Customizable SLO and Its Near-Precise Enforcement for Storage Bandwidth

NING LI, School of Computer Science & Technology, Huazhong University of Science and Technology, Wuhan, China
HONG JIANG, Department of Computer Science and Engineering, University of Texas at Arlington, USA
DAN FENG and ZHAN SHI, School of Computer Science & Technology, Huazhong University of Science and Technology, Wuhan, China

Cloud service is being adopted as a utility for large numbers of tenants by renting Virtual Machines (VMs). But for cloud storage, unpredictable IO characteristics make accurate Service-Level-Objective (SLO) enforcement challenging. As a result, it has been very difficult to support simple-to-use and technology-agnostic SLO specifying a particular value for a specific metric (e.g., storage bandwidth). This is because the quality of SLO enforcement depends on performance error and fluctuation that measure *the precision of SLO enforcement*. High precision of SLO enforcement is critical for user-oriented performance customization and user experiences. To address this challenge, this article presents V-Cup, a framework for VM-oriented customizable SLO and its near-precise enforcement. It consists of multiple auto-tuners, each of which exports an interface for a tenant to customize the desired storage bandwidth for a VM and enable the storage bandwidth of the VM to converge on the target value with a predictable precision. We design and implement V-Cup in the Xen hypervisor based on the fair sharing scheduler for VM-level resource management. Our V-Cup prototype evaluation shows that it achieves satisfying performance guarantees through near-precise SLO enforcement.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Measurement Techniques, Modeling Techniques; D.4.2 [**Operating Systems**]: Storage Management—*Secondary storage*; D.4.7 [**Operating Systems**]: Organization and Design—*Distributed systems*

General Terms: Algorithms, Design, Management, Performance

Additional Key Words and Phrases: Cloud storage, storage management, service-level objective, end-to-end control

ACM Reference Format:
Ning Li, Hong Jiang, Dan Feng, and Zhan Shi. 2017. Customizable SLO and its near-precise enforcement for storage bandwidth. ACM Trans. Storage 13, 1, Article 6 (February 2017), 25 pages.
DOI: http://dx.doi.org/10.1145/2998454

1. INTRODUCTION

In a commercial cloud system (e.g., Amazon EC2 [Amazon EC2 2015], Microsoft Windows Azure [Windows Azure 2015], Google Compute Engine [Google Compute Engine 2015]), enterprise-class computing and storage can be offered as public utilities to

This work is supported by the National High Technology Research and Development Program of China Grants No. 2013AA013203, No. 2015AA015301, and No. 2015AA016701; NSFC Grants No. 61303046, No. 61472153, and No. 61402189; and US NSF Grants No. CNS-1116606 and No. CNS-1016609.
Authors' addresses: N. Li, D. Feng, and Z. Shi (corresponding author), Wuhan National Lab for Optoelectronics F307, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan 430074, China; emails: {leebellwind, dfeng, zshi}@hust.edu.cn; H. Jiang, Engineering Research Building, Room 640, Box 19015, Arlington, TX 76010; email: hong.jiang@uta.edu.

DOI: http://dx.doi.org/10.1145/2998454

tenants who pay for cloud services by renting Virtual Machines (VMs) on an as-needed basis. Many organizations move to the cloud by deploying the enterprise services in VMs. Hence, VM-oriented performance guarantee may be a vital and crucial requirement for the stable and efficient run of user applications. It is desirable for tenants to be able to express their demands in Service-Level Objectives (SLOs) for the rented VMs in simple and intuitive specifications that are enforced by the cloud system in a precise way. However, online commercial cloud services can only provide very limited guarantees of Quality-of-Service (QoS) in SLOs (e.g., monthly uptime percentage is higher than 99% [Amazon EC2 SLA 2015]) for tenants. The existing solutions of SLO-based management suffer from low adoption rates in products, largely because of the complexity, inaccurate enforcement of SLOs, and poor support for common tenants to select and specify SLOs.

Compared to network bandwidth, it may be harder to express the storage performance requirement as a simple and intuitive specification. This can be attributed to the highly unpredictable IO characteristics (i.e., request size, the degree of sequentiality, read and write ratio, etc.) that are heavily workload dependent and storage specific, challenging the storage performance guarantees. In contrast, network bandwidth allocation is not heavily influenced by the workload-dependent factors such as message size that can be uniformly regulated by protocol configurations. Consequently, approaches such as max-min control [Shue et al. 2012; Gulati et al. 2010, 2012] and proportional sharing [Gulati et al. 2009; Wachs et al. 2007; Jin et al. 2004; Wu and Brandt 2006; Povzner et al. 2008] have been proposed for storage resource management, which are characterized by nonintuitive and technology-dependent SLOs for end users.

Near-Precise SLO Enforcement (N-PSE) in this article refers to the support of a simple, technology-agnostic and user-oriented SLO that specifies a particular value for a specific metric (e.g., bandwidth of 30MB/s) with predictably small performance error and fluctuation. Thus, a user can participate in the formulation of SLO and be given the quality of performance guarantees quantified by the performance error within a *Statistical Interval*, or *SI* for short, which reflects the timeliness of SLO enforcement, and a degree of performance fluctuation predicted from profiling the user application. This enables an arguably more reasonable and enticing pricing model in which customers pay for their rendered services in terms of the quality of SLO enforcement instead of the consumed resource capacity, a model that is also evidently desired by the users of cloud services [Nathuji et al. 2010]. However, the state-of-the-art approaches based on max-min control [Gulati et al. 2010, 2012; Shue et al. 2012] mostly focus on the system-wide resource fair sharing among VMs rented by the tenants, which enforces a performance range between the reservation and the upper limit of the resources based on the proportional-sharing model. Obviously, it is nonintuitive and in fact difficult for novice users of cloud services to specify their performance requirements by customizing the complex and technology-dependent SLOs in terms of reserved resources, resource upper limit, proportion of sharing, etc. Moreover, the error between the measured performance and the target performance in SLO is closely associated with the length of the interval in which the error is measured. For example, an error of the same average value over 1 second is likely very different from that over 1 minute. This time sensitivity of performance error, as we will demonstrate, will have a significant impact on the timeliness of IO control by the I/O scheduler of the VM hypervisor that is typically driven by the measured performance and affect the quality of SLO enforcement. Unfortunately, the quality of SLO enforcement in terms of timeliness and its impacts have not been thoroughly explored for VM storage. In this article, we focus on near-precise SLO enforcement by exploiting the association and interplay between the timeliness of IO control and the precision of SLO enforcement.

In addition, the IO-resource-management-based approaches, such as mClock [Gulati et al. 2010], SRP [Gulati et al. 2012], and Pisces [Shue et al. 2012], emphasize IO

throughput allocation across VMs and thus are difficult to be extended to allocating storage bandwidth. In the cloud environment, bandwidth is the commonly used metric for both the network and storage subsystems because it is intuitive for novice users and directly relevant to IO-bound applications [Liu et al. 2008; Wu et al. 2011; Lu et al. 2012] running in a cloud. However, SLO enforcement in the metric of storage bandwidth is arguably more challenging than in the metric of IO throughput because the former is inherently compounded by two conflicting factors, IO throughput and request size.

To address the preceding problems, we propose a semantic layer tied to the individual VM as a complement to the resource-management-based approaches. Through the technique of fine-grained performance tuning that limits the request size variability and provides better IO control timeliness, the storage bandwidth of a VM can be converged on the desired value with a predictable precision. We implement the semantic layer as a distributed framework for VM-oriented customizable SLO and its near-precise enforcement in cloud storage, also called *V-Cup* for short. It consists of multiple auto-tuners integrated into the IO scheduler of Xen hypervisor [Barham et al. 2003]. Since each auto-tuner acts like a wrapper for the workloads running on a VM, V-Cup is transparent to the VM layer and to the resource management. Our results show that V-Cup achieves satisfying performance guarantees with near-precise SLO enforcement for the storage bandwidth of VMs in all the evaluation experiments.

The rest of the article is organized as follows. Section 2 discusses the relevant metrics that are considered necessary for quantifying and measuring the precision of SLO enforcement. Section 3 presents the framework design of V-Cup, and the implementation issues are discussed in Section 4. A detailed performance evaluation on a real system is presented in Section 5. Section 6 presents the related work of QoS-based storage management under virtualized data centers. Finally, we conclude the article with an overview for future work in Section 7.

2. DEFINING AND QUANTIFYING PRECISION OF SLO ENFORCEMENT

Average-based metrics are widely adopted in existing SLO-based approaches [Wachs et al. 2007; Lumb et al. 2003] for storage management. Even for the state-of-the-art approaches based on max-min control [Gulati et al. 2010, 2012; Shue et al. 2012; Thereska et al. 2013], the lower and upper bounds set for performance range can also be considered as the average-based metrics that bound the performance fluctuation but are time insensitive in that they fail to reflect and reveal the timeliness consequence of fluctuation. This time insensitivity of average-based metrics decreases IO control timeliness, which in turn adversely affects user experiences.

For example, a video server can use one of the commonly used buffer cache mechanisms, such as interval caching [Dan and Sitaram 1996] and FlashStream [Ryu and Ramachandran 2013], to reduce the overhead of disk accesses and absorb bursty user requests by exploiting temporal data-access locality and caching/prefetching the potentially reusable data with higher IO cost. As a result, the video server is able to increase its effective capacity to tolerate the fluctuation of storage performance to a certain degree. However, a persistent and prolonged below-SLO storage performance can cause the buffer cache to be eventually depleted of useful data for the users because the prefetching speed can no longer keep up with the user demand, rendering the buffer cache mechanism ineffective. On the other hand, maintaining an above-SLO storage performance would mean an unnecessary over-provisioning of resources.

As shown in Figure 1, where the maximum allowable Statistical Interval (SI) is assumed to be 30 seconds, during which the obtained average performance approaches the target performance specified in SLO (1MB/s) with an acceptable error (e.g., $\leq$10%) for an application, the SLO enforcement of Case 1 can lead to a continuous 50-second duration of below-SLO storage performance every 100 seconds and cause the user to

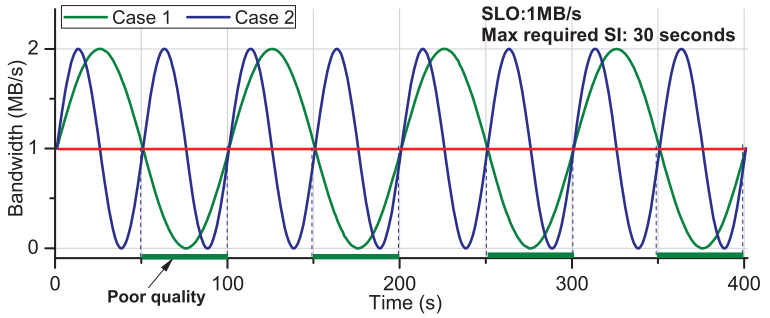

Fig. 1. The illustration of the timeliness for SLO enforcement. Note that Case 1 suffers from 50-second periods of continuous below-SLO storage performance, in contrast to the 25-second ones for Case 2 with much less adverse impact to user QoS experience than Case 1, despite identical Max-Min bounds for both cases.

suffer from poor quality of SLO guarantees. In contrast, the SLO enforcement of Case 2 lowers the below-SLO storage performance to a continuous period of only 25 seconds per 50 seconds, which can provide a better user experience. This is because a 50-second continuous below-SLO performance is far more likely to render the employed buffer cache mechanism ineffective than a 25-second one, as discussed earlier. What is more important to notice is the fact that this happens despite the fact that the average bandwidth and max-min bounds obtained during a period longer than 100 seconds for both cases are identical. Hence, we believe that the average performance error during a specific SI is the desired metric to evaluate the timeliness of IO control for SLO enforcement. Since storage bandwidth is commonly measured in terms of KB/s or MB/s, we choose the minimum interval of measurement to be 1 second.

