More IOPS for Less: Exploiting Burstable Storage in Public Clouds

Hojin Park, Gregory R. Ganger, George Amvrosiadis
Carnegie Mellon University

Abstract

Burstable storage is a public cloud feature that enhances cloud storage volumes with credits that can be used to boost performance temporarily. These credits can be exchanged for increased storage throughput, for a short period of time, and are replenished over time. We examine how burstable storage can be leveraged to reduce cost and/or improve performance for three use cases with different data-longevity requirements: traditional persistent storage, caching, and ephemeral storage. Although cloud storage volumes are typically priced by capacity, we find that each AWS `gp2` volume starts with the same number of burst credits. Exploiting that fact, we find that aggressive interchanging of large numbers of small short-term volumes can increase IOPS by up to 100× at a cost increase of only 10–40%. Compared to an AWS `io1` volume provisioned for the same performance, such interchanging reduces cost by 97.5%.

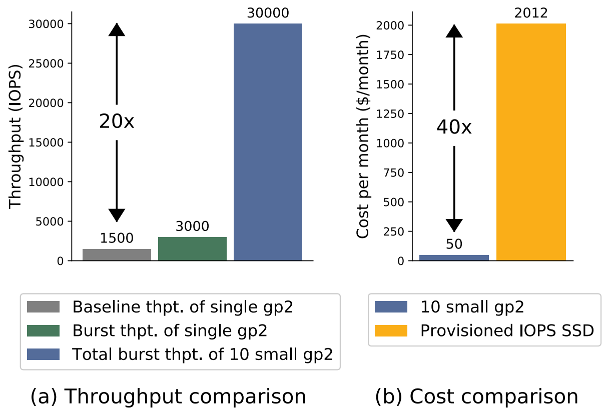


Figure 1: Throughput comparison (left) of three storage configurations of the same cost. 10 small `gp2` volumes provide 20× more IOPS (during burst) than a single volume. Purchasing this throughput directly increases cost by 40× (right).

1 Introduction

Public clouds have democratized access to computing resources for a variety of workloads by offering a plethora of services to users. For data storage, the broad array of solutions includes products targeted to data expected to remain cold, accessed at high rates, or accessed at specific granularities. Many storage services, however, follow a cost model centered around throughput per GiB. *Burstable storage* is an emerging storage service type that augments this model by endowing cloud storage volumes with credits. Each credit provides an increase in throughput, for a limited time, and is replenished after a predetermined period of time; these parameters are determined by cloud provider policies. This paper describes systematic ways of leveraging burstable storage to increase performance at low cost, opening new opportunities for system designers and challenges for interface designers.

Endowing a cloud storage volume with burst credits is currently available at no additional cost. Amazon Web Services (AWS) has provided the burst feature through EBS volumes since 2016, and recently Microsoft Azure also introduced burst credits for Azure Premium SSD disks with a policy similar to the one provided by AWS. Available credits are used when the storage volume exceeds the throughput it has been rated for. During periods when requested throughput is below this threshold, credits can be re-accumulated.

We observed two interesting characteristics of burstable storage (§2). First, the *bucket* of burst credits is initially full when a burstable volume is created. This allows the burst duration to be extended by replacing burstable storage volumes that have emptied their credit buckets with new ones. Second, burst throughput limits are set per volume, whereas a volume's cost is determined by its capacity. In other words, using two burstable storage volumes, instead of using one with their combined capacity, doubles the total available burst throughput without affecting total cost. Figure 1 illustrates this effect further and is fully explained in Section 2.2.

We show that exploiting these two characteristics of burstable storage volumes can significantly increase performance and/or reduce cost for three use cases with different data-longevity requirements: traditional persistent data storage (§3.1), tiered storage where the faster storage tier is used to cache data from the slower tier (§3.2), and ephemeral storage that can be discarded once a given task is completed (§3.3). We prototyped and evaluated the first two use cases on AWS. Naturally, less work is involved with interchanging volumes when the data-longevity requirements are lower—for ephemeral data, older volumes can simply be discarded in favor of new empty ones, whereas all data must be migrated for traditional persistent storage. Still, our experiments show that aggressive interchanging of burstable storage volumes can deliver up to 100× more IOPS at only 10-40% higher cost, or the higher performance level at 97.5% lower cost. Although we demonstrate our idea on AWS, we believe our approach can be applied to other public clouds (e.g., Microsoft Azure) as well, so long as every new burstable storage initially has full burst credits and there is no penalty for using the credits.

