virtustream

Whitepaper

IOTune

Adaptive I/O Tuning for Elastic SSD Volume Performance

Abstract

We present IOTune, an adaptive I/O tuning framework for elastic SSD volume performance in multi-tenancy environments. IOTune exploits IOPS statistical multiplexing of co-located volumes and the *in situ* IOPS adjustment supported by the virtualization layer to break through the conflicts between IOPS static reservation and right-provisioning. Trace-replay based evaluations demonstrate that volumes managed by IOTune adapt to workload fluctuations. Compared with existing static IOPS provisioning, IOTune enables a volume to achieve same QoS with near 50% less resource reservation, which halves volume bills. IOTune also reduces I/O tail latencies by one to two orders of magnitude.

1 Introduction

In addition to being used as cloud caches [1, 2, 3, 4, 5, 6], SSD volumes are becoming a popular type of persistent data storage in data centers [7, 8, 9, 10]. IOPS is a main QoS metric of SSD volumes. IOPS is charged either independently or together with volume capacity. Tenants may explicitly [8] or implicitly [9, 10] specify IOPS of SSD volumes, which provides twofold benefits. First, provisioned IOPS is a service level agreement (SLA), which indicates what storage QoS tenants can expect, thus making volume performance predictable. Second, provisioned IOPS throttles resource consumption of a volume and prevents it from becoming too aggressive, facilitating performance isolation and fair resource allocation. Current SSD volumes in public clouds adopt static IOPS provisioning. Tenants specify IOPS values upon volume creations and the values cannot be changed there-after. Production workloads are bursty and fluctuant [11, 12, 13, 14]. Peak I/O rates are usually more than one order of magnitude higher than average rates. Thus, static IOPS provisioning places tenants in a dilemma. Under-provisioning fails to support peak loads, resulting in significant I/O tail latencies. Over-provisioning may meet peak requirements but wastes a lot of reservation in low-load periods, resulting in exorbitant costs.

Platform	SSD volume (128GB) IOPS features		
	IOPS	Configurable	Adjustable after creation
GCE [9]	3840	No	No
EC2 [8]	100-6400	Yes	No
Azure [10]	500	No	No

Table 1: SSD volume IOPS features supported by IaaS platforms including Google Compute Engine (GCE), Amazon EC2, and Microsoft Azure

Virtualization platforms provide a unique chance to achieve volume IOPS right-provisioning. In virtualization environments, IOPS limits of volumes are controlled in software stacks. The limits are software-defined, thus can be adjusted *in situ*. So instead of off-loading [12] I/O requests during bursts, IOPS limits of IaaS volumes can be promoted *in situ* to support I/O peaks.

The multi-level IOPS provisioning and metering of IOTune is a practical model for SSD volume management in multi-tenancy environments.

2 Motivation and Problem Statement

2.1 Conflicts between IOPS static reservation and right-provisioning

Table 1 lists the SSD volume IOPS features of mainstream IaaS platforms. EC2 allows tenants to specify the IOPS of a volume at its creation time. GCE allocates IOPS based on a predefined IOPS to GB ratio, which is currently 30. Azure only provides premium storage disks in three sizes: 128, 512, and 1024GB with predefined IOPS of 500, 2300, and 5000. None of the them enable adjusting the IOPS of a volume after its creation.

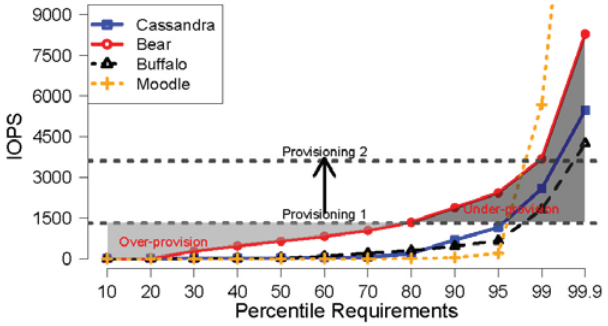


Figure 1: IOPS requirements of real workloads. Peak loads are much higher than the average.

