Provider versus Tenant Pricing Games for Hybrid Object Stores in the Cloud

The tiered object store detailed here is designed for the cloud, taking into account both fast and slow storage devices. The resulting hybrid store exposes the tiering to tenants with a dynamic pricing model based on the tenants' usage and the provider's desire to maximize profits. The tenants leverage knowledge of their workloads and current pricing information to select a data placement strategy that would meet the application requirements at the lowest cost. This approach allows both a service provider and its tenants to engage in a pricing game, which yields a win-win situation.

Yue Cheng and
M. Safdar Iqbal
Virginia Tech
Aayush Gupta
IBM Research–Almaden
Ali R. Butt
Virginia Tech

Cloud object stores offer a cost-effective solution for storing vast amounts of data in the cloud. To make data analytics easy to deploy and elastically scale in the cloud, cloud providers typically let their tenants run Big Data processing jobs on the data stored in the object stores. This eliminates the need to copy terabytes of data to the local storage of the computing nodes running the data analytics jobs. Commercial cloud-based Big Data platforms such as Amazon Elastic MapReduce (EMR) and Microsoft Azure HDInsight go a step further and directly employ object stores as their primary storage.

Cloud vendors continue to use low-cost hard disk drives (HDDs) as the underlying storage medium for their cloud-object storage deployments. This is because the price gap between HDDs and solid-state drives (SSDs) continues to be significant, especially for data-center-scale deployments. Object stores have traditionally been used as data dumps for large objects, such as backup archives and large-volume pictures or videos — essentially, use cases where SSDs would incur a high acquisition as well as maintenance cost,[1] such as premature device replacement. Nevertheless, recent research has shown that SSDs can deliver significant benefits for many types of Big Data analytics workloads.[2,3] Even newer, promising technology on this front doesn't adequately address the cost and performance tradeoffs. For example, while the 3-bit multi-level cell (MLC) negative-AND (NAND) technology promises to deliver higher SSD densities and potentially drive down the acquisition cost, it has taken a major toll on the SSD device lifetime.[4-6]

Tiered storage is used in many contexts to balance the HDD–SSD cost and benefits by distributing the work-

Published by the IEEE Computer Society
1089-7801/16/$33.00 © 2016 IEEE

Related Work in Cloud Storage and Pricing

Over the years, many have grappled with packaging cloud services that provide clients with performant yet cost-efficient cloud storage, handling workloads in a variety of ways, while still issuing competitive-yet-profitable price points. In terms of *strategic tiering*, researchers have demonstrated that adding a solid state drive (SSD) tier for serving reads is beneficial for Hadoop and the Hadoop Distributed File System (HDFS)-based HBase.[1] Existing implementations of cloud object stores provide mechanisms for tiered storage. Ceph, which exposes an object store API, has also added tiering support (see http://docs.ceph.com/docs/hammer/rados/operations/pools/). The cost model and tiering mechanism used in traditional storage tiering approaches[1,2] can't be directly applied to batch-processing analytics running in a public cloud environment, mainly due to cloud storage and analytics workload heterogeneity. Our work focuses on providing insights into the advantages of dynamically priced, tiered, object storage management involving both cloud providers and tenants.

Researchers have also looked at cloud *dynamic* pricing.[3,4] CRAG[5] focuses on solving cloud resource allocation problems using game-theory-based schemes, while Jorge Londono and his colleagues[6] propose a cloud resource allocation framework using colocation game strategy with static pricing. Agmon Ben-Yehuda and his colleagues[7] propose a game-theory-based market-driven bidding scheme for memory allocation in the cloud. In the context of network pricing, Soumya Sen and his colleagues[8] study a wide variety of pricing and incentivizing mechanisms for solving the network congestion problems. We adopt a simplified game-theory model, where the cloud providers give incentives in the form of dynamic pricing and tenants adopt tiering in object stores for achieving their goals.