2 Whither Burstable Storage?

This section provides background on current burstable storage services, and key characteristics of burstable storage volumes that inform the way we design systems to use them.

2.1 Burstable Storage Services

Many real-world workloads exhibit burstiness in resource usage. This introduces the classical tradeoff between overprovisioning cost and the ability to meet Service Level Objectives (SLOs) for users. To help users navigate this tradeoff, public cloud providers such as AWS, Google Cloud Engine (GCE), and Microsoft Azure have started providing services that allow for resource usage bursts. One example is burstable instances, which can temporarily absorb CPU usage bursts at no additional cost. The example we focus on in this paper is the equivalent service for storage, i.e., the *burstable storage service*. When applied to a storage volume, the volume is converted into a *burstable storage volume* endowed with I/O credits that can be redeemed to absorb I/O bursts.

As an example, AWS currently provides `gp2` volumes that are rated for 3 IOPS per GiB of provisioned capacity. For a volume that is smaller than 1,000 GiB, converting it into a burstable storage volume allows for a maximum burst throughput of 3,000 IOPS beyond its baseline throughput for 30 minutes. This performance boost is accounted for through I/O credits, used per I/O operation. Initially, each `gp2` volume is assigned 5.4 million I/O credits at creation time and the rate at which they are used is capped, hence $\frac{\text{5.4 million I/O credits}}{\text{Max Burst IOPS} - \text{Current IOPS}} \geq 30$ minutes. When fewer IOPS are used than the maximum threshold, I/O credits are accrued.

2.2 Characteristics of Burstable Storage

In our work, we find two important characteristics of burstable storage volumes. First, every new burstable storage volume is endowed with enough credits to achieve maximum burst throughput for at least 30 minutes. Thus, burstable volumes whose credits are depleted can be replaced by new volumes to maintain this maximum burst throughput. For workloads that treat storage as ephemeral this extra throughput comes at no cost. For other workloads, data will need to be migrated before a volume is replaced. We evaluate the migration overhead in Section 3 and show that it is negligible enough compared to the performance benefit.

Second, even though cloud storage is typically priced by capacity, burstable volume I/O credits are assigned per volume. Thus, splitting a burstable storage volume into multiple smaller volumes can result in additional I/O credits. Figure 1(a) shows a throughput comparison between three 500 GiB `gp2` AWS storage configurations. Using the aggregated burst throughput of 10 `gp2` volumes with a capacity of 50 GiB each achieves 20× higher throughput than a single 500 GiB `gp2` volume. Note that all three volumes cost the same, since `gp2` charges by capacity. Figure 1(b) shows the cost difference between two configurations of the same capacity and IOPS. The blue bar represents a burstable volume that is rated for 20× less throughput, and the yellow bar represents a Provisioned IOPS SSD (`io1`) volume that is guaranteed to achieve 30,000 IOPS at all times. The price of using 500 GiB of `io1` with 30,000 IOPS is $2012.5 per month, which is 40× higher than using `gp2`. In Section 3, we describe how to exploit these two characteristics to enhance performance.

3 Burstable Storage Applications

We demonstrate three use cases that can take advantage of a burstable storage service: persistent data storage (§3.1), SSD caching (§3.2), and ephemeral data storage (§3.3). Our experiments emphasize two benefits of burstable storage: high aggregated burst throughput of small volume collections, and burst credit renewal through volume replacement.

3.1 Persistent data storage

One of the benefits we identified with burstable storage volumes is that new volumes start with full credits, so they can be used to replace old ones that have spent their burst budget. To leverage this characteristic for persistent data, we need to ensure data is migrated before the old volume is removed, and that the data remains available during the migration process.