Workload bursts are a real problem for IOPS static reservation of storage volumes. To demonstrate the problem, we analyze the IOPS requirements of real workloads using four traces: Bear, Buffalo, Moodle [15], and Cassandra. Cassandra traces contain I/O statistics of three production Cassandra VMs on our public cloud platforms. The percentile IOPS requirements are shown in **Figure 1**. One common characteristic of these workloads is that the volumes have low or moderate IOPS requirements more than 70% of the time. However, the tail IOPS requirements exponentially hike. Our analysis on Bear trace shows that top 30% peak periods contribute about 70% of the total I/O requests.

Assuming we need to create a provisioned IOPS SSD volume for Bear workloads and we would expect that 80% of the time the IOPS requirement can be satisfied. We have to provision the volume IOPS as provisioning 1, which is 1300, in the figure. This is a moderate provisioning. However, in the other 20% of the time the IOPS is under -provisioned and workloads will notice long response time. As an alternative, the IOPS can be aggressively set as provisioning 2, which is 2400 and satisfies requirements 95% of the time. However, the resource over-provisioning will cause serious waste during the low-load periods. In general, IOPS static reservation conflicts with workload fluctuations and rarely achieves resource right-provisioning.

2.2 IOPS statistical multiplexing of co-locating volumes

SRCMap [13] recognized significant variability in I/O intensity on storage volumes. Everest [12] validated the chance of statistical multiplexing of storage resources in production environments. We briefly demonstrate the chance of IOPS statistical multiplexing of co-located volumes. We adopt the approach of concurrently replaying six one-hour trace episodes[1] on six different SSD volumes. **Table 2** lists the percentile IOPS of the volumes and their statistical aggregates. The aggregate peak I/O rate is obviously lower than the sum of each individual peak I/O rate. For example, the sum of the 95th percentile IOPS of all episodes is 11355 while the 95th percentile aggregate IOPS is only 7966, which is 30% less than the sum due to stagger I/O peaks of volumes. Assuming the six volumes are all provisioned with IOPS of their 90th percentile arrival rates, the total IOPS reservation will be 8042, which can satisfy the 95th percentile aggregate IOPS requirement if the IOPS reservations of all volumes are multiplexed.

Volume	IOPS				
	Average	90%	95%	99%	99.9%
1	906	1877	3255	4026	5592
2	632	1626	2289	4433	6976
3	338	1084	1412	2050	3271
4	362	1077	1439	2192	2739
5	396	1257	1570	3262	6940
6	347	1121	1390	2024	5133
Aggregate	2982	6793	7966	10387	13469

Table 2: IOPS distributions of six volumes and their aggregate distribution. Each volume backs a one-hour episode of the Bear trace. Note that the aggregate percentile IOPS is not the algebraic sum of all episodes due to stagger I/O arrivals.

Then came our designing of IOTune, an adaptive I/O tuning framework for elastic SSD volume performance. IOTune exploits IOPS statistical multiplexing of co-located volumes and the *in situ* IOPS adjustment supported by the virtualization layer [16, 17, 18] to break through the conflicts between IOPS static reservation and right-provisioning.

3 IOTune

3.1 Design Overview

IOTune is designed as a plug-in component of IaaS platforms. It supports system-level configurations such as baseline IOPS per GB, aggregate performance limits of physical devices, utilization threshold of physical device for I/O tuning, unit IOPS price per GB minute, and so on. I/O tuning uses these parameters to manage storage volumes. Execution of IOTune includes two stages.

Stage 1: Volume Instantiation. A volume is a block device management unit. A volume including capacity and IOPS bills is a billing entity. Thus, a volume is a natural management unit of IOTune. Upon the request of creating a storage volume, the tenant specifies the size of the volume. Once the creation completes, IOTune instantiates the volume, pulls volume information, including storage path, size and creation time, calculates the multi-level IOPS metrics, and initializes the metering data.

