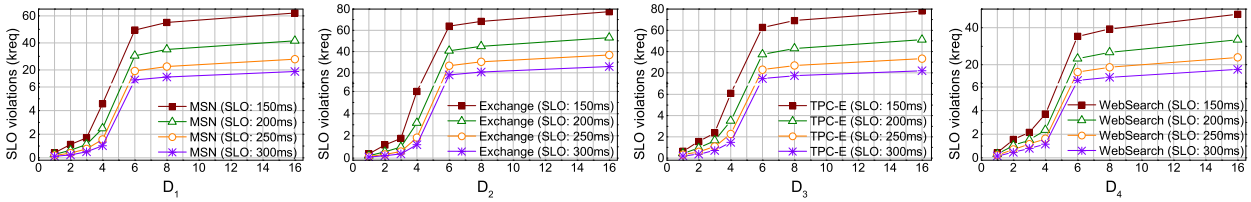
Objectives: In the following experiments, we first investigate the impact of D_k, i.e., the upper bound on the number of read or write requests in the issue queue of VM_k, on the X^{th} percentile latency and throughput of VM_k. D_k is the key parameter for PSLO to strike an appropriate trade-off between the X^{th} percentile latency and throughput by coordinating the level of IO concurrency. This is enabled by the forecast module when the normalized service time is larger than a pre-specified threshold. The arrival rate limit for each VM is set at 150 IOPS. We examine how latency SLO violations, throughput and latency distribution change with D_k, thus illustrating the impact of the IO concurrency level on latency and throughput.

To comprehensively evaluate PSLO, we design two application scenarios to verify the effectiveness of PSLO in optimizing the X^{th} percentile latency or throughput for consolidated VMs under different policies. We let the VMs deployed with totally different applications be consolidated on the same server sharing the 16-disk RAID0 disk array. This configuration allows the interplay of IO flows with distinct IO characteristics to aggravate the IO interference and result in very different latency SLO violations among the consolidated VMs. As a result, PSLO can be stress-tested in a realistic scenario that requires strong adaptivity to adequately address the interplay of highly variable IO characteristics when enforcing the X^{th} percentile latency and throughput SLOs.

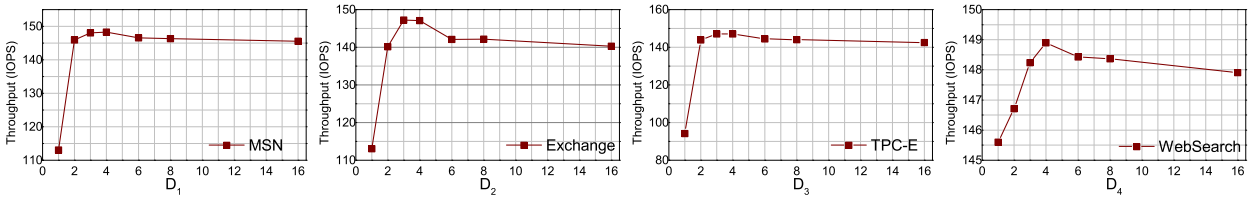
The first scenario consists of four applications (i.e., MSN, Excange, TPC-E and WebSearch) that are deployed on 100 VMs. Each application requires an IO latency SLO of 300ms with at least a 90% compliance ratio and a throughput SLO of 15000 IOPS. Thus, based on the analysis presented in Section 2.1, we should set the 99.9^{th} percentile latency SLO at 300ms and the throughput of 150 IOPS for each VM. *With this scenario, we assess the ability of PSLO in optimizing throughput under the X^{th} percentile latency SLO constraint as well as enforcing the fairness of throughput allocation among consolidated VMs*.