References

1. T. Harter et al., "Analysis of HDFS under HBase: A Facebook Messages Case Study," *Proc. Usenix Conf. File and Store Technologies*, 2014; www.usenix.org/node/179859.
2. L.M. Grupp, J.D. Davis, and S. Swanson, "The Bleak Future of NAND Flash Memory," *Proc. Usenix Conf. File and Store Technologies*, 2012; http://cseweb.ucsd.edu/~swanson/papers/FAST2012BleakFlash.pdf.
3. Z. Liu et al., "Pricing Data Center Demand Response," *ACM Sigmetrics Performance Evaluation Rev.* vol. 42, no. 1, 2014, pp. 111–123.
4. W. Shi et al., "An Online Auction Framework for Dynamic Resource Provisioning in Cloud Computing," *Proc. ACM Conf. Sigmetrics*, 2014, pp. 71–83.
5. V. Jalaparti et al., *Cloud Resource Allocation Games*, tech. report, Dept. Computer Science, Univ. of Illinois, Urbana-Champaign, 2010; http://hdl.handle.net/2142/17427.
6. J. Londono, A. Bestavros, and S.-H. Teng, "Colocation Games: And Their Application to Distributed Resource Management," *Proc. Usenix Hot Topics in Cloud Computing*, 2009; article no. 10.
7. A. Ben-Yehuda et al., "Ginseng: Market-Driven Memory Allocation," *Proc. ACM Conf. Virtual Execution Environments*, 2014, pp. 41–52.
8. S. Sen et al., "Incentivizing Time-Shifting of Data: A Survey of Time-Dependent Pricing for Internet Access," *IEEE Comm.*, vol. 50, no. 11, 2012, pp. 91–99.

load on a hybrid medium consisting of multiple tiers,[7,8] with each tier comprised of HDDs, SSDs, or another storage device type. Data analytics applications are particularly amenable to such tiered storage deployments because of the inherent heterogeneity in workload I/O patterns. For example, I/O-intensive applications benefit from faster storage to a larger degree than computation-intensive applications. Hence, the choice of tiers depends on the composition of tenants' workloads and the performance benefits achieved by using specific tiers.[2] This choice isn't available in traditional tiering approaches that allocate "hot" and "cold" data on separate tiers.[2,7,8] To this end, we propose an innovative tiered object store that exposes this tiering control to tenants. Thus, the tenants can meet their price–performance objectives by partitioning their workloads to use different tiers based on their application characteristics. The cloud provider offers the tiers under dynamic pricing based on tenant's usage of each tier, enabling it to offset the increased management costs of faster tiers.

In this article, we argue that traditional HDD-based object stores are inefficient. First, for cloud tenants, an HDD-based object store can't effectively meet their requirements (for example, deadlines) due to the relatively slow I/O performance of HDDs. Second, for cloud providers, an HDD-only object store doesn't provide any pricing leverage, which reduces profitability. A faster tier can provide a higher quality of service (QoS), which can be strategically priced to increase profits. Hence, using game theory, we show that a hybrid HDD–SSD approach is desirable for both cloud providers and tenants.

Overview and Contributions

To verify our argument, we conducted a trace-driven simulation study by replaying two 250-job snippet traces from a larger set of traces from Facebook's production Hadoop cluster.[9] (For detailed performance parameters of different

```
Input: Job information matrix: L̂, Analytics
       job model matrix: M̂, Runtime configuration:
       R̂, Initial solution: P̂_init.
Output: Tiering plan P̂_best
begin
    P̂_best ← {}
    P̂_curr ← P̂_init
    exit ← False
    iter ← 1
    temp_curr ← temp_init
    U_curr ← Utility (M̂, L̂, P̂_init)
    While not exit do
        temp_curr ← Cooling(temp_curr)
        for next P̂_neighbor in AllNeighbors(L̂, P̂_curr) do
            if iter > iter_max then
                exit ← True
                break
            U_neighbor ← Utility (M̂, L̂, P̂_neighbor)
            P̂_best ← UpdateBest(P̂_neighbor, P̂_best)
            iter++
            if Accept(temp_curr, U_curr, U_neighbor) then
                P̂_curr ← P̂_neighbor
                U_curr ← U_neighbor
                break
    return P̂_best
```

Algorithm 1. Tiering solver.