Stage 2: Periodical I/O Tuning. Upon the instantiation of a volume, IOTune conducts an initial tuning, which sets a baseline IOPS limit on the volume. Then, IOTune periodically makes tuning decisions and conducts tuning. In default, we set the tuning interval as one second. IOTune checks the I/O monitoring data to judge the promotion or demotion of current IOPS level. Once tuning decisions are made, IOTune calls hypervisor API to make real-time resource adjustments.

3.2 IOTune Architecture

Figure 2 presents the IOTune architecture and its interaction with other system components. The key objective of IOTune is dynamically adjusting volume IOPS reservations based on their real requirements. To achieve this, IOTune requires a module for making tuning decisions and a module for tuning execution. The decision making depends on requirement prediction, which can be based on real-time metrics or historical statistics. For example, we compare the real-time IOPS consumption with the currently reserved IOPS of a volume to judge whether it is overloaded. The IOPS of a volume can be promoted only if underlying devices have spare capability. Device level I/O metrics are required to calculate the storage utilization. Metering policies are also required to make fair resource allocation. In general, the volume management function of IOTune is supported by I/O monitoring, tuning decision, tuning execution, and metering modules.

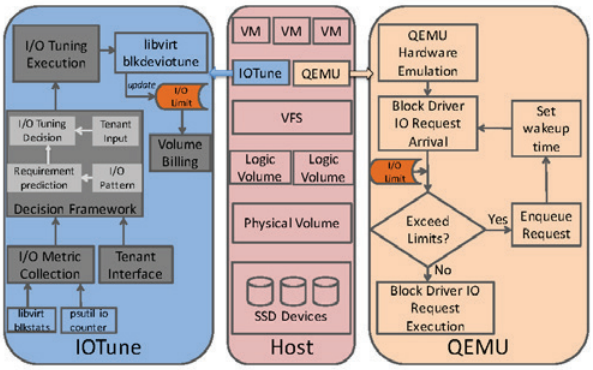


Figure 2: A systematic overview of IOTune. IOTune is a userspace program. Its I/O metric collection module interacts with physical and virtual block drivers via **psutil** and **libvirt** APIs. Its I/O tuning execution module uses **libvirt** to interact with QEMU for adaptive I/O tuning.

I/O Monitoring. IOTune collects two categories of I/O metrics. The first is the I/O metrics of SSD volumes. On our platforms, QEMU block driver layer provides interfaces to obtain the I/O statistics of each virtual disk. Considering QEMU block driver is also the location where I/O limits are enforced, IOTune employs **libvirt API** which calls **DomainBlockStats** interface of QEMU to obtain the IOPS and bandwidth data of target volumes. The other category of I/O metrics is about physical devices. The decision making framework of IOTune requires the physical device utilization as an input. I/O monitoring component of IOTune directly reads block device I/O counters via **psutil API** to calculate the IOPS and bandwidth usage of physical devices. These metrics are further used to calculate storage utilization.

I/O Tuning Decision Framework. IOTune decision framework decides when to adjust the IOPS of which volume by how much. First, when to adjust the IOPS of a volume. Temporal I/O patterns can be used to predict the volume IOPS requirements so as to promote the IOPS level of a volume before bursts arrive. For example, diurnal variations have been recognized in web server [11], Hotmail and Messenger [14] disk I/O loads, so even the wall-clock time can be a hint for predicting volume IOPS requirements. IOTune collects volume IOPS time series. Machine learning algorithms can be integrated into the IOTune decision framework to recognize I/O arrival patterns and predict I/O rates. For highly random workloads, IOTune decision framework supports real-time metric-based decision making. I/O Monitoring module provides real-time IOPS metrics, which can be used to predict the subsequent IOPS requirement of a volume. Currently, we empirically set the tuning period as one second. It guarantees the IOPS requirements of volume are adjusted in one second. This has proved to be very effective. Second, IOPS of which volume should be adjusted. Most of the time I/O requirements of all volumes can be satisfied because I/O peaks are staggered. But in extreme cases, peaks of volumes may overlap and surpass the storage capability. In such promotion contention scenarios, fairness and efficiency are two considerations for making tuning decisions. For efficiency, the promotions which will maximize storage utilization should be applied. For fairness, the promotion of volume which has the lowest current IOPS level should be prioritized. Current IOTune adopts the fairness first policy. For example, if volume A is at L_1, B is at L_0, and both desire a higher IOPS, IOTune prioritizes the promotion of volume B. Third, the IOPS of a volume is adjusted by how much. Considering the arriving and fading of I/O peaks are both immediate, IOTune employs multiplicative increase and decrease to adjust volume IOPS. For promotion the volume IOPS is doubled, and for demotion it is halved. This policy also simplifies the reservation metering.