Definition 2.1. Storage bandwidth error (B): Let b_i be the storage bandwidth measured at the i^{th} second of the interval $[t_1, t_2]$. Let $T = (t_2 - t_1 + 1)$. Let $\hat{b}$ be the target value of the storage bandwidth in SLO. The storage bandwidth error in the interval $[t_1, t_2]$ is defined as

$$B = \left(\sum_{i=t_1}^{t_2}(b_i - \hat{b})/\hat{b}\right)\Big/T. \tag{1}$$

The interval during which B is derived represents the timeliness of IO control for SLO enforcement. It is referred to as the *statistical interval* (SI). Based on the metric of B, absolute storage bandwidth error (β) is proposed to measure the quality of guarantee for the average storage bandwidth. *A smaller value of β in a smaller SI is preferred* (e.g., β in the SI of 30 seconds is smaller than 10%).

Definition 2.2. Absolute storage bandwidth error (β): Let B_i be the storage bandwidth error measured at the i^{th} SI [(i-1)·λ, i·λ) in the interval $[t_1, t_2]$, where λ is the length of SI. Let $N = (t_2 - t_1 + 1)/\lambda$. The absolute storage bandwidth error in the interval $[t_1, t_2]$ is defined as

$$\beta = \left(\sum_{i=1}^{N}|B_i|\right)\Big/N. \tag{2}$$

In addition, the degree of performance fluctuation reflects the extent of performance stability. For the storage-intensive applications of tenants (e.g., video server, file server, and web server, etc.), stable storage bandwidth means stable execution, which is very

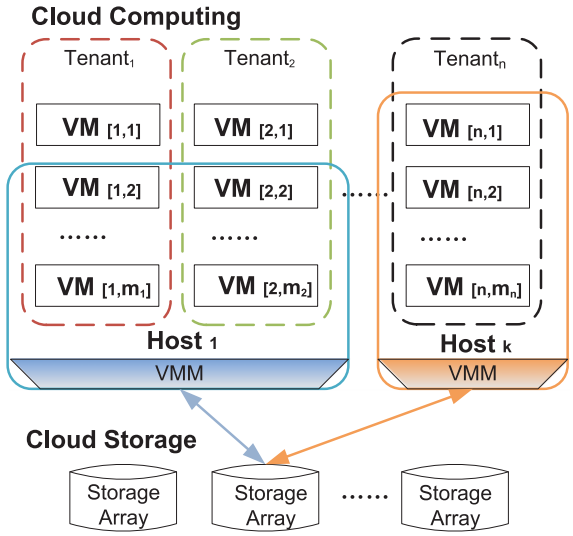

Fig. 2. System model.

important to the user experience. *A smaller value of the storage bandwidth variability V means a higher stability for the storage bandwidth* (e.g., V is smaller than 20%).

Definition 2.3. Storage bandwidth variability (V): Let b_i be the storage bandwidth measured at the i^{th} second of the interval $[t_1, t_2]$. Let $T = (t_2 - t_1 + 1)$. Let $\overline{b}$ be the average storage bandwidth during the interval $[t_1, t_2]$. The storage bandwidth variability in the interval $[t_1, t_2]$ is defined as

$$V = \left(\sum_{i=t_1}^{t_2} |b_i - \overline{b}| / \overline{b} \right) \Big/ T. \tag{3}$$

3. THE V-CUP FRAMEWORK

3.1. System Model

V-Cup is designed for a cloud platform consisting of a cloud-computing system supported by a cloud storage subsystem. As shown in Figure 2, the cloud-computing system provides the shared infrastructure of utility computing for tenants by encapsulating the user applications into VMs. And the cloud storage subsystem involves one or more storage arrays. Each storage array can be partitioned into several logical units (LUNs) as the virtual disks (i.e., VM disk) dedicated to VMs. A VM disk is represented by a logical volume on an exclusive LUN. We call the VMs that have the VM disks on the same array *coscheduled VMs*. Thus, the VM cluster in the cloud-computing system can share the storage arrays connected over a Storage Area Network (SAN). As a result, V-Cup views the underlying storage devices and network infrastructure supporting coscheduled VMs as a black box. And the interferences from the inner operations of storage devices, such as garbage collection in Solid State Drive (SSD) [Kim et al. 2015], RAID reconstruction [Wu et al. 2009, 2012], etc., are represented by VM-level performance fluctuation measured by storage bandwidth error and storage bandwidth variability for V-Cup.

However, a VM is not always pinned to a specific host, namely, for a VM disk, due to live migration [Mashtizadeh et al. 2011; Bradford et al. 2007]. Moreover, each rented VM is required to service a specific tenant with a desired SLO. Therefore,

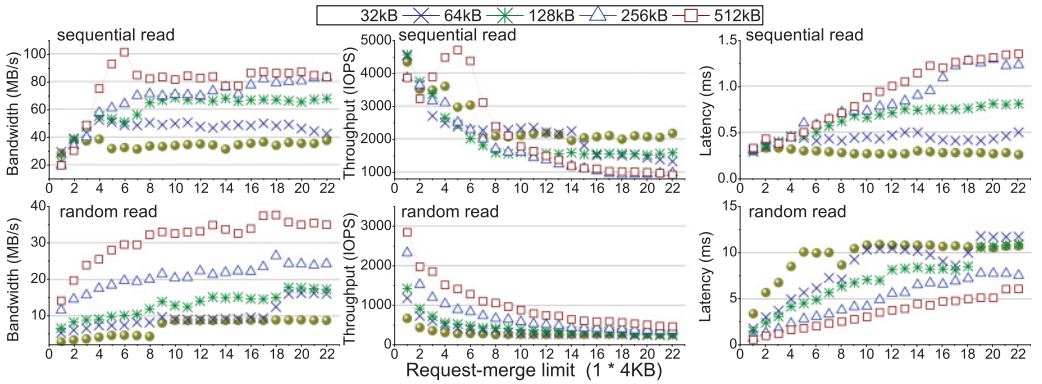

Fig. 3. The variation of bandwidth, throughput, and latency for the sequential or random read workloads with IO size ranging from 32KB to 512KB under the control of request-merge limit.

V-Cup is constructed based on multiple auto-tuners integrated into the IO scheduler of hypervisor, aiming to support VM-oriented SLO customization and enforcement.

A VM disk is actually a black box, resulting from resource management, from the viewpoint of the auto-tuner of V-Cup. Each VM can maintain a set of pending requests at the VM disk (i.e., IO queue). This IO queue represents the IO requests scheduled by the VM and currently pending at the array. IO queue size can have a great impact on the throughput and IO latency for a VM. Fortunately, IO queue size of a workload can be bounded adaptively to the observed latency for a better trade-off between the desired throughput and latency [Gulati et al. 2009; Jin et al. 2004]. In addition, IO throughput can also be limited for the target IOPS in SLO by IO throttling [Karlsson et al. 2005; Lumb et al. 2003; Zhang et al. 2006] or max-min control [Gulati et al. 2010, 2012; Shue et al. 2012]. Compared to IO queue size and IO throughput, request size is mostly workload dependent and hard to control on the side of hypervisor IO scheduler. More importantly, highly variable request size can directly affect storage performance (e.g., bandwidth) in an uncontrolled fashion, which can lead to unpredictable SLO enforcement. Thus, it is hard to guarantee the quality of SLO enforcement. So it is necessary to evaluate the impacts of request size variability on the storage performance in different metrics.

3.2. Impacts of Request Size Variability

To explore the effects of bounding the request size variability on storage performance, we conduct an experiment by running a VM (created by Xen [Barham et al. 2003]) that excessively accesses a RAID 0 disk group over two disks under a read workload mimicked by FIO [FIO 2015] with different IO characteristics (i.e., IO size issued by the workload ranges from 32KB to 512KB, sequential and random access patterns, etc.). We limit the upper bound of the request size issued to the disk array by gradually increasing the request-merge limit from 4KB in an increment of 4KB every 30 seconds. In this way, we can observe the storage performance under the upper limit of request size increased from 4KB and onwards.

Observing the experiment results shown in Figure 3, we are led to the following conclusions: (1) The storage performance in different metrics (bandwidth, throughput, and latency) can all be affected by request size variability to various extents for either sequential or random workloads issuing requests with different sizes. (2) The effect of request size variability on storage performance becomes more pronounced with the increasing IO size issued by workloads. (3) Relaxing request-merge limit (i.e., forming

larger IOs by increasing the limit) beyond a certain point will not further increase bandwidth significantly as bandwidth tends to plateau with the increasing request-merge limit, but instead a higher latency or lower throughput will result. Therefore, we argue that bounding request size by setting request-merge limit appropriately can potentially support a better trade-off between the desired bandwidth or throughput and the allowable latency. This motivates us to explore this property in this V-Cup study, where we will focus on the bandwidth aspect of the property, and in our future work.

3.3. Bandwidth Control by Adjusting Request-Merge Limit

Merge of requests, which groups small requests with adjacent addresses into a single larger request, is a core function of locality optimization in the block layer and disk schedulers. When a disk request is issued by an application, it will be handled by the file system as follows. The IO request will be first processed by the file system to allocate multiple segments (where each segment is a physical memory page) to accommodate it. These segments, as the basic merge units, are then encapsulated into an intermediate structure (e.g., bio in the Linux kernel) and merged in the block level before being sent to the disk scheduler as an incoming request. If the request has a chance to be merged with a pending request in the disk scheduler, a merge operation will be carried out on the condition that the size of the merged request does not exceed the request-merge limit.

Request-merge limit is the upper bound of the request size after request merge. If the sum of IO sizes of two mergeable requests is larger than the request-merge limit, the request merge will be disallowed.

Request merge can have a substantial impact on the pattern of request service time consumed by the disk mechanism that consists of seek time, rotation delay, and transfer time. This is because request size executed by the disk can be increased by request merge. Consequently, the data-transfer portion of the request service time is improved. This means that storage bandwidth can be increased if the IO rate is kept stable. However, request merge can inherently delay requests in the queue of the IO scheduler for a chance to form a larger request. So request merge has the dual effects of reducing IO rate (i.e., request merge reduces the number of IOs per unit time) and increasing request size. Thus, the impact of request merge on storage bandwidth can be divided into two broad stages. In the first stage, the request-merge limit increases gradually from a minimum unit σ (e.g., a physical memory page, 4KB) until the bandwidth reaches its maximum value. Nevertheless, the increasing trend of bandwidth can be weakened since the IO rate may be reduced by request merges. Moreover, the increase in request size resulting from relaxing the request-merge limit is still limited by the upper bound of IO size issued by the application and affected by the degree of sequentiality. Thus, in the second stage, bandwidth tends to become stabilized and flattened out as the request-merge limit continues to be relaxed.