Design. Figure 2 illustrates our prototype design. To expose a total capacity C, we create a RAID-0 array with N burstable storage volumes, each with a capacity of $\frac{C}{N}$. Initially, only the N volumes are present (Fig. 2(a)), and their aggregated performance is N times that of a single burstable volume. As the volumes use up their burst credits, a new set of N volumes is created and each volume is paired up to an old volume in a RAID-1 scheme (Fig. 2(b)). Then, the software RAID rebuild process for each pair migrates data from the old volumes to the new ones, while maintaining data availability (Fig. 2(c)). After the rebuild process is complete for all RAID-1 pairs, old volumes are deleted and we transition to using the credits available in the new volumes (Fig. 2(d)).

Experimental setup. We evaluate our design with data capacities of 100 GiB and 300 GiB. For each case, we use 10 `gp2` volumes as the individual burstable storage volumes with capacities of 10 GiB and 30 GiB each, respectively. As our baseline we use a single `gp2` volume with the same total capacity. Volumes are attached to a c5.9xlarge AWS instance that supports up to 40,000 IOPS for attached volumes, which is sufficient to show the performance of our design. Our evaluation uses two workloads generated by the Flexible I/O Tester, FIO [6]: random reads, and an equal mix of random reads and writes. The I/O unit used is 4 KiB, and file sizes are 80 GiB and 250 GiB for each 100 GiB and 300 GiB of persistent data storage.

Evaluation. Figure 3 shows an 80-minute window of persistent data storage throughput with total capacity of 100 GiB and 300 GiB. We limit capacity to 300 GiB per instance be-

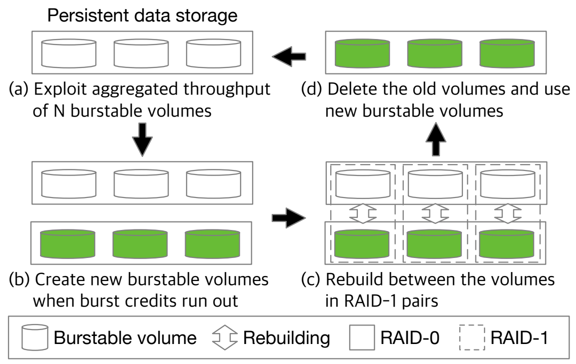

Persistent data storage
(a) Exploit aggregated throughput of N burstable volumes
(d) Delete the old volumes and use new burstable volumes
(b) Create new burstable volumes when burst credits run out
(c) Rebuild between the volumes in RAID-1 pairs
Burstable volume
Rebuilding
RAID-0
RAID-1

Figure 2: Persistent data storage using burstable storage volumes. Initially, (a) the system uses N small volumes to exploit their aggregated burst throughput. After using up their burst credits, (b) N new volumes are created, (c) data is migrated using software RAID, and (d) old volumes are deleted.

cause we are limited by the number of volumes that can be attached to a single instance, and larger per-volume capacities would result in using up more than half of the volume's burst credits for data migration. Our design's expected aggregate throughput is 30,000 IOPS, because we use 10 `gp2` volumes and each volume's maximum burst throughput is 3,000 IOPS. Figure 3 shows our design achieves this throughput, which is 100× and 33× higher than the baseline `gp2` volume throughput with a capacity of 100 GiB and 300 GiB, respectively.

The main challenge of using burstable volumes for persistent data storage is data migration for volume replacement. Since volumes in each software RAID-1 pair are rebuilt at the block level, the rebuild process duration is proportional to the volume's capacity. For 10 volumes of equal capacity that amount to 100 GiB, 300 GiB, and 500 GiB of total capacity, we have measured the rebuild time to take 3.6, 12, and 18 minutes out of the 30-minute burst throughput interval.