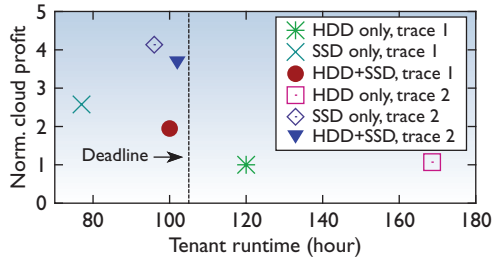

Figure 1. Tenant workload runtime for `trace 1` *and* `trace 2`*, and the provider's profits under different configurations. Cloud profits are normalized to that of* `HDD only, trace 1`*.*

applications please refer to our earlier work.[2]) We set the HDD tier price as \$0.0011/Gbytes/day — which is an average of the Google Cloud Storage price of \$0.00087/Gbytes/day and the Google Cloud's HDD persistent block storage price of \$0.0013/Gbytes/day — and the SSD tier price as \$0.0044/Gbytes/day (essentially 4× the HDD price). Note that we chose to use per-day pricing for our study, as our proposed pricing strategy also adjusts prices on the per-day granularity. We found that `trace 1` consumes 12 Tbytes of data and generates 4.7-Tbyte output, while `trace 2` consumes 18 Tbytes of data and generates 8.2-Tbyte output. For the hybrid storage tiering case (HDD+SSD), the tenant places jobs in different tiers with a desired workload completion deadline of 105 hours. For this purpose, we use Algorithm 1, which optimizes the tier allocation to meet the deadline while minimizing the cost. Our simulation assumes that the cloud provider has enough SSD capacity to satisfy the tenant's demand — that is, the tenant's SSD allocation isn't limited by the provider's SSD capacity.

Figure 1 shows the results, from which we make three observations. First, workloads with an HDD-only configuration can't meet the tenant-defined deadline and the cloud provider earns the lowest profit. Second, with the HDD+SSD tiering configuration, both workloads are able to meet the deadline, while the cloud provider sees significantly more profit (`trace 2` has larger input and output datasets, and hence, yields more profit). This is because the tenant places part of the workloads on the SSD tier, which is more expensive than the HDD tier. Third, the SSD-only configuration improves performance, but with marginally higher profit, compared to HDD+SSD. This is mainly due to HDD+SSD's tiering optimization. This experiment demonstrates that through object storage tiering, both the cloud provider and tenants can effectively achieve their goals.

Cloud providers have a multitude of device options available for deploying object storage infrastructure, including HDDs with different revolutions per minute (RPMs), serial advanced technology attachment (SATA), and peripheral component interconnect express (PCIe) SSDs, as well as the emerging storage-class memory (SCM) devices. These options offer different performance–price tradeoffs. For example, each device type offers a different cost per gigabyte and, with regard to SSDs and SCMs, different endurance. In a tiered setup comprising such a variety of storage choices, estimating price points while keeping maintenance costs under control is a challenge. For example, while the cloud providers might want to encourage tenants to use more SSDs to increase their profits, they run the risk of SSD wear-out earlier than expected, resulting in increasing management costs (for example, replacing SSDs) and decreasing overall profits.

To remedy this issue, we introduce a dynamic pricing model that providers can leverage to mitigate additional costs and increase overall operating profits for providers. The dynamic pricing model has two objectives: to balance the price increase and SSD wear-out rate by exploiting the tradeoff between high revenue versus high operational costs for high profit; and to provide an effective incentivizing mechanism to tenants so that tenants can meet their goals via object store tiering in a more cost-efficient fashion.

Generally, storage tiering has been looked at from just one entity's perspective. In contrast, we adopt a leader/follower game-theory model, where the objectives of the cloud provider and tenants are either disjoint or contradictory. We turn the pricing of object storage into a game, placing the cloud provider in the role of the leader, and letting the cloud provider decide prices for the offered storage tiers. The tenant follows this decision by deciding how much of each tier to use, such that its deadlines can be met with the minimum possible cost. The provider starts the next move by adjusting its prices based on the usage of each tier by the client in the previous move, in a way that maximizes its profit. With this work, we take the first step toward providing such a cloud-provider-driven game-theory-based pricing model through object storage tiering. Another unique aspect of our storage-tiering approach involves handling the lack of information available to the players (the cloud provider and tenants). For example, unlike private datacenters, cloud providers don't expose any information about the tiering setup to tenants. Thus, not only are the motivations different from the private deployments for providers and tenants in a public cloud, but they also have to make decisions based on partial information.