The second scenario has the same four applications deployed on 25 VMs, 15 VMs, 100 VMs and 100 VMs respectively. These four applications need the latency SLOs of 100 ms, 100 ms, 200 ms and 200 ms respectively with at least a 90% compliance ratio and the throughput SLOs of 3750 IOPS, 2250 IOPS, 10000 IOPS and 10000 IOPS respectively. Thus, each of the VMs running MSN requires the 99.6^{th} percentile latency SLO of 100 ms and the throughput SLO of 150 IOPS. Each of the VMs with Exchange needs a latency SLO of 100 ms at the 99.3^{th} percentile and a throughput SLO of 150 IOPS. And each VM of the VMs for TPC-E and WebSearch requires the 99.9^{th} percentile latency SLO of 200 ms and a throughput SLO of 100 IOPS. *In this scenario, we evaluate the robustness and effectiveness of PSLO in optimizing the X^{th} percentile latency or throughput under different SLO constraints*.

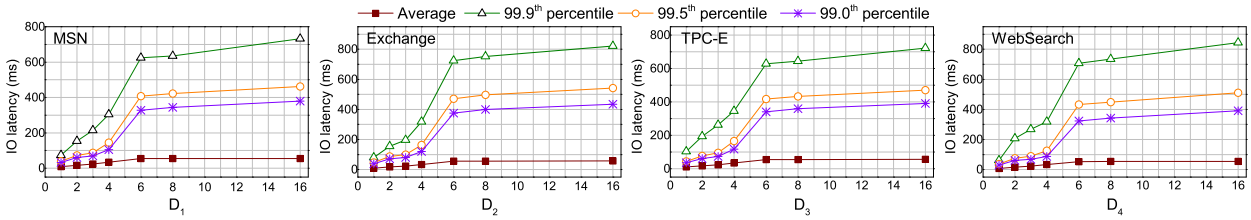
Polices: PSLO can enforce the X^{th} percentile latency and throughput SLOs for consolidated VMs according to different policies. As listed in Table 1, under the *PSLO(TS+L)* policy, PSLO optimizes the X^{th} percentile **l**atency under **t**hroughput **S**LO constraints and aims to achieve a low value of the X^{th} percentile latency for each VM with the best effort on the condition that the throughput SLO of each VM is complied with. With the *PSLO(LS+T)* policy, PSLO optimizes **t**hroughput under the X^{th} percentile **l**atency **S**LO constraint. If we only focus on the performance of the X^{th} percentile latency, PSLO will keep the lowest level of IO concurrency (i.e., set the value of $D_k(t)$ at 1) for each VM, which is the *PSLO(L)* policy. In addition, if the arrival rate limit $R_k(t)$ is set to be infinity, PSLO adopts the *PSLO(LS+U)* policy that optimizes the throughput for each VM under the X^{th} percentile **l**atency **S**LO constraint, but likely leading to throughput allocation **u**nfairness. The *PSLO(T)* policy means that PSLO only enforces throughput SLO for each VM by controlling its arrival rate with the integral control model [8, 14]. Finally, the *No PSLO* policy is the earliest deadline first (EDF) scheduling policy implemented on the side of the storage server and the deadline for each request is configured with its arrival time so as to realize



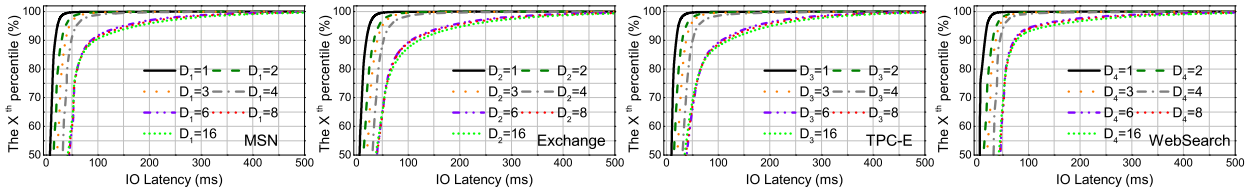
(a) Number of SLO violations as a function of $D_k (1 \leq k \leq 4)$ for different SLOs



(b) Throughput as a function of $D_k (1 \leq k \leq 4)$



(c) Average latency and latencies at the 99.9^{th}, 99.5^{th} and 99^{th} percentiles as a function of $D_k (1 \leq k \leq 4)$



(d) Latency distribution as a function of $D_k (1 \leq k \leq 4)$

Figure 4: Impact of D_k on the X^{th} percentile latency and throughput of VM_k, $1 \leq k \leq 4$.