We aim to map the request-Merge Limit Range (MLR) into as large a corresponding Storage Bandwidth Range (SBR) as possible. Thus, a wider adjustable range of bandwidth can be offered to tenants of cloud services. So, before attempting to guarantee the storage bandwidth of a VM, V-Cup will first carry out a profiling test running in the real environment of VM consolidation to obtain average values of bandwidth in a fix-length interval (e.g., 1 second) by increasing the request-merge limit by σ from σ to MAX_ML[1] $\cdot\sigma$ (the upper bound for the MLR). If the bandwidth value b obtained under the request-merge limit L is the highest within the MLR, the bandwidth range of $(0, b]$

[1]*MAX_ML* is limited by the maximum physical pages per request allowed by hypervisor (e.g., 11 is allowed for Xen [Barham et al. 2003] by default).

is accepted as the SBR guarantee for the VM. This leads to MLR being adjusted from $[\sigma, MAX\_ML \cdot \sigma]$ to $[\sigma, L \cdot \sigma]$. However, if the bandwidth in the smallest request-merge limit is b°, how do we control the bandwidth within the range $(0, b^{\circ})$? We adopt the *IO throttling* technique based on the smallest request-merge limit to achieve the required control. To guarantee the precision of storage bandwidth SLO enforcement, the delay time between two consecutive IO requests (i.e., *delay slice*) is a key factor in IO throttling for a VM. Too short a delay slice may be insufficient to dampen the rising bandwidth while too long a delay slice can lead to a bandwidth lower than the target value. So the length of the delay slice is given in the throughput target derived from the bandwidth SLO and the request size obtained under the current request-merge limit.

If there is no need to guarantee an IO throughput target (i.e., no IO throughput SLO), V-Cup can enforce a bandwidth SLO by adjusting the request-merge limit when the bandwidth SLO falls in the SBR. In this way, the IO throttling submodule does not work. When the bandwidth SLO is below the SBR, V-Cup will set the delay slice length to be the ratio of a physical page size to the bandwidth SLO to drive IO throttling. If there are both throughput SLO and bandwidth SLO, V-Cup will carry out IO throttling according to the throughput SLO target and implement request-merge limit control to achieve the required request size (i.e., the bandwidth SLO divided by the throughput SLO). In fact, V-Cup can potentially adopt almost all existing IO throughput control techniques (e.g., mClock [Gulati et al. 2010], Leaky bucket technique [Chambliss et al. 2003], and proportional-integral control [Franklin et al. 1998; Hellerstein et al. 2004]) to both enforce bandwidth SLO and throughput SLO for coscheduled VMs, which is demonstrated in Section 5.

3.4. Fine-Grained Performance Tuning

Bandwidth control by adjusting the request-merge limit can only provide a set of scattered and coarse-grained values. For example, a VM can have a bandwidth of 15MB/s with a request-merge limit of 4KB, and 70MB/s with a request-merge limit of 8KB. The request-merge limit can be increased by an increment σ of 4KB at a time, but how to control the bandwidth to reach the target of, say, 30MB/s?

We define the function for bandwidth control as $f(x)$ where x refers to the control variable (i.e., request-merge limit) and the range of $f(x)$ is the storage bandwidth. *Fine-grained performance tuning* refers to adjusting the performance to a desired objective between $f(x_1)$ and $f(x_2)\,(x_1 > x_2)$ by setting the control variable x at x_1 or x_2 at different times in a fixed-length interval ω. For simplicity, we divide ω into n equal-length *time windows*, where the length of the time window is called *Window Size* (WS). And setting control variable x at x_1 (or x_2) for a time window is referred to as a *tuning action*. A smaller WS means better IO control timeliness (i.e., higher frequency of tuning actions), which can be beneficial to reducing the error between the weighted arithmetic mean achieved by fine-grained performance tuning and the target value. Nevertheless, the number of requests collected during a smaller WS may be too small to derive sufficiently precise statistics of performance, which is the key indicator for the system to make a right decision. In other words, fine-grained performance tuning depends on an appropriate trade-off between the IO control timeliness and the precision of performance statistics by setting an appropriate WS.

4. IMPLEMENTATION ISSUES

We implement a prototype of V-Cup based on the fair-sharing scheduler (e.g., CFQ [Axboe 2004] and mClock [Gulati et al. 2010]) across coscheduled VMs. And V-Cup, integrated into the hypervisor, enforces storage bandwidth SLOs for coscheduled VMs by establishing multiple auto-tuners each of which is dedicated to an individual VM.

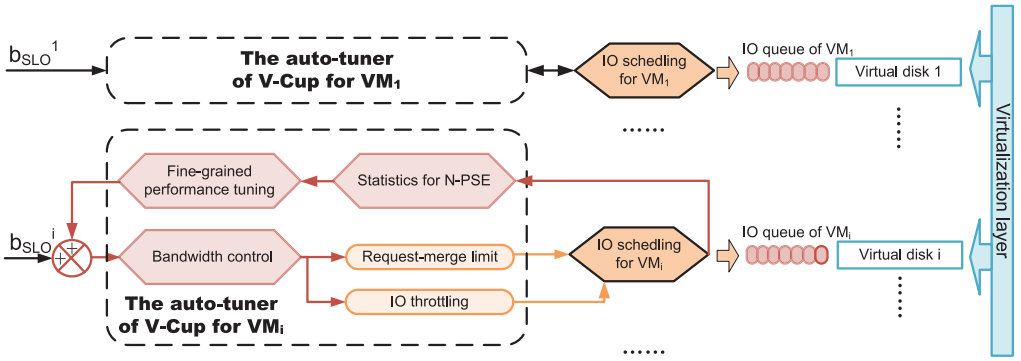

Fig. 4. An illustration of the feedback control in V-Cup.

As shown in Figure 4, an auto-tuner consists mainly of three modules that are responsible for *bandwidth control*, *statistics for N-PSE*, and *fine-grained performance tuning*, respectively. V-Cup assumes the existence of a fair-sharing scheduler.[2] IO capacity in terms of throughput can be allocated to VMs according to a specific fair-sharing algorithm, based on which the auto-tuner of V-Cup for each VM can carry out a feedback control to enable the bandwidth to converge on the target value specified in SLO. The module of statistics for N-PSE can trigger the module of fine-grained performance tuning only if the precision of SLO enforcement is worse than the desired level (e.g., storage bandwidth error (B) is within the range of $(-5\%, 5\%)$). In this case, the module of bandwidth control can be called to adjust the request-merge limit that controls the process of request merge or change the length of the delay slice during which IO dispatching is temporarily blocked (i.e., forced delay of IO dispatching). The precision of SLO enforcement, that is, how closely storage bandwidth converges on the target value, can be increased if the auto-tuner can make the right decision when carrying out bandwidth control and IO control timeliness can be guaranteed. In addition, better IO control timeliness is critical to reducing the fluctuation of bandwidth and more rapidly coordinating bandwidth to adapt to variable IO characteristics. This relies on the appropriate configuration for WS.

The overhead of V-Cup stems mainly from collecting IO statistics for each running VM, which consumes a tiny fraction of the CPU resources (smaller than 1% CPU utilization in all the experiments reported in Section 5). In addition, V-Cup hardly requires an additional memory or disk storage resources.

4.1. Bandwidth Control

The bandwidth control module controls the storage bandwidth for the VM by adjusting the request-merge limit or the length of the delay slice. The variable `max_phys_segments` in the Linux kernel is used by V-Cup to delimit the maximum IO size for the requests in the IO queue at the block layer. When the total IO size of two mergeable requests exceeds the maximum IO size, the request merge will be disallowed. IO throttling is adopted as a submodule in the bandwidth control module, which is used to delay IO dispatching for a period of time equal in length to the specified delay slice.

4.2. Statistics for N-PSE

The module of IO statistics is used to monitor and obtain the SLO enforcement precision measures of B (Definition 2.1), β (Definition 2.2), and V (Definition 2.3) for different

[2]V-Cup does not confine the fair-sharing scheduler to the host level, a switch or the storage server.

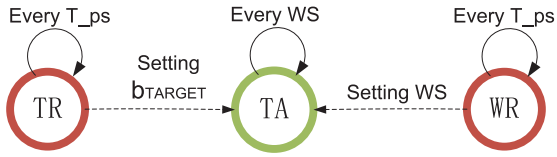

Fig. 5. Relationship and interaction among TR, WR, and TA.

Table I. Notations and Their Definitions

Notations	Descriptions
b_{SLO}	Storage bandwidth SLO
b_{TARGET}	The target value used in TA as the objective of adjustment
T_ps	The statistical interval
V_ps	The storage bandwidth variability obtained over T_ps
β_ps	The absolute storage bandwidth error obtained over T_ps
BEST_V	The threshold of storage bandwidth variability
B_threshold	The threshold of absolute storage bandwidth error
last_V	The storage bandwidth variability obtained over the last statistical interval
U	The threshold to discern if the newly measured V_ps suffers from sudden IO interference
UN_WIN	The minimum search range size for optimizing the length of time window

strategies (i.e., real-time statistics and periodic statistics) by collecting the information of IO dispatching from the IO scheduler. *Real-time statistics* refers to the statistics in WS. *Periodic statistics* means the statistics over a longer period of time (e.g., `T_ps`, set at 30 seconds).

4.3. Fine-Grained Performance Tuning

The fine-grained performance tuning involves three complementary and interactive operations: Tuning Action (TA), Target Revision (TR), and time Window Revision (WR). TA enables the storage bandwidth of a VM to converge on the target value. However, the dynamic IO characteristics and the fluctuation in allocated resources can affect the error between the measured bandwidth and the target value specified in SLO (i.e., b_{SLO}). So, as shown in Figure 5, V-Cup uses TR to optimize the target value b_{TARGET} used in TA as the objective of adjustment. WR is responsible for choosing an appropriate WS for TA, which can be triggered by a higher V value than a threshold (e.g., 20%) or the initial configuration. In terms of statistical methods, TR and WR use the period statistics (i.e., running every `T_ps`), while TA uses the real-time statistics (i.e., running every WS). The used notations are explained in Table I.

