Fighting with Unknowns: Estimating the Performance of Scalable Distributed Storage Systems with Minimal Measurement Data

Moo-Ryong Ra and Hee Won Lee
AT&T Labs Research
{mra, knowpd}@research.att.com

***Abstract*—Constructing an accurate performance model for distributed storage systems has been identified as a very difficult problem. Researchers in this area either come up with an involved mathematical model specifically tailored to a target storage system or treat each storage system as a black box and apply machine learning techniques to predict the performance. Both approaches involve a significant amount of efforts and data collection processes, which often take a prohibited amount of time to be applied to real world scenarios. In this paper, we propose a simple, yet accurate, performance estimation technique for scalable distributed storage systems. We claim that the total processing capability per IO size is conserved across a different mix of read/write ratios and IO sizes. Based on the hypothesis, we construct a performance model which can be used to estimate the performance of an arbitrarily mixed IO workload. The proposed technique requires only a couple of measurement points per IO size in order to provide accurate performance estimation. Our preliminary results are very promising. Based on two widely-used distributed storage systems (i.e., Ceph and Swift) under a different cluster configuration, we show that the total processing capability per IO size indeed remains constant. As a result, our technique was able to provide accurate prediction results.**

***Index Terms*—performance modeling, distributed storage systems, measurement**

I. Introduction

Modern distributed storage systems provide a rich set of features and configuration options along with horizontal scalability. The users of such storage systems will observe a linear increase in performance as more hardware resources (i.e., hosts and disks) are added to the infrastructure. Having any centralized components in its datapath can hamper the desirable level of scalability. Thus, such systems exploit a form of consistent hashing mechanism. Good examples include CRUSH algorithm [1] for Ceph and distributed hash table (DHT) for Swift [2] and Cassandra [3]. The essential role of these algorithms is to distribute the load evenly across the available hardware resources (load balancing) so that we can eliminate performance bottleneck. In addition, each distributed storage system provides a different guarantee to users. For instance, Ceph provides strong consistency for write operations while Swift/Cassandra have eventual consistency semantics for the same operation. Moreover, many such systems provide an option to enable erasure coding to have a reduced storage space with the same level of reliability. Some storage systems offer multiple storage APIs while others focus on providing a single API[1]. All these characteristics and features complicates the overall dynamics of distributed storage systems.

Due to the complexity discussed so far, the performance modeling of such systems has been identified as a difficult problem. Unless one would like to go through an exhaustive measurement effort to cover the whole parameter space as in [4], researchers in this area either construct a fairly involved mathematical model specifically tailored to a target storage system [5], [6] or treat each system as a black box and apply machine learning techniques to predict the performance [7], [8]. Both approaches involve a significant amount of efforts and data collection processes, which often take a prohibited amount of time to apply to real world scenarios.

In this paper, we propose a simple, yet accurate, performance estimation technique for scalable distributed storage systems. Specifically, our technique aims to help capacity planning processes, e.g., identifying max IOPS for an arbitrary read/write ratio with a minimal evaluation process. Our technique is motivated by the fact that one of the main design goals of modern distributed storage systems is to distribute the workload as evenly as possible across all available compute resources. Thus, our technique is developed for a class of distributed storage systems that do not have any centralized components such as metadata servers in their datapaths. With a sufficient scale, the host or disk specific behavior is amortized by the aggregation at a higher level. Based on the observation, we claim that the aggregated processing capability of a whole distributed storage system remains unchanged for a given IO size. In our preliminary evaluation, we present a promising result describing that the total processing power of the system is indeed a constant per IO size, thereby demonstrating that we can get accurate estimation results with our proposed technique. Our evaluation results based on two popular distributed storage systems (i.e., Ceph and Swift) exhibit reasonably accurate results for small IOs ($< 10\%$) and still remain reasonable for larger IOs ($5\%\sim25\%$). The technique also shows similar accuracy for mixed IO sizes (Sec. III).

[1]Many commercial object storage system supports Amazon S3 API, whereas a system like Ceph supports multiple APIs, e.g., block, file system, and object by adding additional layer.