For workloads that don't have a high requirement for quick response, IOPS promotion may not be attractive. Batch processing is an example. IOTune provides interfaces which enable tenants to disable the default automatic IOPS adjustment of specific volumes to avoid additional bills. The interfaces can also be used by tenants to customize their requirements such as specifying the IOPS of a volume during a predefined time period.

The following algorithms demonstrate the real-time metric based I/O tuning decision process.

```
Algorithm: IOPSTuneJudge(lv)
Input: logic volume instance lv.
Return: Tuning decision.
Procedure:
    for every one second do
        curiops ← getiops(lv.lvpath)
        curlevel ← getiopslevel(lv)
        storageutil ← StorageUtil(lv.getdev())
        if curiops reaches current limit
           and curlevel is not the top level
           and storageutil is lower than threshold
              return "promote"
        else if curlevel is not the baseline level
           and curiops is less than lower level limit
              return "demote"
        else
              return None

Algorithm: StorageUtil(dev)
Input: Physical device dev.
Return: Physical device utilization.
Procedure:
    riops, wiops, rbw, wbw ← metricvalues(dev)
    iopsutil ← riops / MAXRIOPS +
                wiops / MAXWIOPS
    bwutil ← rbw / MAXRBW + wbw / MAXWBW
    return max(iopsutil, bwutil)
```

I/O Tuning Execution. Tuning execution uses libvirt library, which calls QEMU block device tuning interfaces. VM I/O requests traverse virtualization storage stacks and become host-side asynchronous I/O requests executed by QEMU. QEMU provides software interfaces for cloud administrators to specify the IOPS and bandwidth limits of a volume [18]. As it is demonstrated in **Figure 2**, upon the arrival of a block driver **aio** request, the QEMU **system emulator block driver** intercepts it and checks if an I/O wait is needed for I/O rate throttling purpose. If an I/O wait is needed, the request will be sent into a QEMU **coroutine queue**, waiting for a specific time determined by a **throttle schedule timer** and then be executed. The main I/O management feature provided by IOTune is dynamically adjusting the IOPS limit of a volume in real-time according to the time-variant I/O requirements of the volume.

Reservation Metering. Being different from existing storage resource allocation, IOTune takes the volume pricing and metering policies into consideration. Pricing and metering are important for public clouds, in which storage resources are charged. IOTune targets the platform that IOPS is charged separately. The IOPS metering of IOTune is more complicated than static reservation, because the IOPS reservation of a volume is dynamic during its lifetime due to the QoS-aware allocation policy. The metering function gets the QoS level of a volume and updates the metering data at the interval of the tuning period. The metering data records the durations a volume has been served on all IOPS levels. Upon the close of a billing period, the total IOPS bill is calculated. It is the sum of bills of all IOPS levels. The $level_i$ IOPS bill is the product of $level_i$ IOPS price and the duration that the volume has been served at $level_i$. Since the IOPS promotion is based on prediction, it is possible that the IOPS of a volume is promoted but not really used. IOTune metering is based on real resource usage, which avoids tenants being overcharged due to misjudgement of promotions.