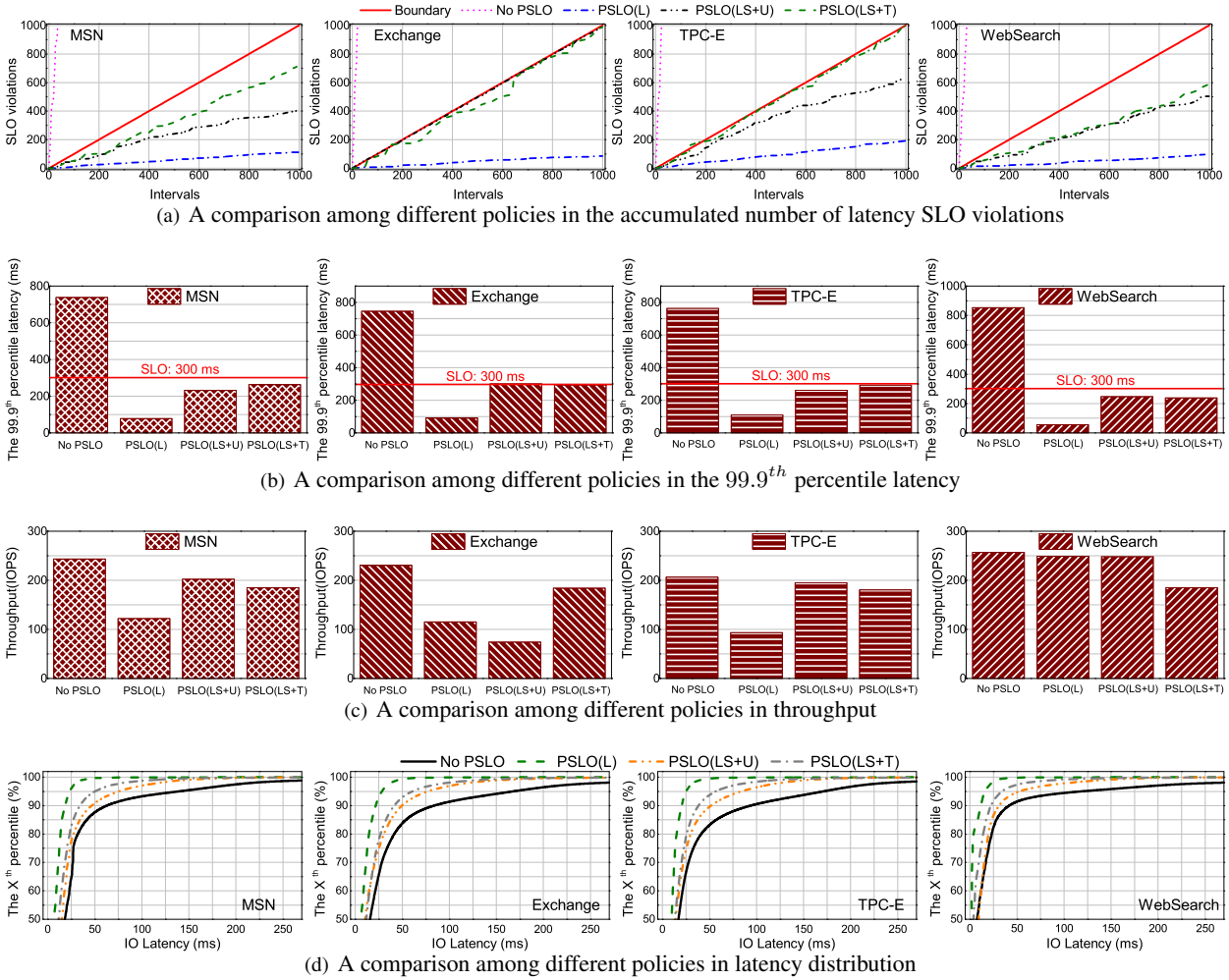
the first-come, first-served (FCFS) order without additional IO wait. The VM running the File copy workload is always controlled under the PSLO(T) policy in all the following experiments.