Although software RAID-1 allows the system to be available for reading and writing during the rebuild phase, usable storage throughput decreases. Specifically, write throughput is degraded because writes must be persisted to both RAID-1 volumes. But as expected, read throughput increases because data already rebuilt can be read from both of the volumes, at twice the original bandwidth. Regarding the compute overhead during the RAID-1 copy, we have measured that the copy between two volumes only incurs 1-2% CPU utilization on a single core and 6% for three simultaneous copy operations. Therefore, we believe that CPU overhead is low even with multiple on-going RAID-1 copy operations.

Lastly, we study the financial efficiency of our design for 100 GiB of persistent storage. The baseline approach, using a single 100 GiB `gp2` volume costs \$10 per month. Our design uses up to twice the capacity across all burstable storage volumes during the rebuild phase, which accounts for 10% of total running time on average. Thus, our approach costs \$11 per month. This means that our approach achieves 100× higher throughput at 10% additional cost. In the case of 300

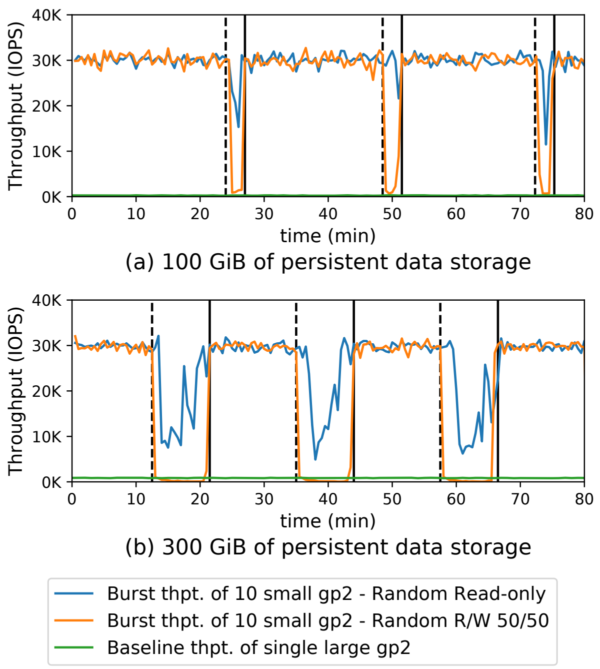


Figure 3: Throughput evaluation of 100 GiB and 300 GiB persistent data storage. Vertical dashed and solid lines indicate the start and end of data migration. Our design (blue and orange lines) shows $33-100\times$ higher throughput for 40% additional cost compared to the baseline (green line).

GiB persistent data storage, the cost is 40% higher for 20× higher throughput.

Implications. Our results indicate that as the total capacity of persistent data storage increases, both cost-efficiency and average throughput decrease because the rebuild time increases as well. It is possible to reduce the rebuild time by using more smaller-capacity burstable volumes, because data migration time is proportional to the volume capacity, but the number of volumes that can be attached to the single instance is limited. We can also mitigate the throughput loss during the rebuild time by replacing a single volume at a time instead of replacing all the volumes at once, but the system will spend a larger percentage of time with at least one rebuild in progress. The choice of which design to use depends on the workload characteristics. Nonetheless, even for the least well-fit of our examples (when all data is long-term and must be retained), burstable storage can be exploited to significant benefit.

3.2 SSD caching

SSD volumes are often used as caches due to their high throughput and low latency compared to HDD volumes, especially for workloads with lower degrees of sequentiality. The latter remain an attractive storage solution due to their relative low cost. The temporal locality of accesses that improves data cache value is a good match for burstable storage volumes, which provide temporary throughput bursts. Moreover,

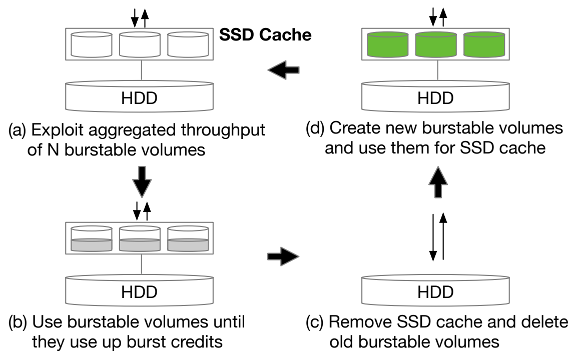

SSD Cache
HDD
(a) Exploit aggregated throughput of N burstable volumes
(b) Use burstable volumes until they use up burst credits
(c) Remove SSD cache and delete old burstable volumes
(d) Create new burstable volumes and use them for SSD cache