Storage-specific Issues. IOTune handles storage-specific issues including the I/O request type and size. Storage systems usually show different read and write performance. This can be caused by device performance features and data layouts. For example, SSDs usually show higher read IOPS than write. Different data layouts on multiple devices for data mirroring or parity may also yield various aggregate read and write performance. IOTune supports separate IOPS tuning for read and write. Larger I/O sizes consume more storage bandwidth. For a volume issuing mostly large requests, instead of IOPS, storage bandwidth will be the dominant bottleneck. Either the provisioned IOPS metric may not be achieved, or the single volume will consume too much bandwidth resource. Thus, I/O requests with various sizes should be treated differently. Although we focus on discussing IOPS, in practice IOTune takes bandwidth into consideration for making tuning decisions.

4 Implementation and Evaluation

In this section, we present results from a detailed evaluation of IOTune using a complete implementation on a QEMU/KVM virtualization platform. The host OS is a 64-bit Ubuntu 14.04 with Linux kernel version 3.13.0-85-generic. QEMU emulator version 2.4.0 and KVM are used as the hypervisor. Ubuntu 14.04 64-bit image is run on the VM as the guest operating system with 4 VCPUs, 16GB memory and ten 100GB SSD virtual disks.

IOPS, tail latency, and cost of ownership are key metrics for data center disks [19]. We examine the following key questions: (1) Compared with static resource allocation, to achieve a same IOPS level how much resource reservation can IOTune reduce? (2) For a same volume admission control policy, how much I/O tail latency and storage utilization can IOTune improve?

4.1 Meeting IOPS requirements with reduced resource reservations

We compare the volume IOPS under various resource provisioning policies via replaying the Bear trace for twelve thousand seconds. We randomly replayed three Bear subtraces and observed similar results. The results we report come from the first twelve thousand seconds replay of the same Bear subtrace analyzed in **Table 2**. For legibility purpose, **Figure 3** only shows a ten minutes run-time episode of volume IOPS. Unlimited indicates there is no IOPS limit imposed on the volume. It represents the natural IOPS requirements of the workload. Static_limit1 and Static_limit2 are set using the 80th percentile and 95th percentile IOPS requirement, which is 1300 and 2400 respectively, of the workload. As it is demonstrated in **Figure 3**, during the periods that the workload has higher IOPS requirements than the reservations, the I/O bursts cannot be served on time. The delay is very serious for Static_limit1 where the resource is under-provisioning. Static_limit2 can serve 95th percentile requirement, but peak I/O requests still show obvious delays. In contrast, IOTune adapts to I/O fluctuations. When I/O bursts arrive, IOTune promotes the volume IOPS limit in near real-time

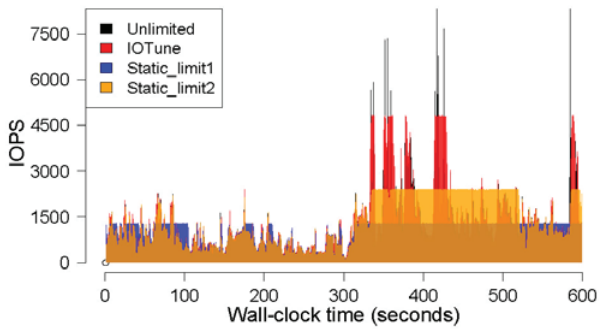


Figure 3: Demonstration of live volume IOPS under various provisioning policies.

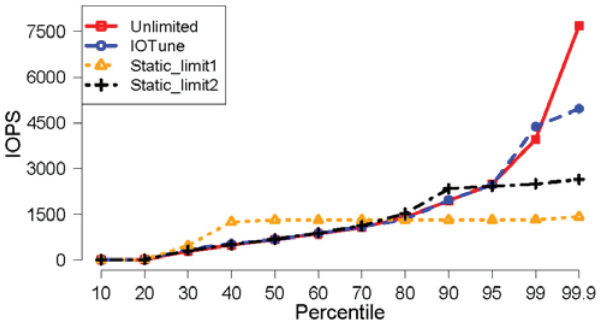
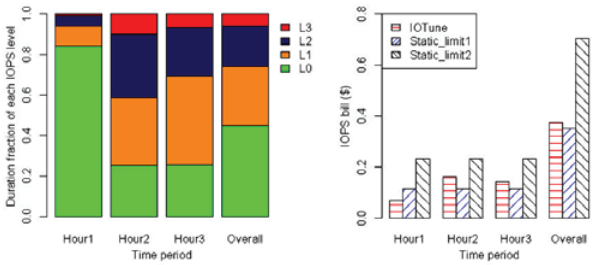


Figure 4: Quantitative analysis of volume IOPS distributions under various provisioning policies.