Each measurement in the following experiments involves 1 million requests issued by each VM and the measurement period is divided into 1000 intervals with each having 1000 completed requests. Unless otherwise specified, the throughput for each VM is obtained by averaging over the measurement period according to Formula 8.

5.2 IO Concurrency Control

PSLO controls the level of IO concurrency for consolidated VMs by adjusting the upper bound on the number of read or write requests in each VM issue queue, i.e., D_k for VM_k, also referred to as upper bound on issue queue length. To comprehensively investigate the impact of D_k on IO latency and throughput SLO enforcement, we measure the number of latency SLO violations, throughput, average latency, latencies at different percentiles and latency distributions as a function of D_k whose value ranges from 1 to 16.

As shown in Figures 4(a) and 4(c), we observe that D_k is positively correlated with both the number of latency SLO violations under different SLOs and the latency measures in average and percentiles. The higher the D_k value, thus the level of IO concurrency, the greater the number of latency SLO violations. Specifically, there are three successive phases of D_k, namely, (1) $1 \leq D_k \leq 4$, (2) $4 < D_k \leq 6$ and (3) $D_k > 6$, that exhibit distinctive impacts of D_k on latency SLO violations and latency measures. For Phase (1), latency SLO violations and latency measures at different percentiles for all the VMs increase slowly with D_k. Phase (2) is characterized by a steep and rapid increase in both latency SLO violations and latency measures at different percentiles with D_k, where a 10-fold increase in the number of latency violations is observed for all the VMs. This is followed immediately by the continued but very smooth and slow growth trends of SLO violations and latency measures at different percentiles in Phase (3). Actually, with the increment of D_k in Phase (3), the effectiveness of performance isolation enforced by D_k is gradually weakened and eventually lost, thus aggravating IO contention among consoli-



(a) A comparison among different policies in the accumulated number of latency SLO violations

(b) A comparison among different policies in the 99.9^{th} percentile latency

(c) A comparison among different policies in throughput

(d) A comparison among different policies in latency distribution

Figure 5: A comparative evaluation of different policies in terms of the number of latency SLO violations, the 99.9^{th} percentile latency, throughput and latency distribution for VMs with the 99.9^{th} percentile latency SLO of 300ms.

dated VMs as indicated by the increased number of latency SLO violations (Figure 4(a)) and decreased throughput (Figure 4(b)) in Phase (3).

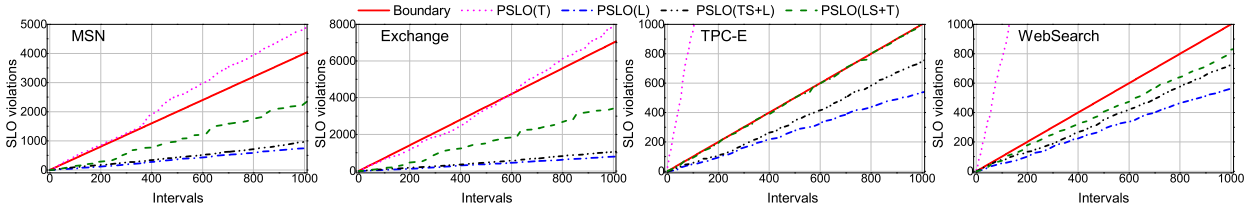
In addition, as shown in Figure 4(d), a smaller value of D_k means a lower latency at the X^{th} percentile except for Phase (3). In Phase (3) the effectiveness of D_k in affecting the latency SLO enforcement is basically lost, thus the latency distributions in this phase (i.e., $D_k = 6, 8, 16$) almost perfectly coincide with one another. In contrast, as shown in Figure 4(b), the throughput for all the VMs can be observed to increase sharply and reach the peak value in Phase (1) immediately before the beginning of Phase (2).