Specifically, we make two major contributions: First, we design a leader/follower gaming model with the goal of maximizing the cloud provider's profit. The provider makes the pricing decisions by estimating the tenants' storage capacity demand distribution among different tiers; driven by the prices, tenants employ a simulated annealing-based tiering solver to guide object storage tiering for maximizing their utility. Second, we demonstrate through trace-driven simulations that our novel object storage-tiering pricing mechanism can deliver increased profit for the cloud provider and potentially achieve a win-win scenario for both the provider and tenants.

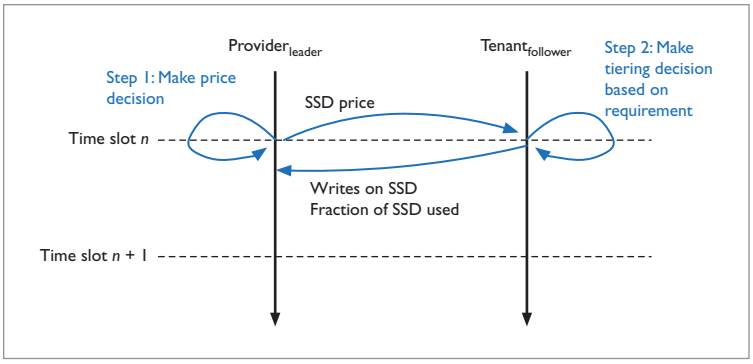

Figure 2. Illustration of the leader/follower game. The two entities repeat the same sequence of events on time n + *1.*

Model Design

We design a leader/follower cloud-pricing framework with the objective of maximizing a cloud provider's profit. In our model, the game is played in two steps (see Figure 2). First, the cloud provider (leader) makes the pricing decisions based on predictions about the tenants' demand on storage resources. Second, given prices of different storage resources, a tenant (follower) makes tiering decisions based on her own requirements, and the strategy is represented by the tenant's storage-tiering specification — thereby determining which jobs use what tier. (In our current tenant model, we assume two things: that tenants only make tiering decisions for the next time slot; and the adjustment for previously launched workloads due to price changes probably isn't necessary, because the workload-per-slot can be finished within that particular slot.)

While the tenants can see the provider's price changes, they're unaware of the actual reasons for the changes. Even if the tenants understood the reasons, multitenancy prevents modeling the provider's behavior. Hence, in our formulation tenants can only predict the provider's price movements based on historical data. Similarly, the provider is unaware of explicit tenant requirements, and only garners information from the requested storage capacity and the writes operations (`PUT` requests are tracked for accounting purposes). Thus, the provider also only uses historical information about these aspects to predict tenant demand. Consequently, both the tenants and provider model adopted in our game are purposefully "myopic" controls for predicting only the next time slot, and not beyond that in the future.

Provider Model

We model the provider cost as follows. Assuming that the fraction of cost that comes from SSDs' wear-out is $t < 1$ (we choose to use a fixed t here for simplicity, with the understanding that in the real world, there are numerous factors that come into play and t might not be a constant), the cost can be modeled as follows:

$$cost = \frac{1}{t} \cdot \frac{p_{ssd}}{endurance} \cdot w,$$

where p_{ssd} is the market price of one SSD, *endurance* is the endurance lifespan of the particular SSD, and w is the amount of data written to the SSD in gigabytes. The pricing decision-making process can be modeled as a nonlinear optimization problem that maximizes the profit defined as $profit = \sum_i (\sum_f (capacity_f \cdot f_{(i,f)} \cdot p_{(i,f)}) - cost_i)$, where i is the time unit index and f is the storage service. The details of the provider model can be found elsewhere.[10]

In a time slot n, we predict the SSD demand proportion for the next time slot $n + 1$ ($f_{(n+1,ssd)}$), which depends on the difference between the predicted SSD price for $n + 1$ and the calculated SSD price for n. The predicted HDD demand proportion is in turn determined by the total storage demand, subtracting the predicted SSD demand proportion.