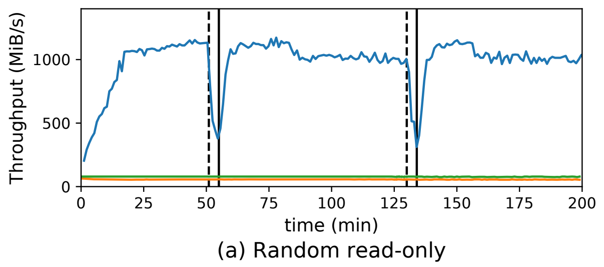
Figure 4: SSD caching using burstable storage volumes. Initially, (a) *N* burstable volumes are used as an SSD cache. (b) Cache hits then consume burst credits, and (c) once burst credits are depleted, the volumes are removed and (d) replaced with new burstable volumes.

cache data need not be persistent (unless write-back caching is used), so data migration is not necessary when we replace old burstable volumes with new ones. These characteristics allow our design to achieve low cost overhead for significant performance improvement for caching.

Design. Figure 4 demonstrates our design of SSD caching using burstable volumes. We initially create a RAID-0 array of *N* burstable volumes to act as the cache storage (Fig. 4(a)), which uses the LRU policy. Although the maximum throughput of this cache is *N* times higher than the burst throughput of a single volume of the same capacity, our cache cannot exploit the maximum throughput before it is populated with data. Over time burst credits are used, either as the cache storage is organically populated by cache misses, or by requests served directly (and faster) by the SSD (Fig. 4(b)). Once the burst credits are used, the cache storage volumes are removed (Fig. 4(c)) and replaced by a new set of *N* volumes (Fig. 4(d)).

Experimental setup. We evaluate our SSD cache design using a cache storage with a total capacity of 100 GiB. For our design we create this cache storage using 10 gp2 volumes with a capacity of 10 GiB each. For our baseline we use a single gp2 volume with a capacity of 100 GiB. For the underlying HDD storage, we use 2 TiB of Throughput Optimized HDD (st1) EBS volume, which has a baseline throughput of 80 MiB/s. In both our setup and the baseline, the volumes are attached to a AWS c5.9xlarge instance. We use two workloads generated through the FIO benchmark: random read-only and a mix of 90% random reads and 10% random writes. We use the Zipfian random distribution with theta 1.2 to generate the data access pattern (i.e., 90% of requests access 10% of the total data), 1 MiB I/O unit, and a file size of 200 GiB. We prototype our design using *lvmcache* [11] configured as a write-through cache, with a cache chunk size of 32 KiB.

Evaluation. Figure 5 shows a 200-minute window of SSD caching performance. We compare the performance of our design with the baseline, which is using a single 100 GiB gp2 volume as cache storage. For read-only and 90%-read/10%-write access patterns, our design performs 18.6× and 13.9× faster than the baseline on average, respectively.



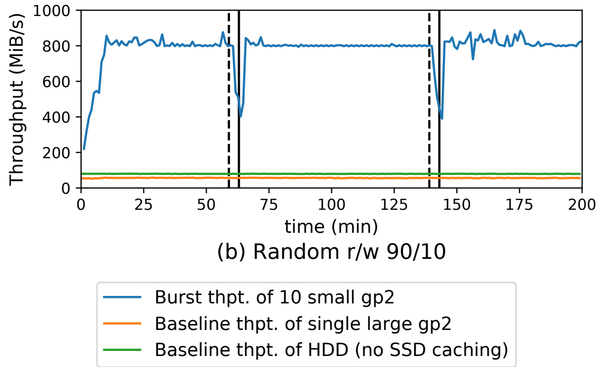


Figure 5: Throughput evaluation of SSD caching using burstable storage. Vertical dashed and solid lines indicate the start and end times of replacing burstable volumes. Our design (blue line) increases throughput by 14-19×, at 5.2% additional cost compared to the baseline (orange line). Using a single SSD cache volume results in worse performance than using no cache at all (green line) due to the low throughput of single gp2 volume.