TA: The storage bandwidth error `B_tw` is obtained between the average bandwidth measured in the last time window and b_{TARGET}. If `B_tw` is out of the allowable range (e.g., [−5%, 5%]), TA controls the bandwidth by adjusting the request-merge limit during MLR or the length of delay slice (i.e., `delay_slice_len`). As discussed in Section 3.3, bandwidth can be basically increased with the request-merge limit within MLR. So TA can increase the request-merge limit by a minimum unit at a time for a higher bandwidth, and vice versa. Since TA runs every WS (often more than 10HZ), TA can quickly find two consecutive request-merge limits suitable for fine-grained performance tuning for b_{TARGET}. Thus, even though the bandwidth as a function of the request-merge limit is not strictly monotonous, TA can still work well albeit at the cost of consuming some time windows in search of the proper request-merge limit. If the request-merge limit reaches the minimum request-merge limit and `B_tw` is still greater than the upper

ALGORITHM 1: Time Window Revision (WR) Algorithm

Input: Storage bandwidth variability over T_ps (V_ps), the absolute storage bandwidth error over T_ps (β_ps), and the SLO target of storage bandwidth (b_{SLO}).
Output: The suboptimal WS (ws).
$ws\_max = 1024$ ms; $ws\_min = 32$ ms; $ws = ws\_max$;
repeat
  if *($V\_ps - U \leq last\_V$) and ($\beta\_ps \leq B\_threshold$);*
  then
    $ws\_max = ws$;
  else
    $ws\_min = ws$;
  end
  $ws = (ws\_min + ws\_max) / 2$;
until *($ws\_max - ws\_min \leq UN\_WIN$) or (($V\_ps \leq BEST\_V$)and ($\beta\_ps \leq B\_threshold$))*;

bound of allowable range, TA will set `delay_slice_len` to be the ratio of a physical page size to the bandwidth SLO to drive IO throttling. Nevertheless, TA can fail under the condition that `B_tw` is lower than the lower bound of the allowable range and request-merge limit reaches the maximum request-merge limit. In this case, the IO capacity allocated to the VM is inadequate for the target bandwidth and IO throughput reallocation [Gulati et al. 2010] or VM migration [Mashtizadeh et al. 2011] may be triggered.

WR: As shown in the WR algorithm, it is an $O(\log n)$ algorithm. We first analyze the time complexity of the WR algorithm as follows. Suppose the initial size of the search space of WS is n for the first iteration, the value will be n/2 for the second iteration. By that reasoning, the search space size will be $n/4, n/8, \ldots, n/(2^k)$ for the subsequent iterations. And k is the maximum number of iterations. So the time complexity of the WR algorithm can be represented by $O(\log n)$. As a heuristic optimizer, the goal of WR is to search for a suboptimal WS (i.e., `ws`) for TA to guarantee storage bandwidth variability over the SI of `T_ps` (i.e., `V_ps`) that is smaller than the threshold of `BEST_V`. Moreover, the absolute storage bandwidth error β_ps in `T_ps` should be kept lower than a threshold (i.e., `B_threshold`). In this case, WR returns `ws` as a suboptimal WS for TA. Otherwise, WR will set the upper bound of WS (i.e., `ws_max`) as `ws` only if `V_ps - U` is smaller than or equal to `last_V` (i.e., `V_ps` measured in the last `T_ps`), or else the lower bound of the time window is changed to `ws`. `U` is a parameter for reducing the impact of sudden IO interference on `V_ps`, which, if unhindered, can prematurely stop the exploration of WR for a more optimizing WS. And the average values of `ws_max` and `ws_min` will be set as the input parameters of WR for the next iteration that will be triggered after an interval of `T_ps`. WR will continue to achieve its goal until the difference between `ws_max` and `ws_min` becomes smaller than `UN_WIN`, at which point WR will exit.

TR: It aims to keep the storage bandwidth error `B_ps`, which is obtained between the target b_{SLO} specified in SLO and the measured bandwidth `b_ps` averaged over `T_ps`, within an allowable range (e.g., [−5%, 5%]) by adjusting b_{TARGET} for the TA algorithm. The plus or minus sign of `B_ps` shows the change direction of the adjustment variable denoted by `adjust_factor` (i.e., decreasing or increasing b_{TARGET} by `adjust_factor` percent of b_{SLO}). To adjust b_{TARGET} in time, TR linearly increases `adjust_factor` for the case of `B_ps` being larger than a threshold (e.g, 10%) by multiplying the number of consecutive adjustments in the same change direction. For the first run, `adjust_factor` uses `B_ps/2` as the initial correction. In addition, `adjust_factor` is increased or decreased by the amount of 2% at a time.

4.4. VM-Oriented End-To-End Control in V-Cup

The auto-tuner of V-Cup is able to carry out VM-oriented end-to-end storage bandwidth control with the request-merge limit submodule and IO throttling submodule for each VM. Thus, for an auto-tuner, the inputs are the prespecified storage bandwidth SLO for and the IO statistics obtained from a VM, while outputs are the request-merge limit or delay slice length for the VM, resulting in a closed-loop feedback control. As a result, each auto-tuner can implement single-purpose optimization for its own control target without additional impacts from other auto-tuners, which obviously helps achieve stable and accurate control. When live migration happens for a VM, the auto-tuner previously of the VM will be revoked. And the V-Cup on the destination host where the VM migrates to will reinstall an auto-tuner for the VM. In doing so, the VM-oriented end-to-end control mechanism of V-Cup is able to work well in spite of VM live migration. If storage migration [Mashtizadeh et al. 2011] takes place for a VM, only the Storage Bandwidth Range (SBR) profiling test (elaborated in Section 3.3) for the VM will be redone. However, it is noted that VM storage migration is commonly time consuming and occurs relatively rarely. Thus, the cost of redoing the SBR profiling test triggered by VM storage migration is almost negligible.

5. PERFORMANCE EVALUATION

In this section, we present the results of a detailed evaluation of V-Cup in a real cloud-computing system established by Xen 3.4.2 [Barham et al. 2003] and based on a large-scale disk array. To provide IO resources fairly for coscheduled VMs, CFQ [Axboe 2004], the default IO scheduler for mainstream Linux versions, is adopted as the Fair-Queuing scheduler (FQ). In addition, the state-of-the-art approach mClock [Gulati et al. 2010] is used as another hypervisor IO scheduler across VMs to verify V-Cup's ability to enhance existing state-of-the-art approaches by being orthogonal and complementary to them. The cloud-computing server is a PowerLeader PR2760T server with two Intel Xeon E5620 quad-core processors, 12GB of RAM, and 10Gbit/s NIC connected to a disk array over an iSCSI SAN. Unless otherwise specified, the VM disks are hosted on a 16-disk (7200RPM, 250GB) RAID 0 disk group on the array.

The two baseline systems are cloud-computing systems with fair allocation of storage resources (e.g., IO throughput) for VMs by FQ and mClock [Gulati et al. 2010], respectively, referred to as FQ and mClock. Our V-Cup prototype systems based on FQ and mClock, referred to simply as V-Cup + FQ and V-Cup + mClock, provide VM-oriented customizable SLO with its near-precise enforcement for the storage bandwidth of VMs. This allows us to comprehensively evaluate V-Cup's effectiveness by comparing the two cases in a cloud-computing system with and without V-Cup. We have the following two objectives for the experimental evaluation of V-Cup. The first is to prove the effectiveness of V-Cup in providing near-precise SLO enforcement for the storage bandwidth of VMs by observing the measures of absolute storage bandwidth error (β) in a SI of 30 seconds and storage bandwidth variability (V). The second objective is to verify the VM-oriented customizability of SLO supported by V-Cup. All the results of our experiments are collected from the IOs issued from the VMs that include the IOs requested by the benchmarks, the software supporting the benchmarks (e.g., MongoDB and Java running environment), and the guest OS.

5.1. Control-Specific Challenges

The TR and WR mechanisms are designed to address two control-specific challenges facing the TA, namely, the phase effect and the performance drift, to be elaborated later. Briefly, the former refers to the fact that the β and V measures exhibit three distinct phases as the WS decreases, while the latter signifies the notable changes in

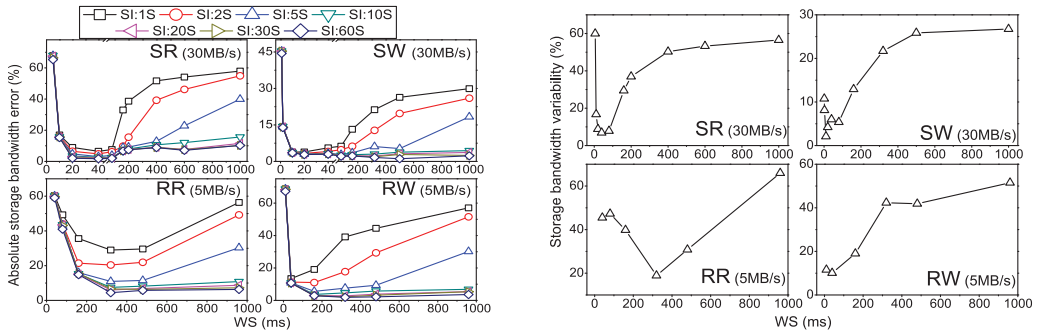

Fig. 6. Absolute storage bandwidth error β (figure on the left) and storage bandwidth variability V (figure on the right) as a function of the window size (WS) under the fine-grained performance tuning. Recall from Section 2 that a smaller value of β in a smaller SI is preferred, while a smaller value of V means a higher stability for the storage bandwidth.

β as IO characteristics and thus resources allocated to a VM change dynamically. To better observe and understand the role in these challenges played by IO characteristics (e.g., sequential or random, read or write, fixed or variable IO size), we use FIO [FIO 2015] in the following experiments to simulate seven different workloads by forking 10 threads in a VM to seek their 100G files on the VM disk with asynchronous IO engine. These workloads are Sequential Read (SR), Sequential Write (SW), Random Read (RR), Random Write (RW), Sequential Read with Variable request size (SRvar), Sequential Write with Variable request size (SWvar), and Random Read with Variable request size (RRvar). For the workloads with variability (i.e., with a "var" in their labels), IO size can range from 4KB to 128KB; otherwise, the IO size is fixed at 128KB. The IO depth in FIO is set at 20 for all seven workloads.

5.1.1. Phase Effect and WR Enforcement. Based on the analysis in Section 3.4, the IO control timeliness depends on the configuration of WS that determines the frequency of tuning actions. In fact, if we continuously increase this frequency, starting from 1HZ (i.e., a WS of 1 second), there are three successive phases that exhibit distinctive characteristics on absolute storage bandwidth error β and storage bandwidth variability V. Moreover, the phenomenon, which we refer to as *phase effect*, exists for workloads of almost all the IO characteristics (random or sequential, read or write). To verify the phase effect, we deploy four workloads, SR, SW, RR, and RW, in a VM, respectively, to access a two-disk RAID zero disk group under different WSs and plot the values of β and V over 5 minutes in Figure 6.