II. Our approach

As discussed, modern distributed storage systems emphasize horizontal scalability. In other words, adding more disks and/or nodes will linearly increase the storage capacity and IO performance of the entire storage system. A primary design principle of such systems include a) no single point of failure and b) automatic load balancing enabled by placing data objects as evenly as possible across all available hardware. Because of this nature, a well-designed distributed system often shows reasonably stable performance if its configuration is unchanged and the workload is stable. Hence, we make the following claim.

Claim: If HW/SW/workload settings remain unchanged, the total processing capability of a distributed storage system is invariant for a given IO size.

Now we will elaborate what this statement actually means. Let U_{read} and U_{write} be a unit work for the system to perform read and write operation respectively. Then, for a mixed workload with an arbitrary read/write ratio, the claim can be re-written as follows:

$$B = U_{read} \cdot T_{read} + U_{write} \cdot T_{write} \quad (1)$$

$$U_{write} = U_{read} \cdot f_{rw} \quad (2)$$

where B is a constant, and T_{read} and T_{write} are read and write performance (ops/s), respectively. f_{rw} is a coefficient that reflects the load difference between read and write. If we combine the two equations, we get the following result:

$$C = T_{read} + T_{write} \cdot f_{rw}. \quad (3)$$

Again, C is a constant ($\frac{B}{U_{read}}$) for a given IO size (object/block size). With this form, we can acquire f_{rw} value with just two data points, i.e., the measured IOPS performance of the systems when all of the IOs are read (100% read ops/s) and write (100% write ops/s). In detail, with 100% read IOs, Eq. 3 becomes $C = T_{100\%read}$. With 100% write, the equation becomes $C = T_{100\%write} \cdot f_{rw}$, which in turn can be re-written to:

$$f_{rw} = \frac{T_{100\%read}}{T_{100\%write}}. \quad (4)$$

At this point, it is worth noting that f_{rw} changes along with IO size. We show our experimental results in Sec. III.

A. IO workload with arbitrary read/write ratio

With our hypothesis, it is fairly straight-forward to estimate the performance of a mixed workload with an arbitrary read/write ratio. Suppose that we have a target workload specification: the ratio of R_{read} to R_{write} where $R_{read} + R_{write} = 100(\%)$. Then, IOPS for read and write operations can be written as $k \cdot R_{read}$ and $k \cdot R_{write} = k \cdot \{100 - R_{read}\}$ where k is a constant.

We plug these terms into Eq. 3 and get the following:

$$k \cdot R_{read} + k \cdot R_{write} \cdot f_{rw} = C \quad (5)$$

where constant C can be estimated using 100% read IOPS (i.e., $C = T_{100\%read}$). If we solve this new equation with respect to k, then we get the following close-form solution:

$$k = \frac{T_{100\%read}}{R_{read} + \{100 - R_{read}\} \cdot f_{rw}}. \quad (6)$$

Since we can obtain f_{rw} from Eq. 4, once we get the value of k, it is trivial to calculate $T_{read} = k \cdot R_{read}$ and $T_{write} = k \cdot R_{write}$.

B. Extension to support mixed IO sizes

Another dimension often occurred in a real world scenario is to have a workload with mixed IO sizes. Our model can be extended to support mixed IO sizes (e.g., multiple object sizes or multiple block sizes). Suppose that we have heterogeneous object sizes in our target workload, $S_1, S_2, \cdots, S_N$ and know the exact proportion of each object size to the total objects, $P_1, P_2, \cdots, P_N$ where $\sum_{i=0}^{N} P_i = 1$.

Now the total processing capability per object size (S_N) becomes a fraction of the whole workload that is proportional to P_N, i.e., $P_N \cdot C^{S_N}$. If we plug the result into Eq. 6, we can get an array of equations as follows.

$$\bar{k}^{S_1} = \frac{P_1 \cdot T^{S_1}_{100\%read}}{R_{read} + \{100 - R_{read}\} \cdot f^{S_1}_{rw}} = P_1 \cdot k^{S_1}$$

$$\vdots$$

$$\bar{k}^{S_N} = \frac{P_N \cdot T^{S_N}_{100\%read}}{R_{read} + \{100 - R_{read}\} \cdot f^{S_N}_{rw}} = P_N \cdot k^{S_N}.$$