We measure data access performance of st1 without SSD caching at 79 MiB/s on average for both access patterns. After using a single 100 GiB gp2 volume as cache storage, the total throughput *decreases* to 56 MiB/s in both cases. Throughput performance decreases despite the cache storage because the baseline throughput of gp2 is worse than that of st1. With our multi-volume design, however, total throughput increases to 1,047 MiB/s and 784 MiB/s for random read-only and random read/write workloads, respectively. For the read-only workload this matches the expected throughput of 30,000 IOPS that is supported by our design. For the workload mix of 90%-read/10%-writes, the performance is limited by the write throughput of st1 volume, which is 79 MiB/s.

Next, we study the cost overhead of replacing the burstable storage volumes for the SSD cache. One of the advantageous properties of using the burstable volumes as an SSD cache storage is that cache data does not need to be persistent if a write-through policy is used. This feature results in a negligible overhead of replacing the burstable volumes since data migration is unnecessary. Figure 5 illustrates the storage replacement duration. In terms of cost overhead, our design charges almost the same price as the baseline, because the volume replacement overhead is negligible. The evaluation results indicate that our approach uses twice the number of gp2 volumes for 5.2% of the total runtime on average, i.e., the cost increases by only 5.2% for 19× and 14× higher throughput for read-only and read-write workloads, respectively.

Implications. One of the costs of replacing burstable volumes in our design is that cache state is reset after the replacement. Although the cache warm-up duration seems negligible in Figure 5, which is in part due to using a data access pattern based on the Zipfian distribution, we can reduce it even further by replacing a single volume at a time. Another notion is that it is possible to lengthen the volume replacement time interval. We used a fixed, 75-minute interval in our evaluation, which is typical of the time it takes for the FIO workloads to almost exhaust all burst credits. For arbitrary workloads, it is possible to extend the time interval by monitoring remaining burst credits and selecting the best time to reset the cache. A longer interval would increase cost-effectiveness further. It is possible to check the remaining burst credits with small overhead using the monitoring tools (e.g., Amazon CloudWatch [5]) provided by public clouds.

3.3 Ephemeral data storage

This section discusses one final use case we consider: using burstable volumes for ephemeral storage. We refer to data that are temporary and transient as ephemeral data. One of the appeals of using burstable volumes as ephemeral data storage is the harmony between short burst duration of burstable volumes and the short-lived nature of ephemeral data. We specifically discuss two applications that leverage ephemeral storage: (1) scientific application checkpointing, and (2) local storage storing ephemeral data such as intermediate computation results for cloud programming framework worker nodes.

Checkpointing is a common fault tolerance technique used by scientific applications, which consists of periodically persisting the application's state to storage. In the event of a failure, the application can restart from the point in time captured by the last checkpoint, instead of repeating all the computation that preceeded the checkpoint. Checkpointing overhead, however, is becoming increasingly significant as applications increase in size (alongside the clusters they run on) much faster than storage bandwidth [12]. Burstable volumes can play a key role in enhancing the performance of checkpointing with no additional cost. The design is straightforward: the system uses a collection of small burstable volumes as the ephemeral storage backend for checkpointing, exploiting the aggregated burst throughput and replacing old burstable volumes as they use up their burst credits. During volume replacement, no data migration is required. The system can simply detach the old burstable volumes that contain checkpoint data and keep the detached volumes until they are necessary. Further, depending on the application requirements, if (at least) one complete, recent checkpoint is available after removing a given volume and is deemed sufficient, then the removed volume can be discarded to further reduce costs.