As shown in Figure 6, the patterns of change in storage bandwidth can be divided into three broad phases with the increase in the frequency of the tuning actions. For the first, as a characteristic of larger WSs, the absolute storage bandwidth error β can be maintained at a lower level but in a larger SI. In addition, a larger WS often means a larger storage bandwidth variability value V that indicates a more volatile fluctuation for the storage bandwidth measure. If WS is decreased continuously, the second phase emerges with the characteristics of a smaller β in a smaller SI and a smaller V. That is, near-precise SLO enforcement can be achieved in this phase. But, if we continue to reduce WS, the third phase will start to appear, which is characterized by a trend of increasing β and V values. This is because the number of IOs in the time window is too small to provide adequately precise average bandwidth. So, there clearly exists an optimal WS for a given workload. In theory, we can find the optimal WS by exhaustive search in the WS domain from 1ms to 1000ms. This, of course, is time consuming and

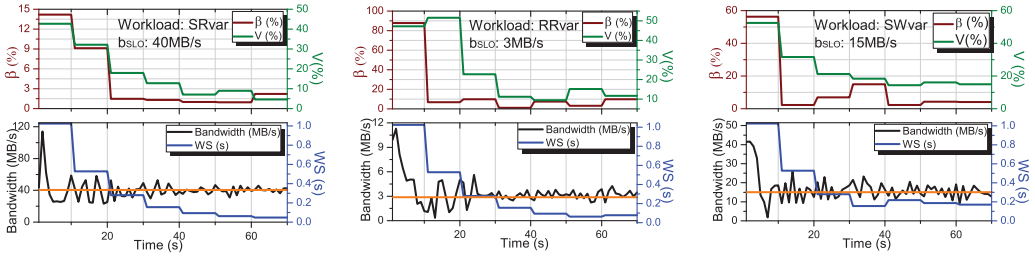

Fig. 7. The process of time window revision for the workloads of SRvar (figure on the left), RRvar (figure in the middle), and SWvar (figure on the right), showing the relationship between the measured bandwidth and the window size (WS), as well as the impact of the latter on the precision of SLO enforcement.

inefficient. To address the problem, the time WR algorithm is designed to optimize WS in an efficient manner.

We use the SRvar, RRvar, and SWvar workloads to verify the effectiveness of the WR algorithm for the workloads. In this and the following experiments, we set WS at 1,024 as the initial value of WS, while `ws_min` and `ws_max` are set at 32 and 1,024, respectively. `BEST_V` and `B_threshold` are set at 5% and 10%, respectively, as the goals for WR. The desired bandwidths b_{SLO} for SRvar, RRvar, and SWvar are 40MB/s, 3MB/s, and 15MB/s, respectively. We set `T_ps` at 10 seconds and `U` at 3%. As shown in Figure 7, for SRvar, the storage bandwidth variability V value can be seen to decrease for the most part except for a minor increase of 1.92% in the sixth iteration that is smaller than `U`, while the absolute storage bandwidth error β is kept below `B_threshold` except for the first iteration. When the WS decreases to 63ms, the value of V falls to 4.62%, slightly below `BEST_V` (5%), and then the WR algorithm returns the optimized WS of 63ms for TA. For RRvar, when WS falls to 63ms (during the period between the 50^{th} second and the 60^{th} second), the storage bandwidth variability V has actually increased from 9.45% to 15.13% in the sixth iteration. This implies an overshoot for WR, forcing the algorithm to continue its search by backtracking slightly (with a WS larger than 63ms) until the difference between `ws_min` and `ws_max` falls below `UN_WIN` (set at 32). In this case, WR finds a suboptimal WS of 78ms for the tuning action, approximately and nearly achieving the goal. For SWvar, the absolute storage bandwidth error β can be observed to rise quickly from 7% to 14.91% in the fourth iteration, which implies an overshoot in β and forces the algorithm to continue its search by backtracking for a larger WS. As a result, the WS of 171ms is decided for SWvar. The number of iterations for SRvar, RRvar, and SWvar are all 7. This implies that 7 * `T_ps` is required for WR to finish the process of time window revision. Larger `T_ps` means a higher precision in SLO enforcement but a longer time of running WR. So, we can reduce the running time of WR by shrinking the search scope of WS (e.g., narrowing from (32, 1,024) to (32, 256)). Moreover, higher `U` can be used to alleviate the impact of search inaccuracy of WR due to smaller `T_ps`.

5.1.2. Performance Drift and TR Enforcement. Dynamic IO characteristics and the fluctuation in allocated resources to a VM can affect the error between the measured bandwidth and the target value specified in SLO, a phenomenon we refer to as *performance drift*. The TR mechanism is proposed to address this problem by adjusting b_{TARGET} timely based on the change in the storage bandwidth error `B_ps` for the TA algorithm. To assess the effectiveness of TR in alleviating performance drift, we deploy the SRvar, SWvar, and RRvar workloads in three coscheduled VMs, respectively, where the target bandwidths are 20MB/s, 15MB/s, and 3MB/s respectively. As shown in Figure 8, `adjust_factor`, which determines the amount of correction applied to b_{TARGET},

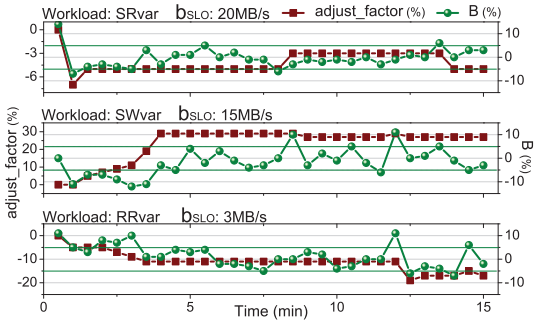

Fig. 8. Target revision for the three coscheduled VMs deployed with the SRvar, SWvar, and RRvar workloads, respectively.

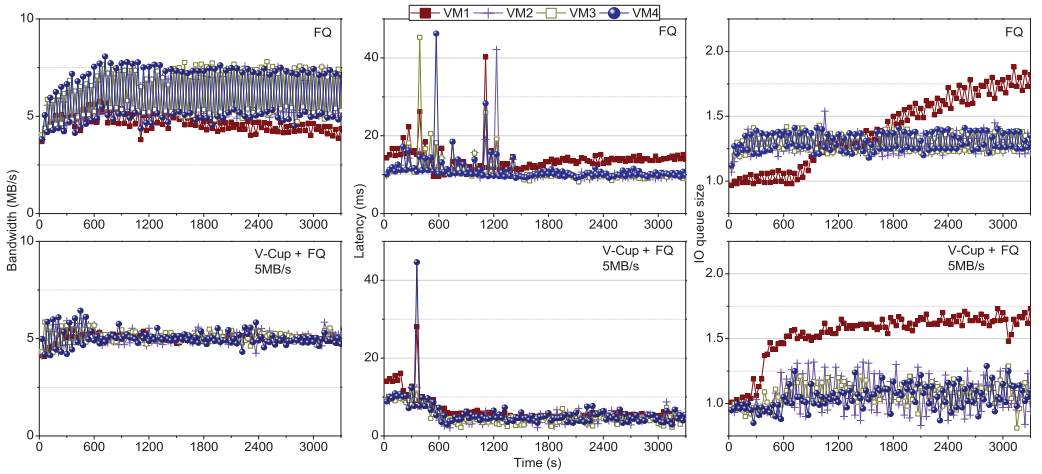

Fig. 9. A comparison between FQ and V-Cup+FQ in bandwidth, latency, and IO queue size of the four VMs deployed with YCSB that are concurrently running for 55 minutes. VM1 has 5,300,000 objects of size 10KB each and each of VM2–VM4 has 10,000 objects of size 160KB each.

can be coordinated adapting to the change in the storage bandwidth error B. As a result, for the concurrently running workloads with distinct IO characteristics, TR can effectively control the storage bandwidth error to be within the range $[-5\%, 5\%]$.

5.2. Workload of YCSB

We use Yahoo Cloud Serving Benchmark (YCSB) [Cooper et al. 2010] to generate a Zipfian distributed key-value request workload that includes 50% reads and 50% updates. Specifically, each VM runs an instance of YCSB based on the MongoDB [MongoDB 2015] to access a dataset stored on its VM disk. To evaluate the effectiveness of near-precise SLO enforcement of V-Cup for the VMs with heterogeneous datasets (i.e., different numbers of objects and different object sizes), we established two types of datasets for four coscheduled VMs. VM1 has a dataset of 5,300,000 objects of size 10KB each, while there are 100,000 objects of size 160KB each on the VM disks of the other three VMs (i.e., VM2–VM4). The layout of the dataset for VM1 is different from that of the other three VMs.

To compare with FQ under the same fairness strategy, we customize the same target bandwidth in SLO for each VM (i.e., 5MB/s). As shown in Figure 9, the bandwidth

Table II. The Measured Average Bandwidth and β in V-Cup+FQ and a Comparison in V Value between FQ and V-Cup+FQ for the Four VMs Over 55 Minutes

VMs	Bandwidth (MB/s)		β (%)	V (%)	
	Target	Measured		FQ	V-Cup+FQ
VM1	5	5.02	2.97	13.88	11.26
VM2	5	5.08	5.32	31.98	15.77
VM3	5	5.08	4.36	31.60	15.23
VM4	5	5.04	5.23	32.24	16.43

Table III. Parameters for Filebench Workloads

Services	Files	Threads	File size	IO size
Web Server	50,000	100	16KB	512KB
Mail Server	50,000	16	8–16KB	16KB
File Server	50,000	50	128KB	16KB–1MB
Video Server	—	48	1GB	256KB–1MB

fluctuation for VM2–VM4 in FQ is much wider than that in V-Cup. More importantly, for FQ, the bandwidth of VM1 is smaller than that of any other VM, while its latency is larger than that of any other VM. In addition, a widening gap in latency between VM1 and any other VM is observed in FQ. This is because the IO queue size of VM1 with outstanding requests at the disk array increases steadily with time. Obviously, unfair and disadvantageous IO competition for VM1, due to its different data layout from the other VMs, makes its IO performance unstable, indicating that FQ cannot provide a fair performance for VMs if their data layout is different.

In contrast, as shown in Figure 9, V-Cup+FQ is able to keep the bandwidth for VM1–VM4 relatively stable and at or near their target bandwidth of 5MB/s. More specifically, as listed in Table II, the β values for VM1–VM4 in V-Cup+FQ are all smaller than 6%, while the V values for VM1–VM4 in V-Cup+FQ are much smaller than those in FQ. This indicates that V-Cup+FQ can provide a near-precise SLO enforcement for the storage bandwidth of VM1–VM4 despite their different data layout. In addition, the latency measures for VM1–VM4 are observed to converge to a value smaller than the latency values in FQ and remain relatively stable. This is because V-Cup+FQ can judiciously maintain a proper IO queue size and request size that are just enough for preserving the target bandwidth. As shown in Figure 9, the average IO queue size for VM2–VM4 in V-Cup+FQ is smaller than that in FQ. However, the specific data-layout difference between VM1 and VM2–VM4 means that the IOs issued by VM1 are competing unfairly and disadvantageously with the IOs issued from the other VMs. But with the adjustment by V-Cup, the IO queue size of VM1, with its outstanding requests at the disk array, is increased to be larger than those of the other VMs, which enables the throughput of VM1 to be increased. This effectively compensates for VM1's disadvantage in the IO contention with the other VMs.