Figure 5: IOPS bills: IOTune vs. Static provisionings.



(a) The duration of volume served at each IOPS level in IOTune.

(b) The per-hour and total IOPS bills.

to serve the bursts. Although the baseline IOPS reservation of IOTune is set at mere 600 in this case, the 3-level dynamic IOPS promotion enables the volume to achieve a peak I/O processing rate of 4800 during bursts.

Figure 4 demonstrates quantitative comparisons of the IOPS distributions under all policies. IOTune enables volumes to achieve near optimal performance, as if it has been provisioned without an IOPS limit, in 99% of the time. But **Static_limit1** and **Static_limit2** fail to handle the top 20% and 5% I/O peaks, respectively. Although 20% and 5% are not significant in fraction of time, they are burst periods, thus account for about 50% and 20% of total I/O requests. We also can see that **Static_limit1** has a higher 20^{th} percentile to 80^{th} percentile IOPS; and Static limit2 has a higher 80^{th} percentile to 95^{th} percentile IOPS than Unlimited and IOTune. The reason is the delayed requests from peak periods are filled into subsequently low-IOPS periods, changing the distribution of request completion.

Static provisioning policies have constant resource reservations. IOTune adjusts resource reservations on the fly in four levels: L0, L1, L2 and L3. They correspond to IOPS of 600, 1200, 2400 and 4800. IOTune prints IOPS logs so we calculate the durations the volume is served on each IOPS level. As it is demonstrated in **Figure 5**, 70% of the overall time the volume is served under low reservations, L0 and L1. The reservation is promoted only when it is needed. In the first hour, since the request arrival rate is relatively low, about 95% of the time the volume is provisioned with IOPS less than 1200. In the second and third hours, there is a higher fraction of time that the volume is served at L2 and L3. We use the pricing of EBS Provisioned IOPS SSD (io1) volume, $0.065 per provisioned IOPS-month, to compare the IOPS bills of IOTune with static policies. IOTune costs only 6% higher than **Static_limit1** but provides more than two times higher IOPS in the top 20% peak periods. IOTune costs about one half of **Static_limit2** but still provides much higher I/O handling ability in the top 5% peak periods. These results verify that compared with static reservations IOTune can achieve a same or a higher QoS level with considerably less resource reservation.

4.2 Improving end-to-end I/O latency and storage utilization

From the position of tenants and cloud providers, we further evaluate the effects of IOTune on end-to-end I/O latency and on storage utilization. We replay the six traces of **Table 2** each on a volume. We set the static IOPS limit of each volume at the 90^{th} percentile requirement of its work-load, so the total IOPS reservation for the six volumes is 8050. For IOTune, we ensure the same total IOPS reservation. The baseline L0 IOPS of each volume is the same as its static reservation case. For a fair comparison, if the IOPS of a volume needs to be promoted in IOTune case, the promotion can be executed only if the unused total reservation is more than the promotion requirement.