As a result, it is feasible for PSLO to seek a good trade-off between latency at different X^{th} percentiles and throughput according to different optimization requirements by controlling the level of IO concurrency for each VM.

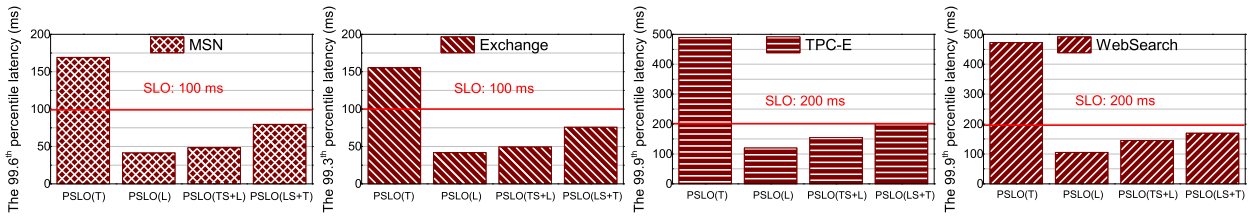
5.3 Throughput Optimization and Fair Allocation

In the first scenario described in Section 5.1, the consolidated VMs running the workloads of MSN, Exchange, TPC-E and WebSearch require a latency SLO of 300 ms at the 99.9^{th} percentile and a throughput SLO of 150 IOPS. We repeat the experiment under different polices, including No PSLO, PSLO(L), PSLO(LS+U) and PSLO(LS+T). As shown in Figure 5, PSLO(L) performs the best in terms of latency SLO enforcement without considering the cost of throughput reduction while No PSLO, which serves as the baseline of latency SLO enforcement, performs the worst among all the policies. In fact, with the exception of the throughput measure, PSLO(L) and No PSLO perform the best and worst among all policies in the measures of SLO violations (Figure 5(a)), the 99.9^{th} percentile latency (Figure 5(b)) and the X^{th} percentile latency distribution (Figure 5(d)). However, as shown in Figure 5(c), No PSLO outperforms all other policies in the throughput measure.

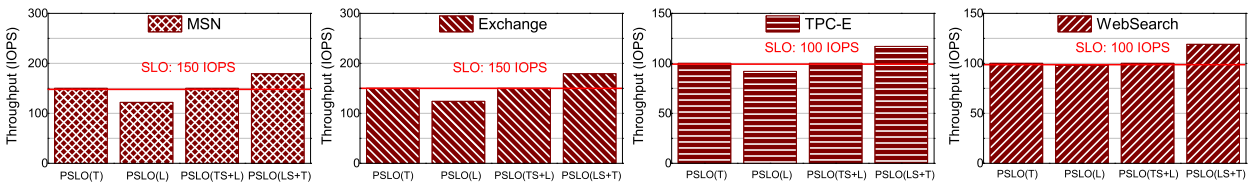
The PSLO(LS+U) policy enforces the X^{th} percentile latency SLO and sets the arrival rate limit at infinity for each consolidated VM. As a result, PSLO(LS+U) only focuses on the coordination of the IO concurrency level without considering the fairness of throughput allocation among consolidated VMs. However, the PSLO(LS+T) policy si-