5.3. VM-Oriented Customizability of SLO

V-Cup provides customizable SLO with its near-precise enforcement for common tenants in an intuitive way, that is, simply informing or expressing the desired storage bandwidth in SLO. Based on the design of V-Cup, the tenant can also customize when and how much the storage bandwidth of VMs may be changed at runtime. The following experiments are designed and conducted to demonstrate this customizability of V-Cup and its feasibility and effectiveness.

First, we deploy different types of servers, including web server, mail server, file server, and video server simulated by Filebench [Mcdougall 2015] (the specific parameters of each workload are listed in Table III), each on a different VM, to test how they

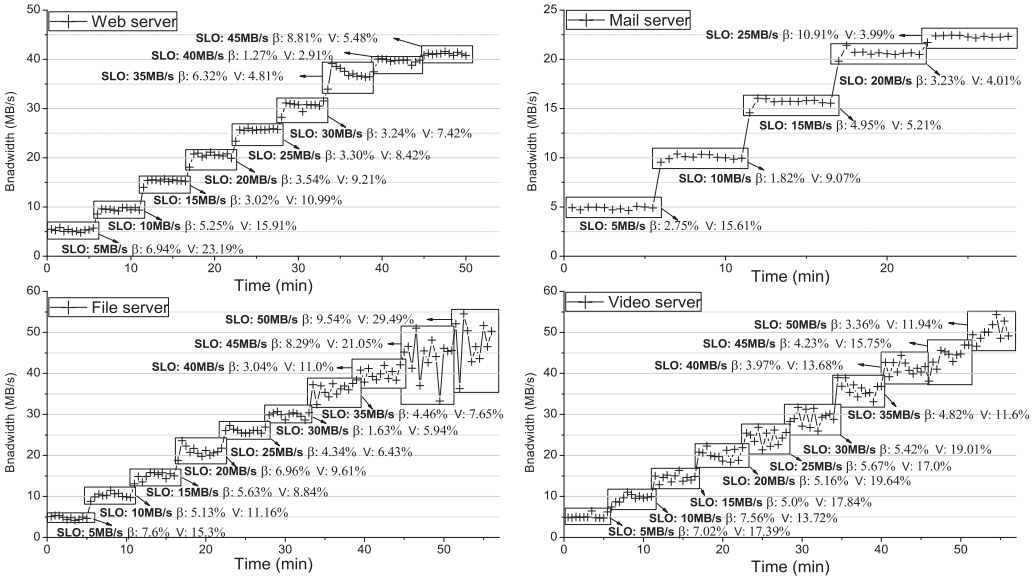

Fig. 10. The storage bandwidth, absolute storage bandwidth error (β), and storage bandwidth variability (V) as a function of the controlled linear increase of SLO for the storage bandwidth of different servers.

each perform in V-Cup. Specifically, we let the target storage bandwidth of a VM in the SLO rise in a controlled fashion, starting from 5MB/s, by an increments of 5MB/s every 5 minutes and observe the measures of the SLO enforcement precision (i.e., β and V) for each of these 5-minute periods. As shown in Figure 10, for the web server, when the SLO is increased from 5MB/s to 45MB/s, the bandwidth averaged over each 5-minute period, or average bandwidth for simplicity, can be adjusted from 5.35MB/s to 41.04MB/s. For the mail server, the average bandwidth can be increased continuously from 4.86MB/s to 22.3MB/s as SLO increases from 5MB/s to 25MB/s. And the average bandwidths for the file server and the video server can also be seen to increase from 4.62MB/s and 5.36MB/s to 45.23MB/s and 48.32MB/s, respectively, as SLO increases from 5MB/s to 50MB/s. It is also observed that the absolute storage bandwidth error (β) over each 5-minute period can be maintained below 10% for almost all the cases except for the case of the highest SLO (i.e., 25MB/s) for the mail server. In this exception case, β is abruptly increased from 3.23% to 10.91% when SLO is changed from 20MB/s to 25MB/s. This is because the measured maximum bandwidth for the mail server is about 22MB/s, which should be the upper bound of SLO enforcement. So, in a real cloud environment, V-Cup should feedback a bandwidth range to the tenant after carrying out a test running, which is discussed in Section 3.3. In addition, storage bandwidth variability (V) can be maintained below 30% for all the cases, where for 91.2% of the cases V can be kept below 20%.

Second, to verify the near-precise SLO enforcement supported by V-Cup in server consolidation, we deploy the web server, mail server, file server, and video server simulated by Filebench [Mcdougall 2015] in four coscheduled VMs (VM1, VM2, VM3, and VM4), respectively. The SLOs for these four VMs are set at 40MB/s, 20MB/s, 10MB/s, and 20MB/s, respectively. And then, we run them concurrently in V-Cup and FQ, respectively, for 20 minutes. As listed in Table IV, the storage bandwidth stability V value for the four VMs in V-Cup is smaller than that in FQ. In addition, the absolute storage bandwidth error β value for the four VMs in V-Cup is also maintained at a lower level, with the maximum value being 8% for VM4.

Table IV. The Average Bandwidth and β for Four Concurrent Workloads (Web Server, Mail Server, File Server, Video Server) in V-Cup and a Comparison in V between FQ and V-Cup of the Four VMs over 20 Minutes

VMs	V-Cup Bandwidth (MB/s)		β (%)	V (%)	
	Target	Measured		FQ	V-Cup
VM1	40	40.6	2.6	10.0	6.02
VM2	20	21.02	5.11	6.28	3.55
VM3	10	9.96	7.05	52.19	22.45
VM4	20	18.74	8.0	103.89	29.74

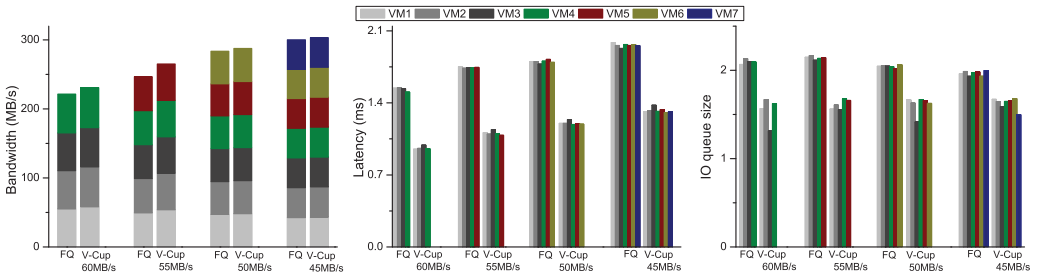

Fig. 11. A comparison between FQ and V-Cup in total bandwidth, latency, and IO queue size averaged over 10 minutes when the number of coscheduled VMs (running SR) increases from four to seven.

5.4. Bandwidth Guarantee with Fixed Storage Capacity

In this section, our evaluation examines a key question: What is the maximum storage bandwidth that can be provided by V-Cup with a *fixed storage capacity*, which refers to the maximum capacity supported by the given devices (e.g., a given array of disks), shared by different numbers of VMs?

We experimented with the workloads of SR (elaborated in Section 5.1) in VMs to simulate streaming media services sharing a fixed storage capacity. To compare with FQ under the same fairness strategy, we customize the same target bandwidth for each VM. To reach the maximum total bandwidth provided by V-Cup, we increase the target continuously until the total bandwidth cannot be further increased. In this scenario, the sum of the target bandwidths for all the VMs can be larger than the total bandwidth averaged over a longer period of time (e.g., 10 minutes). If this happens, we consider the storage capacity inadequate for meeting the objectives of the near-precise SLO enforcement.

We set 60MB/s, 55MB/s, 50MB/s, and 45MB/s as the target bandwidths per VM for the scenarios of four VMs, five VMs, six VMs, and seven VMs, respectively, when running the SR workload on VMs. It is observed from Figure 11 that the total bandwidth for FQ is slightly smaller than that for V-Cup. More specifically, the maximum total bandwidths provided by V-Cup are 230.78MB/s, 265.25MB/s, 287.71MB/s, and 303.49MB/s, in contrast to 221.64MB/s, 246.24MB/s, 283.81MB/s, and 299.48MB/s by FQ, as the number of coscheduled VMs increases from four to seven, respectively. As listed in Tables V and VI, for V-Cup, the absolute storage bandwidth error (β) over 10 minutes can be maintained at a low level (smaller than 6%), while the storage bandwidth variability (V) value for each VM is kept below 10.03%.

Another significant advantage of V-Cup over FQ is its smaller IO latency than FQ for VMs. As shown in Figure 11, the average latency during the 10 minutes for each VM in V-Cup is smaller than that in FQ when the number of coscheduled VMs increased from four to seven. This is because V-Cup can judiciously maintain a proper IO queue size and request size that are just enough for preserving the target bandwidth. As shown

Table V. The β Values Over 10 Minutes for VMs in V-Cup when the Number of Coscheduled VMs (Running SR) Increases from Four to Seven

VMs	Target (MB/s)	Absolute storage bandwidth error β (%)						
		vm1	vm2	vm3	vm4	vm5	vm6	vm7
4	60	3.16	3.73	5.17	3.38	—	—	—
5	55	2.69	3.37	4.25	4.04	3.77	—	—
6	50	3.84	4.22	4.32	4.19	4.02	4	—
7	45	3.96	2.99	3.98	3.29	3.76	3.69	4.01

Table VI. The V Values Over 10 Minutes for VMs in V-Cup when the Number of Coscheduled VMs (Running SR) Increases from Four to Seven

VMs	Target (MB/s)	Storage bandwidth variability V (%)						
		vm1	vm2	vm3	vm4	vm5	vm6	vm7
4	60	9.58	9.51	8.81	9.75	—	—	—
5	55	9.1	9.54	9.13	9.53	10.03	—	—
6	50	8.93	8.68	8.09	8.53	8.28	8.59	—
7	45	8.05	8.48	8.03	7.97	7.83	8.3	7.95

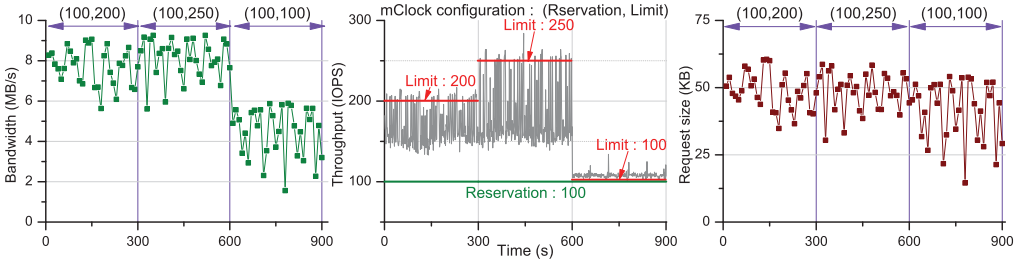

Fig. 12. The throughput, request size, and bandwidth under the max-min control of mClock.

in Figure 11, the average IO queue size and latency during the 10 minutes for each VM in V-Cup are smaller than those in FQ when the number of coscheduled VMs increased from four to seven.