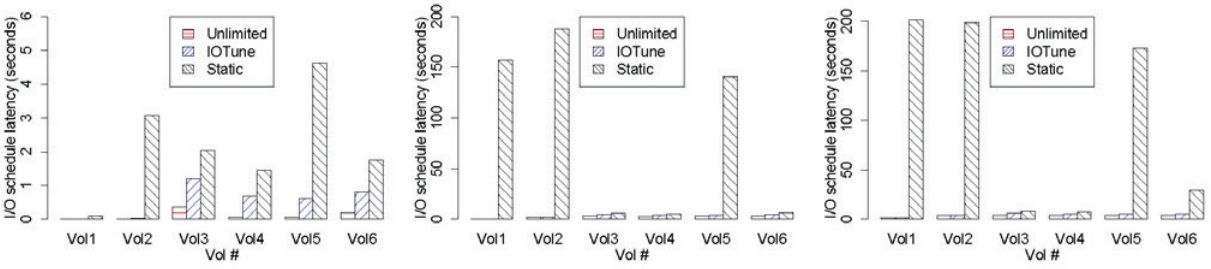


Figure 6: End-to-end I/O schedule latency: IOTune vs. Static provisioning

Figure 6 demonstrates the end-to-end workload I/O latencies of IOTune and Static provisioning. For all volumes and in all cases, IOTune significantly beats the static provisioning and keeps the I/O latency in the same order of magnitude of the Unlimited, which imposes no IOPS limits on volumes. For volume 1, 2, 5, compared with Static provisioning, IOTune reduces 90th and 99th percentile latencies by one to two orders of magnitude. Volume 1, 2, 5 have far higher 90th and 99th percentile than volume 3, 4, 6. This can be explained by **Table 2**, from which we can see the former volumes have much higher 99% to 90% IOPS ratios, thus more dramatic bursts. Another possible reason is volume 1, 2, 5 have higher baseline IOPS reservations, thus their IOPS promotions require more unused resources, which further reduce their chances to get promoted. For example, volume 2 has a 38% promotion failure rate due to lack of resources. For volume 4, the failure rate is only 24%.

For latency evaluation we assume the long I/O delays will be tolerated by tenants. In reality, end users may not tolerate long latencies. For interactive applications, users may leave if a server fails to response in seconds. For storage systems, there are also I/O redirection methods to offload long delayed requests [12]. In these cases, storage system utilization will decrease due to I/O dropout. We assume requests are dropped if the schedule latency is higher than one second and compare the storage utilization of IOTune and Static provisioning. As it is demonstrated in **Figure 7**, if the initial limit is set at 90th percentile arrival rate, IOTune achieves 97% of offline-optimal and has 13% higher overall utilization than Static. The higher utilization mainly results from the first 10 minutes, during which IOTune has 40% higher storage utilization than Static. If the initial limit is set at 80th percentile arrival rate, IOTune achieves 91% of offline-optimal and has 27% higher utilization than Static.

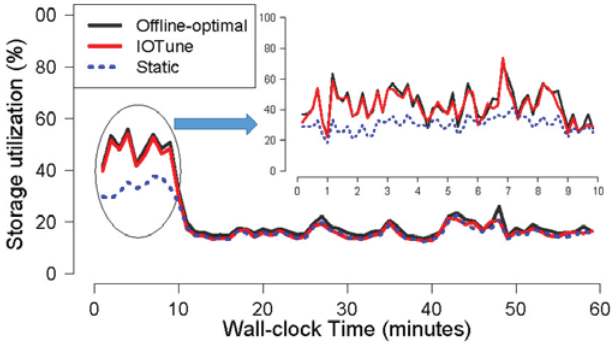


Figure 7: Storage resource utilization: IOTune vs. Static provisioning

5 Conclusion

We have designed IOTune, an adaptive I/O tuning framework for elastic SSD volume performance in multi-tenancy environments. We achieve a complete implementation of IOTune on an Openstack cloud platform. Tests on our staging servers verify that compared with existing static resource reservation IOTune enables a volume to achieve same QoS with 50% less resource reservation, which halves volume bills. Our tests also show that compared with static IOPS provisioning IOTune reduces the end-to-end I/O tail latencies by one to two orders of magnitude. The adaptive resource reservation and metering model of IOTune provides a paradigm for QoS management of cloud storage.

References

[1] S. Byan, J. Lentini, A. Madan, L. Pabon, M. Condict, J. Kimmel, S. Kleiman, C. Small, and M. Storer, "*Mercury: Host-side flash caching for the data center*," in MSST'12, Pacific Grove, USA, April 2012.