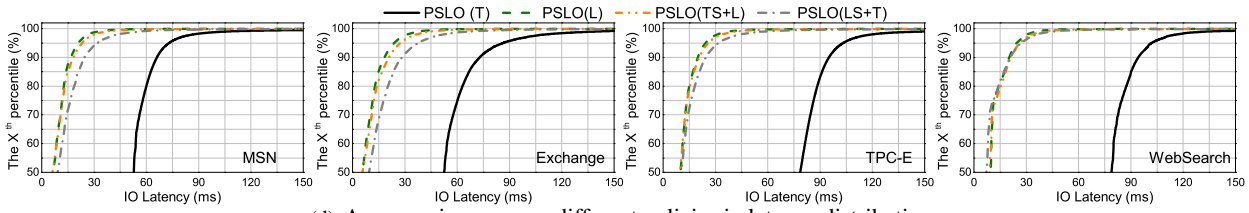
(a) A comparison among different policies in latency SLO violations



(b) A comparison among different policies in the Xth percentile latency



(c) A comparison among different policies in throughput



(d) A comparison among different policies in latency distribution

Figure 6: A comparative evaluation of different policies in terms of the number of latency SLO violations, the X^{th} percentile latency, throughput and latency distribution for VMs with the X^{th} percentile latency and throughput SLOs.

multaneously bounds the issue queue length and the arrival rate of each VM according to Formulas 13 and 14. Thus, PSLO(LS+T) optimizes the throughput of each VM under the X^{th} percentile latency SLO constraints of all the consolidated VMs while simultaneously enforces the fairness of throughput allocation. As shown in Figure 5, although PSLO(LS+U) outperforms PSLO(L) in total throughput (sum of all four throughputs) by 24.2% and achieves a maximum value of 299.8 ms for the 99.9^{th} percentile latency, which complies almost perfectly with the latency SLO of 300 ms, under the Exchange workload its throughput performance of 74 IOPS falls far below the throughput SLO of 150 IOPS. In contrast, the PSLO(LS+T) policy not only achieves very similar advantages over PSLO(L), with a margin of 26.8% in total throughput and a maximum value of 294.4 ms for the 99.9^{th} percentile latency, its throughput measure for each VM is higher than the SLO of 150 IOPS. In other words, the X^{th} percentile latency SLO and throughput SLO for each VM can be met under the PSLO(LS+T) policy. In addition, the actual throughput of the VM running the File copy workload under the PSLO(LS+T) policy is 150.2 IOPS, which perfectly complies with the SLO of 150 IOPS.

5.4 Optimization under Different Constraints

For the second scenario described in Section 5.1, the consolidated VMs differ in the X^{th} percentile latency and throughput SLOs. Under such complicated SLO requirements, we assess the robustness and effectiveness of PSLO in enforcing these SLOs under different constrains and policies. Recall that the PSLO(LS+T) policy optimizes the throughput and enforces the fairness of throughput allocation among consolidated VMs under the constraint that the X^{th} percentile latency SLO of each VM be met. In this optimization, the best case would be when the X^{th} percentile latencies of one or more VM are equal to their latency SLOs respectively, while at the same time the throughput of each VM is increased proportionally to its throughput SLO with the best effort. In contrast, the PSLO(TS+L) policy optimizes IO latency with the best effort under the constraint that the throughput SLO of each VM be complied with. The best case of this optimization would be when the level of IO concurrency is controlled

at the lowest level for each VM issue queue while still meeting the throughput SLO. The PSLO(T) and PSLO(L) policies are used as the baseline policies, since PSLO(T) only enforces throughput SLO for each VM while PSLO(L) enforces the X^{th} percentile latency SLOs of consolidated VMs without considering the cost of throughput reduction.