5.5. V-Cup and mClock

We conduct this experiment to verify V-Cup's ability to enhance max-min control of resource management for coscheduled VMs. Specifically, we adopt mClock [Gulati et al. 2010] as the IO scheduler based on which V-Cup works as a request size control layer for each VM, exporting bandwidth customization interface for users. Since it is hard for the user to determine the weights of VMs sharing the resources subject to the constraint that each VM receives at least its reservation, we assume that each coscheduled VM has the same weight in the experiment of the mClock-based V-Cup (V-Cup+mClock). Moreover, we represent the reservation and limit for each VM in the mClock setting as (reservation, limit), which means that the VM should receive a throughput whose value lies in the range (reservation, limit).

As shown in Figure 12, with different mClock settings of (100, 200), (100, 250), and (100, 100) for VM1, respectively, the max-min throughput of VM1 can be well coordinated according to the corresponding settings. However, the variation in VM1's bandwidth is unpredictable and uncontrollable under mClock because the variability of request size affects the fluctuation of bandwidth even when the reservation and limit are set at the same value of 100 IOPS. Hence, we use V-Cup based on the resource management of mClock for coscheduled VMs to provide near-precise SLO enforcement for VM storage bandwidth. And we only assume that mClock provides a coarse-grained

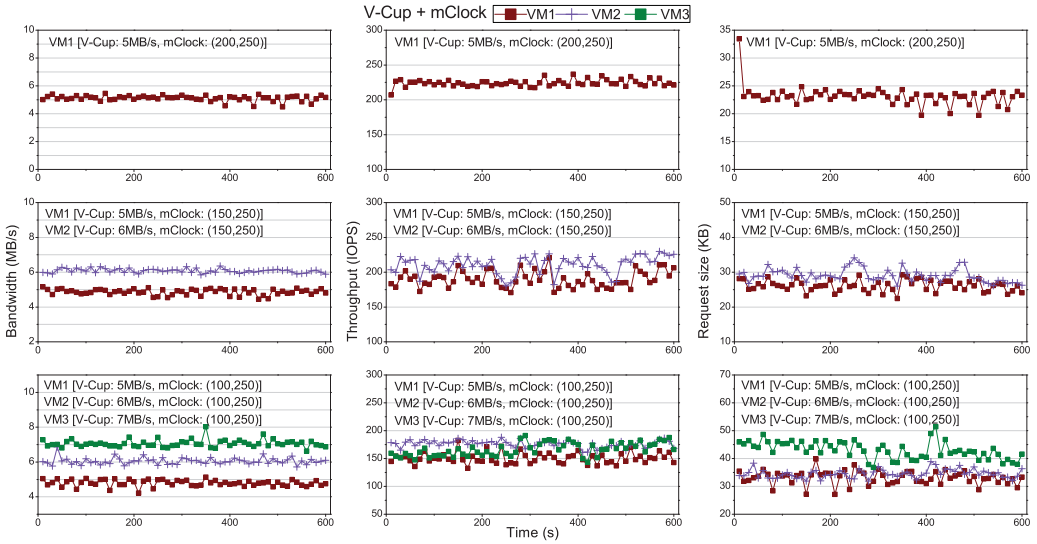

Fig. 13. The variation of bandwidth, throughput, and request size in the one-VM, two-VM, and three-VM configurations deployed with YCSB that are concurrently running for 10 minutes under the control of V-Cup+mClock. VM1 has 5,300,000 objects of size 10KB each and each of VM2–VM3 has 10,000 objects of size 160KB each.

Table VII. The Measured β and V for the One-VM, Two-VM, and Three-VM Configurations under the Control of V-Cup+mClock Averaged Over 10 Minutes

	β (%)			V (%)		
VMs	VM1	VM2	VM3	VM1	VM2	VM3
1*VM	3.06	—	—	9.43	—	—
2*VM	2.69	1.52	—	7.29	5.54	—
3*VM	4.14	1.45	1.12	9.26	6.45	7.00

throughput allocation across VMs because the strict reservation-and-limit settings may perplex novice users. Figure 13 shows the results with the mClock settings of (200, 250), (150, 250), and (100, 250) combined with the V-Cup settings of 5MB/s, 6MB/s, and 7MB/s for the one-VM, two-VM, and three-VM concurrently running configurations each for 10 minutes, respectively. It can be observed that V-Cup can converge the bandwidth of each VM on the target value in all settings by coordinating the request size and throughput in a fine-grained fashion. We can further verify the results of near-precise SLO enforcement in Table VII that shows very small values of β (i.e., all the β values are smaller than 5%) and V (i.e., all the V values are smaller than 10%) can be achieved by V-Cup+mClock. It is noted that V-Cup is compatible with and well complements mClock in that the specific reservation-and-limit setting will not affect the precision of SLO enforcement of V-Cup only if the desired throughput falls between the reservation and the limit set in mClock.

5.6. Request-Merge Limit and IO Throttling

Storage bandwidth is determined by two factors: IO throughput and request size. Thus, both stable IO throughput and stable request size are important to bandwidth SLO enforcement. Thus, it makes sense to investigate the mutual impact of request-merge limit adjustment and IO throttling under the V-Cup control. In this way, we can effectively verify the feasibility and robustness of V-Cup on storage bandwidth SLO enforcement by the cooperation between request-merge limit control and IO throttling.

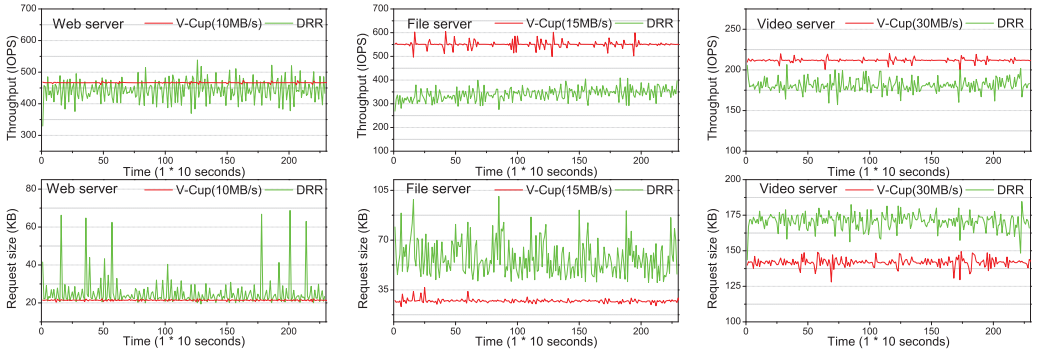

Fig. 14. The variation of throughput and request size for three concurrently running VMs deployed with web server, file server, and video server, respectively, under the cooperation between the request-merge limit control and IO throttling.

To this end, we implement a Proportional-Integral (PI) model, adopted from the classical control theory [Franklin et al. 1998; Hellerstein et al. 2004], in the IO throttling submodule of the V-Cup auto-tuner to control the IO dispatching for each individual VM. In doing so, we can achieve almost zero control error (i.e., the difference between the control target and the actual measured value) of IO throughput in theory [Hellerstein et al. 2004]. And then, we conduct an experiment by running three VMs that concurrently access a 16-disk RAID-0 disk group. We deploy a web server, a file server, and a video server simulated by Filebench [Mcdougall 2015] in these three VMs, respectively. The storage bandwidth SLOs for these VMs are 10MB/s, 15MB/s, and 30MB/s, respectively, while their IO throughput targets are 466 IOPS, 550 IOPS and 211 IOPS, respectively. We choose Deficient Round Robin (DRR) [Shreedhar and Varghese 1987] algorithm as the baseline, which is adopted by the state of the art Pisces [Shue et al. 2012] to guarantee the accuracy of SLO enforcement. As shown in Figure 14, the interference between the request-merge limit control and IO throttling is almost negligible as evidenced by the small variation of throughput and request size for these VMs. This is largely because V-Cup will allocate a request-merge limit submodule and an IO throttling submodule for each running VM so as to minimize the mutual interferences among their feedback control loops. Thus, each submodule can implement single-purpose optimization for its own control target without additional impacts from other submodules, which obviously helps achieve stable and accurate control. Specifically, in contrast to DRR, V-Cup can effectively achieve more stable IO throughput for all three VMs with the averages of 466.75 IOPS, 550.77 IOPS, and 211.81 IOPS, very close to their targets. Simultaneously, V-Cup also outperforms DRR in minimizing request size variability. As a result, the values of absolute storage bandwidth error for these VMs under the V-Cup control are 0.30%, 0.58%, and 0.62%, while those obtained under DRR are 1.12%, 1.24%, and 0.70%, which means V-Cup can achieve more accurate storage bandwidth SLO enforcement for all the concurrently running VMs based on a satisfactory timeliness of IO control.

6. RELATED WORKS

6.1. Virtualization for Data Centers

Modern data centers widely adopt full or paravirtualization (Barham et al. [2003], VMware Infrastructure [2016], and Kivity et al. [2007]) or OS-level virtualization (LXC [2016] and Soltesz et al. [2007]) to support multiple applications running on their individual VMs or containers, respectively, deployed on the same host. Full or

paravirtualization provides a guest OS (and thus an isolated IO stack) for each VM, while OS-level virtualization permits the consolidated containers to adopt a shared IO stack, however, resulting in severe IO interference [Kang et al. 2014].

Many mainstream virtualization techniques [Sugerman et al. 2001; Fraser et al. 2004; Russell 2008] support VM-level device emulation, which can virtualize physical storage devices shared by coscheduled VMs or containers as individual virtual disks. And each virtual disk can be flexibly attached to a specific VM or container with near-native performance [Mansley et al. 2007]. However, the performance isolation among VMs or containers poses a great challenge. The recent research of MultiLanes [Kang et al. 2014] introduces an isolated IO stack for each container above the container-based OS layer for improved performance isolation.

In contrast, our V-Cup system is implemented in the disk IO scheduler at the block layer of the Linux OS, which is not tied to a specific type of virtualization implementation (e.g., Xen or KVM). Furthermore, V-Cup is orthogonal and complementary to the preceding virtualization solutions.

6.2. QoS-Based Storage Sharing

The existing approaches to QoS-Based storage sharing can be broadly divided into three categories according to the IO control technique adopted. The first is the class of approaches that adopt IO throttling techniques, such as time-stamp-based IO rate control [Bennett and Zhang 1996; Golestani 1994; Goyal et al. 1997; Suri et al. 1997], leaky bucket technique [Chambliss et al. 2003], DRR [Shreedhar and Varghese 1987], and control theoretic methods [Franklin et al. 1998; Hellerstein et al. 2004], etc., to guarantee throughput or latency QoS for consolidated workloads, such as mClock [Gulati et al. 2010], Pisces [Shue et al. 2012], Aqua [Wu and Brandt 2006], SARC [Zhang et al. 2006], etc. mClock, based on time-stamp-based IO rate control, is able to control the throughput variability of VMs by appropriately adding the limits, shares, and reservations to the hypervisor's fair scheduler. Fahrrad [Povzner et al. 2008] can translate throughput target into disk time utilization that guides IO throttling by time-stamp-based IO rate control. Pisces enables the sharing of key-value storage with max-min fairness on a per-tenant basis, which adopts DRR [Shreedhar and Varghese 1987] to reduce the performance deviation from the SLO target. Triage [Karlsson et al. 2005] can provide throughput and latency differentiation for each workload by adaptive controller with integral control, which is a type of control model emerged from the classic control theory [Franklin et al. 1998; Hellerstein et al. 2004]. In addition, many existing approaches, such as Aqua [Wu and Brandt 2006] and SARC [Zhang et al. 2006], adopt many variants of the leaky bucket model to guide IO throttling for coscheduled IO workloads so as to meet the throughput target of each workload.