The amount of data that will be written to SSDs is determined by the difference of predicted SSD demand proportion in time slot $n + 1$ to that in time slot n. If the SSD demand is predicted to increase, it implies that the amount of data that will be absorbed by the SSD tier will also increase. Our provider model also defines the SSD tier data-writing constraint, which ensures that the expected amount of data written to the SSD tier won't exceed the threshold that's calculated, based on accumulated historical statistics. The factor indirectly controls the value adaptation of decision variables $p_{(n,b)}$ and $p'_{(n+1,b)}$. We assume HDD prices $p_{(n,a)}$ and $p_{(n+1,a)}$ are fixed, and SSD prices are constrained in a predefined range. We plan to include I/O operations per second (IOPS) per client in our future pricing models.

Tenant Model

The data placement and storage provisioning at the tenant side is modeled as a nonlinear optimization problem as well. The goal is to maximize *tenant utility*, which is defined as $utility = 1/(T \cdot \$)$, where T is the total workload runtime and \$ is the total monetary cost. The tenant utility combines the workload performance and monetary cost, the two factors that tenants are most concerned with, into a single function.

The performance of the tenant's workload is modeled as the reciprocal of the estimated completion time in minutes ($1/T$) and the cost is mainly storage. The cost of each storage service is determined by the workload completion time (storage cost is charged on an hourly basis) and capacity provisioned for that service. The overall storage cost is obtained by aggregating the individual costs of each tier in the object store. A *capacity constraint* is required to ensure that the storage capacity provisioned for a job is sufficient to meet its requirements for all the workload phases (map, shuffle, and reduce). Given a specific tiering solution, the estimated total completion time of the workload is constrained by a tenant-defined deadline. A price predictor at the tenant side is needed, the value of which can also be supplied as a hint by the cloud provider.

We devise a simulated annealing-based algorithm[2] (Algorithm 1) for computing tenants' data partitioning and job placement plans. The algorithm takes as input workload information ($\hat{\mathsf{L}}$), computing cluster configuration ($\hat{\mathsf{R}}$), and information about performance of analytics applications on different storage services ($\hat{\mathsf{M}}$). $\hat{\mathsf{P}}_{init}$ serves as the initial tiering solution that's used to specify preferred regions in the search space. The results from a simple greedy algorithm based on the characteristics of analytics applications can be used to devise an initial placement. The details of the tenant model can be found elsewhere.[2,11]

Our algorithm's main goal is to find a near-optimal tiering plan for a given workload. In each iteration, we pick a randomly selected neighbor of the current solution (*AllNeighbors*(.)). If the selected neighbor yields better utility, it becomes the current best solution. Otherwise, in the function *Accept*(.), we decide whether to move the search space toward the neighbor $\hat{\mathsf{P}}_{init}$ or keep it around the current solution ($\hat{\mathsf{P}}_{neighbor}$). This is achieved by considering the difference between the utility of the current (U_{curr}) and neighbor solutions ($U_{neighbor}$) and comparing it with a distance parameter, represented by $temp_{curr}$. In each iteration, the distance parameter is adjusted (decreased) by a *Cooling*(.) function. This helps in making the

search narrower as iterations increase; reducing the probability of missing the maximum utility in the neighborhood of the search space. We choose to implement the tenant-side tiering using a coarse-grained job-level approach for the following reason. The absence of task-level tier-aware scheduling mechanisms implies that these Hadoop Distributed File Systems (HDFS) block granularity tiering approaches can't avoid stragglers within a job, thus achieving limited performance gains, if any. Integrating a tier-aware, intelligent, task-level scheduler is complementary to our work.

Evaluation

We used trace-driven simulations to demonstrate how our cloud–tenant interaction models perform in practice. We use the production traces collected from a 3,000-machine Hadoop deployment at Facebook.[9] The original traces consist of 25,428 Hadoop jobs; we chose to use a snippet of 1,750 jobs that span a seven-day period. The workload runs on a cloud object store with a built-in tiering mechanism that tenants can control. We set the time slot for our models to one day. The traces we used were collected from a multiuser (tenant) corporation cluster. Thus, our evaluation captures multitenancy behaviors in a simplified approach, where all tenants' goals are to get the best return for their workload deployment.