As shown in Figure 6(a), the curve of latency SLO violations under PSLO(T) lies largely beyond the boundary curve for all the consolidated VMs, meaning that the X^{th} percentile latency SLOs are violated for all the VMs due to lack of latency enforcement under PSLO(T). The PSLO(L) curve lies below the boundary curve and those of all the other policies. This illustrates that PSLO(L) ensures the lowest X^{th} percentile latency for each VM among all the policies, which is verified explicitly in Figure 6(b). However, the measured throughput of the VMs running the workloads of MSN, Exchange and TPC-E under PSLO(L) is lower than their throughput SLOs. In contrast, PSLO(TS+L) exceeds PSLO(L) in the X^{th} percentile latency measure for each VM by 25.4% on average across all the consolidated VMs. In the meantime, as shown in 6(c), the throughput SLO of each VM is accurately complied with. This is perhaps one of PSLO(TS+L)'s best case scenarios.

As shown in Figure 6(a), the curve of SLO violations for the VM running TPC-E almost perfectly coincides with the boundary curve, indicating that the X^{th} percentile latency (200.4 ms) is almost exactly right on the SLO target (200 ms), as verified in 6(b). In this case, we believe that any increase in the level of IO concurrency for any consolidated VM will make the VM running the TPC-E workload violate its X^{th} percentile latency SLO. In other words, PSLO(LS+T) is virtually in its best case scenario too. Moreover, the throughput for the four VMs running the workloads of MSN, Exchange, TPC-E and WebSearch respectively is increased by 19%, 19%, 18% and 19%, almost identical proportions, over their respective throughput SLOs under PSLO(LS+T). That is, the throughput of each VM is increased basically proportionally to its throughput SLO. Thus, the fairness of throughput allocation is enforced under the X^{th} percentile latency SLO constraint.

In addition, as shown in Figure 6(d), the cumulative distribution function (CDF) curve of IO latency obtained under PSLO(T) lies below the CDF curves under other policies for all the consolidated VMs. This indicates that a higher latency at the X^{th} percentile will be obtained under PSLO(T) than those under other polices due to the lack of support for the X^{th} percentile latency SLO enforcement in PSLO(T). In contrast, the CDF curve under PSLO(L) lies above those under all the other policies except for the VM running the WebSearch workload where the CDF curves under PSLO(L), PSLO(TS+L) and PSLO(LS+T) almost coincide with one another. This indicates that PSLO(L), which maintains the lowest level of IO concurrency among all the policies, achieves a lower latency at the X^{th} percentile than those under all other policies. And the PSLO(TS+L) and PSLO(LS+T) CDF curves lie between the PSLO(T) and PSLO(L) CDF curves. These are the results of making trade-offs between the X^{th} percentile latency SLO violations and the cost of throughput reduction under the constraints of throughput SLOs (i.e., PSLO(TS+L)) or the constraints of the X^{th} percentile latency SLOs (i.e., PSLO(LS+T)) for all the consolidated VMs.

6. Conclusion

In this paper, we focus on optimizing the X^{th} percentile latency and throughput under the two-dimensional SLO constraints of the X^{th} percentile latency SLO and throughput SLO targets for consolidated VMs. To achieve this goal, we propose a framework called PSLO. The essence of PSLO is a decision-making system that, based on a given optimization policy and constraints, dynamically and adaptively applies the IO concurrency and arrival rate controls to each individual VM to attain the specific optimization goal. Specifically, PSLO can maximize the throughput for consolidated VMs under the constraints of the X^{th} percentile latency SLOs as well as enforce the fairness of throughput allocation among consolidated VMs. In addition, PSLO is also able to optimize the X^{th} percentile latency under the throughput SLO constraint. We implement a prototype of PSLO based on the Xen hypervisor and verify the robustness and effectiveness of PSLO in enforcing the X^{th} percentile latency and throughput SLOs under different optimization policies and constraints by an extensive production-trace driven evaluation. As future work, we plan to improve PSLO on supporting the X^{th} percentile latency and throughput SLOs under the energy-saving constraints.

Acknowledgments

This work is supported by the National High-tech R&D Program of China (863 Program) No.2013AA013203, No.2015AA015301, No.2015AA016701; NSFC No.61472153, No.61402189; US NSF grant: CNS-1116606 and CNS-1016609. We sincerely thank EuroSys reviewers and Dr. Marco Canini for their constructive and thoughtful comments and suggestions that helped improve the paper.