Just multiplying known read/write ratios with each $\bar{k}^{S_1} ... \bar{k}^{S_N}$ will result in read and write IOPS for each IO size. Thus, more formally, total IOPS ($= T_{total}$) can be calculated using the following equation:

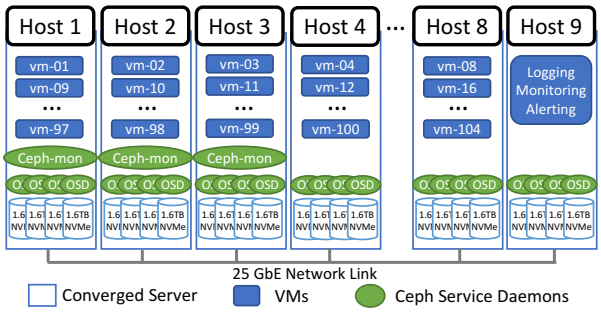
$$T_{total} = \sum_{i=1}^{N} \{R^{S_i}_{read} + R^{S_i}_{write}\} \cdot \bar{k}^{S_i} = 100 \cdot \sum_{i=1}^{N} P_i \cdot k^{S_i}. \quad (7)$$

III. Evaluation

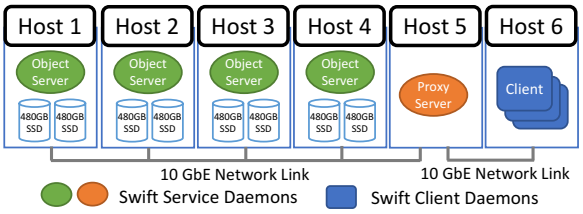
A. Methodology

The main purpose of our evaluation effort is twofold. First, we empirically verify the correctness of our model described in Sec. II. Second, we show the estimated performance numbers are indeed accurate.

To prove our claim, we exhaustively run customized workload profiles covering all necessary IO parameters, e.g., all combinations of interested IO (block or object) sizes, read-/write ratios and mixed IO sizes. For estimation, we used 100% read and 100% write data to calculate the total processing capability of a given IO size (C^S). We then compared the measured results with the estimated performance numbers. In this preliminary evaluation effort, our focus is given to one



(a) Ceph cluster layout



(b) Swift cluster layout

Fig. 1. Experimental Setup

performance metric, i.e., IOPS (ops/s). Lastly, we use the following equation to present an estimation error:

$$Error = \frac{|estimated - measured|}{measured} \times 100 \quad (8)$$

where the value of *measured* is the average performance (i.e., IOPS) during an experimental run[2].

B. Cluster configuration: Ceph and Swift

We use two representative open source distributed storage systems: Ceph [9] and Swift [2]. Each storage system is deployed on a different cluster. Client location per cluster was different. Ceph cluster is in a hyper-converged setting where client VMs are running together with Ceph storage systems. In case of Swift cluster, client hosts are physically separated from the Swift cluster.

Ceph cluster: Figure 1(a) illustrates a Ceph cluster configuration that we used for our experiments. The setup is a hyper-converged cloud deployment with all flash storage where Ceph is providing virtual disks to VMs orchestrated by OpenStack [10]. We use 9 Dell R730xd nodes for the Ceph storage cluster and each node has two Xeon CPU E5-2695 v4 2.10GHz, 256GB memory, 4 NVMe SSDs (SM1715 1.6TB), and 25G Mellanox CX-4 Lx NIC[3]. Storage pools for the experiments are configured as 3x replication. Clients in 104 VMs run fio benchmark tool [11] and consume the Ceph storage via block device interface on guest OS, which is eventually connected to a host-side RBD (Rados Block Device) interface. Each VM's fio process had 8 jobs with ioengine of libaio, iodepth of 32, filesize of 100GB when running with different block sizes and read/write ratios.