We assign to our workload, in a round-robin fashion, four representative analytics applications that are typical components of real-world analytics[9,12] and exhibit diversified I/O and computation characteristics.[2] `Sort`, `Join`, and `Grep` are I/O-intensive applications. The execution time of `Sort` is dominated by the shuffle phase I/O. `Grep` spends most of its runtime in the map phase I/O, reading input and finding records that match given patterns. `Join` represents an analytic query that combines rows from multiple tables and performs the join operation during the reduce phase, making it reduce-intensive. `KMeans` is an iterative CPU-intensive clustering application that expends most of its time in the compute phases of map and reduce iterations.

Figure 3 shows price variation by the cloud provider's model based on the amount of data written by the tenant to the SSD tier on a per-day basis. The HDD price is fixed, while the SSD price is dynamically adjusted on a daily basis for `dynamic` pricing (the pricing is the same as

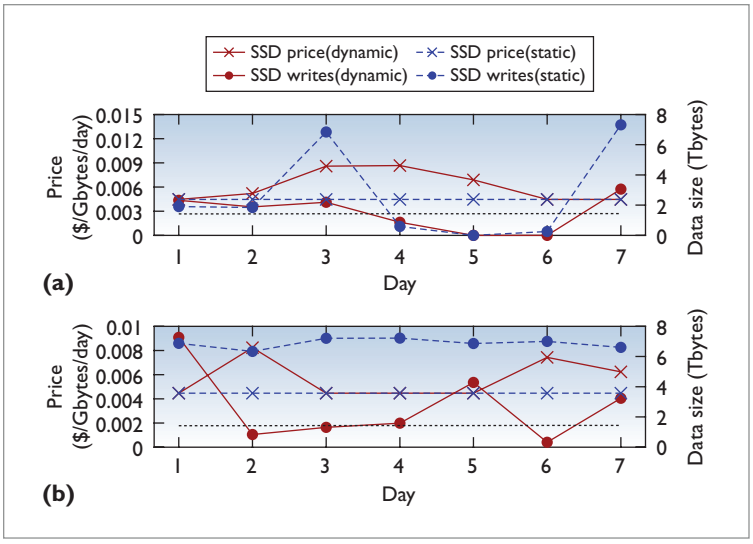

Figure 3. Dynamic pricing by the provider and the tenant's response for two 7-day workloads. The dotted horizontal line represents the daily write limit L_i. The variation in L_i is too small to be discernible at this scale. (a) A seven-day trace. (b) A single-day trace repeated seven times.

for Figure 1). Under `static` pricing, the provider sets a static price and tenants periodically run Algorithm 1, whereas under `dynamic` pricing, the provider and tenants interact. The per-day write limit L_n is dynamically adjusted based on the amount of writes from the tenant side (though not discernible in the figure). Figure 3a shows the price changes for a seven-day trace, with a different workload running on each day. We observe that as the amount of writes by the tenant on the SSD tier increases above the write limit, the cloud provider begins to adjust the SSD price. The tenant's model will adjust the tiering strategy to either put more data in the HDD tier or pay more for the SSD tier in the case of a strict deadline requirement, because each day has a different workload, and hence, a different deadline. For example, from day 4, the tenant — with the goal of maximizing the tenant utility — allocates fewer jobs to the SSD tier. Once the SSD writes are reduced below the threshold, the provider lowers the SSD tier price to incentivize the tenant to use more of the SSD resource on the next day. The tenant responds to this change on day 7, where the workload deadline is stricter than the previous two days.

Figure 3b shows the price changes for a seven-day period with the same single-day trace replayed every day. This trace shows stronger

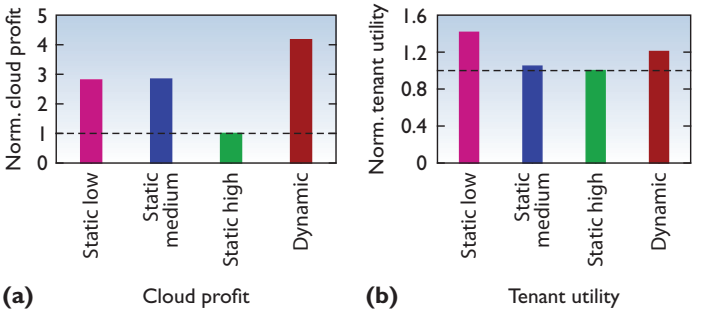

Figure 4. Cloud profit and tenant utility averaged over seven days. Here, `static low`, `static medium`, and `static high` mean a low, medium, and high static solid-state drive (SSD) price, respectively. Cloud profit and tenant utility are normalized with regard to `static high`. Results are normalized to that of `static high`. (a) Cloud profit. (b) Tenant utility.