References

[1] Amazon EC2 website, 2015. URL `http://aws.amazon.com/ec2`.

[2] Storagetrace, storage performance council (umass trace repository), 2015. URL `http://traces.cs.umass.edu/index.php/Storage`.

[3] 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.

[4] J. C. R. Bennett and H. Zhang. WF2Q: Worst-case fair weighted fair queueing. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, 1996.

[5] 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.

[6] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56:74–80, 2013.

[7] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. *Journal of Internetworking Research and Experience*, 1(1):3–26, 1990.

[8] G. F. Franklin, J. D. Powell, and M. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley, 1998.

[9] P. Goyal, H. M. Vin, and H. Chen. Start-time fair queuing: A scheduling algorithm for integrated services packet switching networks. *IEEE/ACM Transactions on Networking*, 5(5):690–704, 1997.

[10] A. Gulati, A. Merchant, and P. J. Varman. pClock: an arrival curve based approach for QoS guarantees in shared storage systems. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2007.

[11] 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.

[12] 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.

[13] 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.

[14] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2004.

[15] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan. Speeding up distributed request-response workflows. In *SIGCOMM*, 2013.

[16] W. Jin, J. S. Chase, and J. Kaur. Interposed proportional sharing for a storage service utility. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):37–48, 2004.

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

[18] S. Kavalanekar, B. L. Worthington, Q. Zhang, and V. Sharda. Characterization of storage workload traces from production windows servers. In *IEEE International Symposium on Workload Characterization*, 2008.

[19] J. Li, N. K. Sharma, D. R. K. Ports, and S. D. Gribble. Tales of the tail: Hardware, OS, and application-level sources of tail latency. In *Proceedings of the ACM symposium on Cloud computing (SoCC)*, 2014.

[20] C. R. Lumb and R. Golding. D-SPTF: Decentralized request distribution in brick-based storage systems. *SIGOPS Oper. Syst. Rev.*, 38(5):37–47, 2004.

[21] 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.

[22] J. C. McCullough, J. Dunagan, A. Wolman, , and A. C. Snoeren. Stout: An adaptive interface to scalable cloud storag. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2010.

[23] A. Povzner, T. Kaldewey, S. Brandt, R. Golding, T. M. Wong, and C. Maltzahn. Efficient guaranteed disk request scheduling with fahrrad. In *Proceedings of the 3th European conference on Computer systems (EuroSys)*, 2008.

[24] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, 1996.

[25] 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.

[26] L. Suresh, M. Canini, S. Schmid, and A. Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *Proceedings of the USENIX conference on Networked systems design and implementation (NSDI)*, 2015.

[27] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low latency via redundancy. In *CoNEXT*, 2013.

[28] 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.

[29] A. Wang, S. Venkataraman, S. Alspaugh, R. Katz, and I. Stoica. Cake: enabling high-level SLOs on shared storage systems. In *Proceedings of the ACM symposium on Cloud computing (SoCC)*, 2012.

[30] 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.

[31] Z. Wu, C. Yu, and H. V. Madhyastha. CosTLO: Cost-effective redundancy for lower latency variance on cloud storage services. In *Proceedings of the USENIX conference on Networked systems design and implementation (NSDI)*, 2015.

[32] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey. Bobtail: Avoiding long tails in the cloud. In *Proceedings of the USENIX conference on Networked systems design and implementation (NSDI)*, 2013.

[33] J. Zhang, A. Riska, A. Sivasubramaniam, Q. Wang, and E. Riedel. Storage performance virtualization via throughput and latency control. *ACM Transactions on Storage (TOS)*, 2 (3):283–308, 2006.

[34] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger. PriorityMeister: Tail latency QoS for shared networked storage. In *Proceedings of the ACM symposium on Cloud computing (SoCC)*, 2014.