Intermediate computation results of cloud programming frameworks is another type of ephemeral data. Many cloud programming frameworks generate an extensive amount of intermediate data in their data pipelines, and they need to save the intermediate data in storage because of its massive amount. If the framework saves or checkpoints the intermediate data between two consecutive jobs of the data pipeline to storage, then storage throughput affects the total running time of the job. Burstable volumes are a cost-efficient option as ephemeral storage to save the intermediate data with high throughput. One possible design is using different collections of burstable storage volumes for each intermediate data. That is, for each intermediate data generated in different steps of data pipeline, the system uses new ephemeral storage with full burst credits. Since each intermediate data lives for a short duration [13], each ephemeral storage can maintain its burst throughput throughout the lifetime of the intermediate data. After the intermediate data is used, the system need not keep the corresponding volumes. Thus, the system can delete the volumes without any data migration.

4 Related Work

Several systems [1–4, 7–10, 14–16] have been proposed to answer the question of how to cost-efficiently exploit resources in the public cloud while maintaining moderate or even higher performance. Tributary [7] suggests a cost-effective way of selecting public cloud resources that satisfy given SLO requirements by using spot instances. Kadekodi et al. [10] present a client-side packing to lower the cost and improve the performance in using cloud object stores. Some of the works [1, 2, 4, 9, 15, 16] focus on analyzing and utilizing burstable resources in the public cloud. Jiang et al. [9] analyze the performance of burstable instances theoretically and present the framework that predicts the performance of provisioned burstable instances. Burscale [4] demonstrates the way of using burstable instances to lower the cost of autoscaling in the public cloud. Wang et al. [15] improve cost savings using both spot and burstable instances to prototype Memcached in the public cloud. MArk [16] also exploited both spot and burstable instances to achieve SLO compliance of machine learning inference serving system cost-effectively. Our work explores another opportunity in this space, burstable storage, and how it can be exploited.

5 Conclusion

Burstable storage, as currently offered in public clouds, creates opportunities for higher performance and/or lower cost than first appears. For three usage-type examples with different data-longevity characteristics, we show that aggressive interchanging of small burstable volumes can achieve up to 100× higher throughput at only 10–40% higher throughput, or a given throughput at 97.5% lower cost. These results expose opportunities for additional exploration of storage application designs and of less exploitable storage pricing models.

Discussion Topics

In this work, we identified a sweet spot of public cloud providers' policy regarding performance and cost of burstable storage and presented use cases that can benefit from the policy. We believe that our approach generates several discussion topics:

1) **Is it reasonable to exploit the policies and services of public clouds for system design?** Many works, including this paper, have investigated ways of exploiting resources of the public clouds based on the policies set by the public cloud providers. However, one of the concerns in this approach is that those policies and services are inherently variable. Therefore, some of the works can become less performant or even inapplicable when public cloud providers modify the policy of their services. The community should discuss how to deal with this issue from the perspectives of both researchers and public cloud providers. Further, should public cloud providers amend their policies to prevent customers from using these services in an unintended way, with the inevitable consequence of more complex policies and/or less widely-effective solutions... or, should they add even more features and allow customers to exploit them in varied ways to achieve even better performance?

2) **Are there any other suitable application types that can take advantage of burstable storage?** We discussed three application types that benefit from burstable storage, yet we believe that there are still other ones not discussed in this paper. It will be interesting to discuss when and how our approach can be a practical design choice for real-world applications. Furthermore, we can also discuss other techniques for the efficient management of burstable storage volumes.

3) **How should cloud storage pricing and SLOs be designed?** Several public cloud services, such as serverless computing, burstable instances, and spot instances, have been studied to identify use cases and ways of using them profitably. Most generally, this points at the consistent challenge of designing pricing and SLO models. We think that this is particularly true for cloud storage, which is commonly provisioned as software-capped IOPS or MiB/s for a given price per GB. Although public cloud providers mention that random and sequential data access patterns may affect the provisioned throughput, our empirical evaluation on AWS does not show the big throughput difference between two different data access patterns. This property may simplify storage system design for clients, but puts pressure on providers to deal with inefficient access patterns that require more resources. So, what is the right balance of responsibility and cost between clients and providers, for not-easily-time-shared resources like storage access, and how should they be reflected in policies?