[2] R. Koller, L. Marmol, R. Rangaswami, S. Sundararaman, N. Talagala, and M. Zhao, "*Write policies for host-side flash caches*," in FAST'13, San Jose, USA, Feb 2015.

[3] C. Li, P. Shilane, F. Douglis, H. Shim, S. Smaldone, and G. Wallace, "*Nitro: A capacity-optimized ssd cache for primary storage*," in USENIX ATC'14, Philadelphia, PA, June 2014.

[4] D. Qin, A. D. Brown, and A. Goel, "*Reliable writeback for client-side flash caches*," in USENIX ATC'14, Philadelphia, USA, June 2014.

[5] R. Koller, A. J. Mashtizadeh, and R. Rangaswami, "*Centaur: Host-side ssd caching for storage performance control*," in ICAC'15, Grenoble, France, July 2015.

[6] D. Arteaga, J. Cabrera, J. Xu, S. Sundararaman, and M. Zhao, "*Cloudcache: On-demand flash cache management for cloud computing*," in FAST'16, Santa Clara, CA, February 2016.

[7] B. Schroeder, R. Lagisetty, and A. Merchant, "*Flash reliability in production: The expected and the unexpected*," in FAST'16, Santa Clara, USA, February 2016.

[8] A. W. Services. *Amazon elastic compute cloud: User guide for linux instances*. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-ug.pdf

[9] Google. *Google compute engine pricing*. [Online]. Available: https://cloud.google.com/compute/pricing

[10] A. Oo. *Premium storage: High-performance storage for Azure virtual machine workloads*. [Online]. Available: https://azure.microsoft.com/en-us/documentation/articles/storage-premium-storage

[11] C. Weddle, M. Oldham, J. Qian, and A.-I. A. Wang, "*Paraid: A gear-shifting power-aware raid*," in FAST'07, San Jose, USA, Feb 2007.

[12] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. Rowstron, *"Everest: Scaling down peak loads through i/o off-loading*," in OSDI'08, San Diego, California, December 2008

[13] A. Verma, R. Koller, L. Useche, and R. Rangaswami, "*Srcmap: Energy proportional storage using dynamic consolidation*," in FAST'10, San Jose, USA, February 2010.

[14] E. Thereska, A. Donnelly, and D. Narayanan, "*Sierra: Practical power-proportionality for data center storage,*" in Eurosys'11, Salzburg, Austria, April 2011.

[15] D. Arteaga and M. Zhao, "*Client-side flash caching for cloud systems,*" in SYSTOR'14, Haifa, Israel, June 2014.

[16] A. Gulati, I. Ahmad, and C. A. Waldspurger, *"Parda: Proportional allocation of resources for distributed storage access*," in FAST'09, San Francisco, California, February 2009.

[17] A. Gulati, A. Merchant, and P. J. Varman, "*mClock: Handling throughput variability for hypervisor io scheduling*," in OSDI'10, Vancouver, Canada, October 2010.

[18] R. Harper. (2011) *Keep a limit on it: Io throttling in qemu*. [Online]. Available: http://www.linux-kvm.org/images/7/72/2011-forum-keep-a-limit-on-it-io-throttling-in-qemu.pdf

[19] E. Brewer. (2016) *Spinning disks and their cloudy future*. [Online]. Available: www.usenix.org/sites/default/files/conference/protect ed-files/fast16 slides brewer.pdf

Notes

[1]We adopt this method because there lacks publicly available concurrent block I/O traces of co-locating volumes. The exact subtrace we use is from http://visa.lab.asu.edu/traces/bear/blkios-2012323010000.gz

About Virtustream

Virtustream, a Dell Technologies business, is the enterprise-class cloud service and software provider trusted by enterprises worldwide to migrate and run their mission-critical applications in the cloud. For enterprises, service providers and government agencies, Virtustream's xStream management platform and Infrastructure-as-a-Service (IaaS) meets the security, compliance, performance, efficiency and consumption-based billing requirements of complex production applications in the cloud - whether private, public or hybrid.