correlation between per-day SSD writes from the tenant and the SSD price from the provider. This workload exhibits the same specifications every day (such as dataset size or a relaxed deadline), and thus the daily writes on the SSD tier remain stable under static pricing. However, the dynamic pricing model can effectively respond to the spike in the amount of writes on the first day and adjust the SSD price accordingly. Given the increased SSD price, the tenant tries to reduce their monetary cost by migrating more jobs to the cheaper HDD tier while still meeting the deadline. This, in turn, is implicitly controlled by the cloud provider's prediction model. The provider is aware of the total storage demand (behavior patterns) from the tenant (this is why we chose to repeat a single-day trace) and hence, by adjusting SSD pricing, the overall writes to the SSD tier is maintained within a reasonably smaller range (which is as desired by the provider, but won't break the relatively relaxed deadline constraint imposed by the tenant).

When the provider lowers the SSD price in response, the tenant increases their use of SSD, prompting the provider to increase the SSD price again. This tenant-provider interaction results in an average of 2.7 Tbytes per day in SSD writes compared to an average of 7 Tbytes per day under static pricing (with no deadline miss rate for both cases). The test demonstrates that our dynamic pricing model can adaptively adjust the SSD pricing based on the amount of data written to the SSD tier to maintain high profits while keeping the SSD wear-out under control by keeping write loads to SSD in check.

In our next test, we examine the impact of different SSD pricing models on provider profit and tenant utility — that is, the cloud-tenant interaction (Figure 4 shows the results). We choose three static prices for SSDs: low (the minimum SSD price that we use is \$0.0035/Gbytes/day), medium (\$0.0082/Gbytes/day), and high (the maximum SSD price, \$0.0121/Gbytes/day). We also compare static prices with our dynamic pricing model. As Figure 4a shows, dynamic pricing yields the highest provider profit, because it increases the price based on SSD writes from the tenant. Both `static low` and `medium high` yield similar profits that are 32.6 percent lower than that gained under dynamic pricing. This is because `static medium` results in more jobs placed on the HDD tier, which lowers the tenant cost while causing a longer workload completion time. Meanwhile, `static low` isn't able to generate enough profit due to the low price, while under `static high` the tenant solely migrates all the jobs to the HDD tier, also resulting in low profit.

Next, we examine Figure 4b's tenant utility. With a `static low` SSD price, the tenant utility is 17.1 percent higher than that achieved under dynamic pricing. However, this would result in a significantly shortened SSD lifetime (by as much as 76.8 percent), thereby hurting the cloud profit in the long term. With the `static high` SSD price, the tenant utility is reduced by 17.1 percent compared to that of dynamic pricing, as the tenant shifts most jobs to the HDD tier. The `static medium` SSD price yields a slightly higher tenant utility as compared to `static high`, but still 13.1 percent lower than that seen under dynamic pricing. This is because the tenant must assign some jobs to the faster SSD tier to guarantee that the workload doesn't run for too long. Dynamic pricing, on the other hand, maintains the tenant utility at a reasonably high level (higher than both `static medium` and `static high`, but slightly lower than `static low`), while guaranteeing that the SSD lifetime constraints are met. This demonstrates that our dynamic pricing model can effectively achieve a win-win for both the cloud provider and tenants.

We show that by combining dynamic pricing with cloud object storage tiering, cloud providers can increase their profits while meet-

ing the SSD wear-out requirements, and tenants can effectively achieve their goals with a reasonably high utility. We demonstrate this win-win situation via real-world trace-drive simulations. For future work, we aim to refine our prediction model and study its long-term effect. We also plan to explore different tiering algorithms and the best dynamic pricing models in multitenant clouds.