Acknowledgements

We thank the anonymous reviewers and Mahesh Balakrishnan, our shepherd, for helping us improve the presentation of the paper. We also thank the members and companies of the PDL Consortium—Alibaba, Amazon, Datrium, Facebook, Google, Hewlett Packard Enterprise, Hitachi, IBM Research, Intel, Micron, Microsoft Research, NetApp, Oracle, Salesforce, Samsung, Seagate, and Two Sigma. Hojin Park is supported in part by Korea Foundation for Advanced Studies.

References

[1] A. Ali, R. Pinciroli, F. Yan, and E. Smirni. CEDULE: A Scheduling Framework for Burstable Performance in Cloud Computing. In *2018 IEEE International Conference on Autonomic Computing (ICAC)*, pages 141–150, Sep. 2018.

[2] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. It's Not a Sprint, It's a Marathon: Stretching Multi-Resource Burstable Performance in Public Clouds (Industry Track). In *Proceedings of the 20th International Middleware Conference Industrial Track*, Middleware '19, page 36–42, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Ahmed Ali-Eldin, Jonathan Westin, Bin Wang, Prateek Sharma, and Prashant Shenoy. SpotWeb: Running Latency-Sensitive Distributed Web Services on Transient Cloud Servers. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery.

[4] Ataollah Fatahi Baarzi, Timothy Zhu, and Bhuvan Urgaonkar. BurScale: Using Burstable Instances for Cost-Effective Autoscaling in the Public Cloud. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, page 126–138, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Amazon CloudWatch. https://aws.amazon.com/cloudwatch/.

[6] Flexible I/O Tester. http://freshmeat.sourceforge.net/projects/fio.

[7] Aaron Harlap, Andrew Chung, Alexey Tumanov, Gregory R. Ganger, and Phillip B. Gibbons. Tributary: spot-dancing for elastic services with latency SLOs. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 1–14, Boston, MA, July 2018. USENIX Association.

[8] Yu-Ju Hong, Jiachen Xue, and Mithuna Thottethodi. Dynamic Server Provisioning to Minimize Cost in an IaaS Cloud. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '11, page 147–148, New York, NY, USA, 2011. Association for Computing Machinery.

[9] Yuxuan Jiang, Mohammad Shahrad, David Wentzlaff, Danny H. K. Tsang, and Carlee Joe-Wong. Burstable Instances for Clouds: Performance Modeling, Equilibrium Analysis, and Revenue Maximization. In *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019*, pages 1576–1584. IEEE, 2019.

[10] Saurabh Kadekodi, Bin Fan, Adit Madan, Garth A. Gibson, and Gregory R. Ganger. A Case for Packing and Indexing in Cloud File Systems. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, Boston, MA, July 2018. USENIX Association.

[11] lvmcache(7) - Linux manual page. http://man7.org/linux/man-pages/man7/lvmcache.7.html. [Online; posted 30-November-2019].

[12] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski. Design, Modeling, and Evaluation of a Scalable Multi-Level Checkpointing System. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, page 1–11, USA, 2010. IEEE Computer Society.

[13] Kai Ren, YongChul Kwon, Magdalena Balazinska, and Bill Howe. Hadoop's adolescence: an analysis of Hadoop usage in scientific workloads. *Proceedings of the VLDB Endowment*, 6:853–864, 08 2013.

[14] Ao Wang, Jingyuan Zhang, Xiaolong Ma, Ali Anwar, Lukas Rupprecht, Dimitrios Skourtis, Vasily Tarasov, Feng Yan, and Yue Cheng. InfiniCache: Exploiting Ephemeral Serverless Functions to Build a Cost-Effective Memory Cache. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 267–281, Santa Clara, CA, February 2020. USENIX Association.

[15] Cheng Wang, Bhuvan Urgaonkar, Aayush Gupta, George Kesidis, and Qianlin Liang. Exploiting Spot and Burstable Instances for Improving the Cost-Efficacy of In-Memory Caches on the Public Cloud. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, page 620–634, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1049–1062, Renton, WA, July 2019. USENIX Association.