The second is the class of solutions that use IO queue depth control to improve IO isolation or fair share across clients at the cost of IO concurrency, such as SRP [Gulati et al. 2012], IOFlow [Thereska et al. 2013], PARDA [Gulati et al. 2009], SFQ(D) [Jin et al. 2004], etc. SRP [Gulati et al. 2012], based on mClock [Gulati et al. 2010], supports max-min fairness and proportional shares at the VM pool level by dynamically allocating the IO queue depth that a host can keep outstanding at the shared storage device. IOFlow [Thereska et al. 2013] aims to enforce end-to-end throughput guarantee policies for each IO flow under virtualized data centers with multilayer IO stack by bounding IO queue depth and dynamically throttling IO rate at each layer. PARDA [Gulati et al. 2009] aims to improve performance isolation and throughput fairness among hosts each of which runs one or more VMs by adjusting per-host IO queue depth. SFQ(D) [Jin et al. 2004] provides a desired level of throughput fairness across concurrently running IO workloads by IO queue depth adjustment at the interposed scheduler, which inherently strikes a trade-off between the maximum performance

deviation and the IO concurrency of the underlying storage server. In addition, some approaches such as Façade [Lumb et al. 2003], Avatar [Zhang et al. 2006], SMART [Nieh and Lam 2003], and BVT [Duda and Cheriton 1999], adopt IO queue depth control to enforce IO latency targets for coscheduled IO workloads with a compromised level of IO concurrency.

The third is the class of solutions that support request size control, such as Cake [Wang et al. 2012] and IOFlow [Thereska et al. 2013]. The performance penalty incurred by a request with a large IO size can very likely lead to SLO violations [Wang et al. 2012]. Cake [Wang et al. 2012] splits a large request (e.g., 512KB) into several smaller requests with an appropriate IO size so as to reduce the queue time. And IOFlow [Thereska et al. 2013] adopts a similar scheme to alleviate the probable performance penalty.

However, the preceding solutions do not address the impact of request size variability on the performance variability that can very likely result in SLO violations, especially for storage bandwidth compounded by two factors: IO throughput and request size. More importantly, the association and the interplay between the timeliness of IO control and the precision of SLO enforcement have not been exploited. V-Cup, by contrast, focuses on the performance variability and the timeliness of IO control for the metric of storage bandwidth by combining the techniques of request size variability control and IO throttling in a VM-oriented end-to-end fashion. In this way, V-Cup is able to support near-precise SLO enforcement under consolidated VM environment. With the support of the easy-to-specify and technology-agnostic SLO by providing a particular value for the bandwidth of a VM, tenants can customize their desired performance more intuitively and easily.

7. CONCLUSION

In this article, we address the challenge of providing VM-oriented customizable SLO and its N-PSE for the storage bandwidth of VMs. We propose a framework V-Cup consisting of multiple auto-tuners. Each auto-tuner can carry out the fine-grained performance tuning for each VM with improved IO control timeliness that is critical to reducing the fluctuation of bandwidth and more rapidly coordinating bandwidth to adapt to variable IO characteristics. Our evaluation of V-Cup shows that it is able to provide the N-PSE for the storage bandwidth of VMs by maintaining a lower level of absolute storage bandwidth error and storage bandwidth variability. As future work, we are trying to realize the idea of N-PSE on customizable hybrid metrics for storage performance in cloud services.

REFERENCES

Amazon EC2. 2015. Amazon EC2 website. Retrieved from http://aws.amazon.com/ec2.

Amazon EC2 SLA. 2015. Amazon EC2 SLA. Retrieved from http://aws.amazon.com/cn/ec2-sla/.

J. Axboe. 2004. Linux block IO—Present and future. In *Proceedings of the Ottawa Linux Symposium.*

P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. 2003. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*.

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

R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schioberg. 2007. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the International Conference on Virtual Execution Environments (VEE)*.

D. D. Chambliss, G. A. Alvarez, P. Pandey, D. Jadav, J. Xu, R. Menon, and T. P. Lee. 2003. Performance virtualization for large-scale storage systems. In *Proceedings of the 22nd International Symposium on Reliable Distributed Systems (SRDS)*.

B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*.

A. Dan and D. Sitaram. 1996. A generalized interval caching policy for mixed interactive and long video environments. In *Proceedings of Multimedia Computing and Networking Conference*.

K. J. Duda and D. R. Cheriton. 1999. Borrowed-virtual-time (BVT) scheduling: Supporting latency-sensitive threads in a general-purpose scheduler. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*.

FIO. 2015. FIO. Retrieved from http://freecode.com/projects/fio.

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

K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. 2004. Safe hardware access with the Xen virtual machine monitor. In *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On Demand IT InfraStructure (OASIS)*.

S. Golestani. 1994. A self-clocked fair queueing scheme for broadband applications. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*.

Google Compute Engine. 2015. Google Compute Engine. Retrieved from https://cloud.google.com/products/compute-engine/.

P. Goyal, H. M. Vin, and H. Cheng. 1997. Start-time fair queuing: A scheduling algorithm for integrated services packet switching networks. *IEEE/ACM Transactions on Networking* 55 (1997), 690–704.

A. Gulati, I. Ahmad, and C. A. Waldspurger. 2009. PARDA: Proportional allocation of resources for distributed storage access. In *Proccedings of the Conference on File and Storage Technologies (FAST)*.

A. Gulati, A. Merchant, and P. J. Varman. 2010. mClock: Handling throughput variability for hypervisor IO scheduling. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*.

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

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

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

J. Kang, B. Zhang, T. Wo, C. Hu, and J. Huai. 2014. MultiLanes: Providing virtualized storage for OS-level virtualization on many cores. In *Proccedings of the Conference on File and Storage Technologies (FAST)*.

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

J. Kim, D. Lee, and S. H. Noh. 2015. Towards SLO complying SSDs through OPS isolation. In *Proceedings of the Conference on File and Storage Technologies (FAST)*.

A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. 2007. kvm: The Linux virtual machine monitor. In *Proceedings of the 2007 Linux Symposium*.

J. Liu, S. G. Rao, H. Zhang, and B. Li. 2008. Opportunities and challenges of peer-to-peer internet video broadcast. *Proceedings of the IEEE* 96, 1 (2008), 11–24.

Z. Lu, J. Wu, Y. Huang, L. Chen, and D. Deng. 2012. CPDID: A novel CDN-P2P dynamic interactive delivery scheme for live streaming. In *Proceedings of IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)*.

C. R. Lumb, A. Merchant, and G. A. Alvarez. 2003. Façade: Virtual storage devices with performance guarantees. In *Proccedings of the Conference on File and Storage Technologies (FAST)*.

LXC. 2016. LXC. Retrieved from https://linuxcontainers.org/lxc/.

K. Mansley, G. Law, D. Riddoch, G. Barzini, N. Turton, and S. Pope. 2007. Getting 10 Gb/s from Xen: Safe and fast device access from unprivileged domains. In *Proceedings of the 2007 Conference on Parallel Processing*. 224–233.

A. Mashtizadeh, E. Celebi, T. Garfinkel, and M. Cai. 2011. The design and evolution of live storage migration in VMware ESX. In *Proceedings of the USENIX Annual Technical Conference (ATC)*.

R. Mcdougall. 2015. A prototype model-based workload for file systems, work in progress. Retrieved from http://solarisinternals.com/si/tools/filebench/filebench nasconf.pdf.

MongoDB. 2015. MongoDB. Retrieved from http://www.mongodb.org/.

R. Nathuji, A. Kansal, and A. Ghaffarkhah. 2010. Q-clouds: Managing performance interference effects for QoS-aware clouds. In *Proceedings of the 3rd European Conference on Computer Systems (EuroSys)*.

J. Nieh and M. S. Lam. 2003. A SMART scheduler for multimedia applications. *ACM Transactions on Computer Systems (TOCS)* 21, 2 (2003), 117–163.

A. Povzner, T. Kaldewey, S.Brandt, R. Golding, T. M. Wong, and C. Maltzahn. 2008. Efficient guaranteed disk request scheduling with Fahrrad. In *Proceedings of the 3rd European Conference on Computer Systems (EuroSys)*.

R. Russell. 2008. virtio: Towards a de-facto standard for virtual I/O devices. *SIGOPS Operating Systems Review* 42, 5 (2008), 95–103.

M. Ryu and U. Ramachandran. 2013. FlashStream: A multi-tiered storage architecture for adaptive HTTP streaming. In *Proceedings of the 21st ACM International Conference on Multimedia*. 313–322.

M. Shreedhar and G. Varghese. 1987. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on Networking* 4, 3 (1987), 375–385.

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

S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. 2007. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *Proceedings of the 3rd European Conference on Computer Systems (EuroSys)*.

J. Sugerman, G. Venkitachalam, and B. H. Lim. 2001. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *Proceedings of the USENIX Annual Technical Conference (ATC)*.

S. Suri, G. Varghese, and G. Chandramenon. 1997. Leap forward virtual clock: A new fair queueing scheme with guaranteed delay and throughput fairness. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*.

E. Thereska, H. Ballani, G. O'Shea, T. Karagiannis, A. Rowstron, T. Talpey, R. Black, and T. Zhu. 2013. IOFlow: A software-defined storage architecture. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*.

VMware Infrastructure. 2016. VMware, Inc. Introduction to VMware Infrastructure. Retrieved from http://www.vmware.com/support/pubs/.

M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger. 2007. Argon: Performance insulation for shared storage servers. In *Proccedings of the Conference on File and Storage Technologies (FAST)*.

A. Wang, S. Venkataraman, S. Alspaugh, R. Katz, and I. Stoica. 2012. Cake: Enabling high-level SLOs on shared storage systems. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*.

Windows Azure. 2015. Windows Azure. Retrieved from http://www.windowsazure.com/.

J. C. Wu and S. A. Brandt. 2006. 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)*.

S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao. 2009. WorkOut: I/O workload outsourcing for boosting RAID reconstruction performance. In *Proccedings of the Conference on File and Storage Technologies (FAST)*.

S. Wu, H. Jiang, and B. Mao. 2012. IDO: Intelligent data outsourcing with improved RAID reconstruction performance in large-scale data centers. In *Proceedings of the USENIX Large Installation System Administration (LISA)*.

Y. Wu, C. Wu, B. Li, X. Qiu, and F. C. Lau. 2011. CloudMedia: When cloud on demand meets video on demand. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*.

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

Received July 2015; revised July 2016; accepted September 2016