References

1. D. Narayanan et al., "Migrating Server Storage to SSDs: Analysis of Tradeoffs," *Proc. ACM European Conf. Computer Systems*, 2009, pp. 145–158.
2. Y. Cheng et al., "Cast: Tiering Storage for Data Analytics in the Cloud," *Proc. ACM Int'l Symp. High-Performance Parallel and Distributed Computing*, 2015, pp. 45–56.
3. T. Harter et al., "Analysis of HDFS under HBase: A Facebook Messages Case Study," *Proc. Usenix Conf. File and Store Technologies*, 2014; www.usenix.org/node/179859.
4. L.M. Grupp, J.D. Davis, and S. Swanson, "The Bleak Future of NAND Flash Memory," *Proc. Usenix Conf. File and Store Technologies*, 2012; http://cseweb.ucsd.edu/~swanson/papers/FAST2012BleakFlash.pdf.
5. Y. Lu, J. Shu, and W. Zheng, "Extending the Lifetime of Flash-Based Storage through Reducing Write Amplification from File Systems," *Proc. Usenix Conf. File and Store Technologies*, 2013; www.usenix.org/system/files/conference/fast13/fast13-final110.pdf.
6. G. Wu and X. He, "Delta-FTL: Improving SSD Lifetime via Exploiting Content Locality," *Proc. ACM European Conf. Computer Systems*, 2012, pp. 253–266.
7. H. Kim et al., "Evaluating Phase Change Memory for Enterprise Storage Systems: A Study of Caching and Tiering Approaches," *Proc. Usenix Conf. File and Store Technologies*, 2014; www.usenix.org/node/179827.
8. Z. Li, A. Mukker, and E. Zadok, "On the Importance of Evaluating Storage Systems' $Costs," *Proc. Usenix Hot Topics in Storage and File Systems*, 2014; www.usenix.org/node/183614.
9. Y. Chen, S. Alspaugh, and R. Katz, "Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads," *Proc. Very Large Database Endowment*, vol. 5, no. 12, pp. 1802–1813.
10. Y. Cheng et al., "Pricing Games for Hybrid Object Stores in the Cloud: Provider vs. Tenant," *Proc. Usenix Hot Topics in Cloud Computing*, 2015; www.usenix.org/conference/hotcloud15/workshop-program/presentation/cheng.
11. V. Jalaparti et al., "Bridging the Tenant-Provider Gap in Cloud Services," *Proc. ACM Symp. Cloud Computing*, 2012, article no. 10.
12. M. Zaharia et al., "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," *Proc. ACM European Conf. Computer Systems*, 2010, pp. 265–278.

Yue Cheng is a PhD student in the Computer Science Department at Virginia Tech. His research interests include distributed storage systems, especially key-value/object stores and caches, storage cache and tier management, and NAND flash SSDs. Cheng has a BS in computer science from Beijing University of Posts and Telecommunications, Beijing, China. He's a member of ACM. Contact him at yuec@vt.edu.

M. Safdar Iqbal is a PhD student in the Computer Science Department at Virginia Tech. His research interests include distributed storage systems, especially storage tier management for cloud and Big Data applications and multilevel cache management for storage-class memory devices. Iqbal has a BS in computer science from Lahore University of Management Sciences, Lahore, Pakistan. Contact him at safdar@vt.edu.

Aayush Gupta is a research staff member at IBM Research–Almaden. His research interests include distributed storage systems, especially in-memory caches and databases, cloud economics, and emerging non-volatile memory devices. Gupta has a PhD in computer science and engineering from Pennsylvania State University. He received a best paper runner-up award at ACM e-Energy 2015. Contact him at guptaaa@us.ibm.com.

Ali R. Butt is an associate professor of computer science at Virginia Tech. His research interests include experimental computer systems, especially in cloud computing, I/O systems, and data-intensive high-performance computing. Butt has a PhD in electrical and computer engineering from Purdue University. He is a recipient of the US National Science Foundation CAREER Award (2008), IBM Faculty Award (2008, 2009, and 2015), Virginia Tech College of Engineering "Outstanding New Assistant Professor" Award (2009), best paper award (MASCOTS 2009), and NetApp Faculty Fellowships (2011 and 2015). He's a member of Usenix and the American Society for Engineering Education, and a senior member of ACM and IEEE. Contact him at butta@cs.vt.edu.