Swift cluster: As a second environment, we construct an OpenStack Swift [2] cluster composed of five nodes and one client node, as shown in Fig. 1(b). For each node, we use a Dell R720xd server equipped with two CPUs (Intel Xeon E5-2630L 2.00GHz), 128GB RAM, two 10GB Ethernet NICs, and two SSDs (SM863 480GB). The proxy server is deployed in one node. For each IO request from a client, it will look up the location of the account, container, or object in the ring and route the request accordingly. The Swift object servers are deployed in four nodes, each of which stores, retrieves and deletes objects on its local devices. We build an object ring with 3x replication using eight SSDs – each with an equal weight – distributed over the object servers. The client node runs COSBench object storage benchmark tool [12] to generate custom IO workloads. We used 32 workers for all COSBench runs. We prepared 16,000 objects before we run experiments (500 objects in each container and a total of 32 containers). When reading and writing objects for data collection, we randomly choose an identifier among 32,000∼64,000 objects across 32 containers.

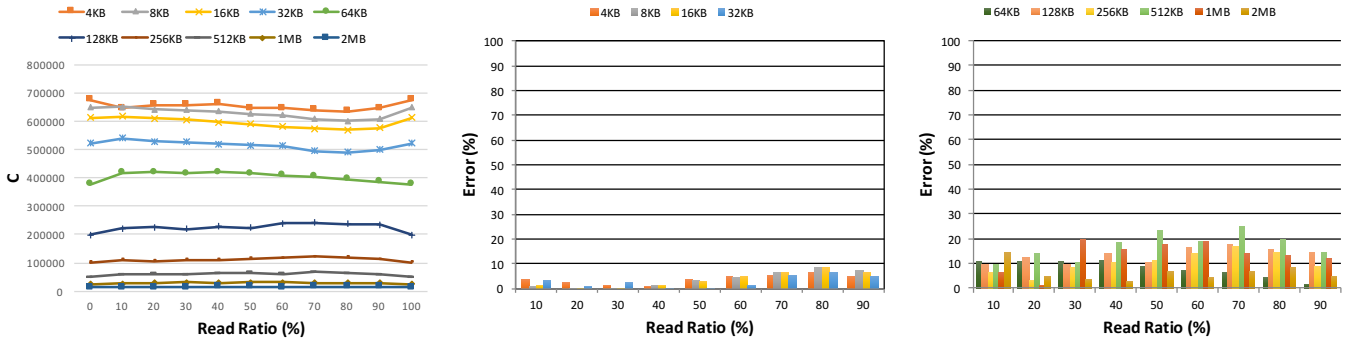
C. Results

a) Verifying the hypothesis:: To verify our hypothesis made in Sec. II, we first calculate $f_{rw} = \frac{T_{100\%read}}{T_{100\%write}}$ for each block/object size based on measured data where $T_{100\%read}$ is 100% read IOPS and $T_{100\%write}$ is 100% write IOPS. Fig. 3 shows the result. The results are very well-aligned with our hypothesis that the total processing capacity is constant across different read/write ratio per block size (C^S). Perhaps not surprisingly, f_{rw} values vary a lot along with block/object sizes and underlying distributed storage system implementation. For example, for Ceph, a range of f_{rw} was 3∼9 depending on the block size. In case of Swift, however, the range was 2∼3 for different object sizes. We explain that this difference is mainly caused by each system's consistency guarantee. Specifically, Ceph provides strong consistency. In our environment, therefore, a write request will be acknowledged after all three copies are fully-written. In contrast, Swift uses eventual consistency, so write operations (with respect to reads) are relatively cheaper than Ceph's case even with the same level of data redundancy (i.e., 3x replication for both Ceph and Swift experiments).

Arbitrary read/write ratio: Once we get f_{rw} per IO size, we plug them into Eq. (6) to get an estimated performance (see Sec II-A). After we get the estimated values, we compare them with the measured data. For the Ceph cluster, Fig. 2(a) shows that C values are approximately constant per IO size regardless of read/write ratios, and Fig. 2(b) and 2(c) show the accuracy of the proposed scheme. For the Swift cluster, Fig. 2(d) shows that C values are also approximately constant per IO size regardless of read/write ratios, and Fig. 2(e) and 2(f) show the accuracy of the proposed scheme. The results are very encouraging. Even though we use a completely different

[2]5 minute duration per run

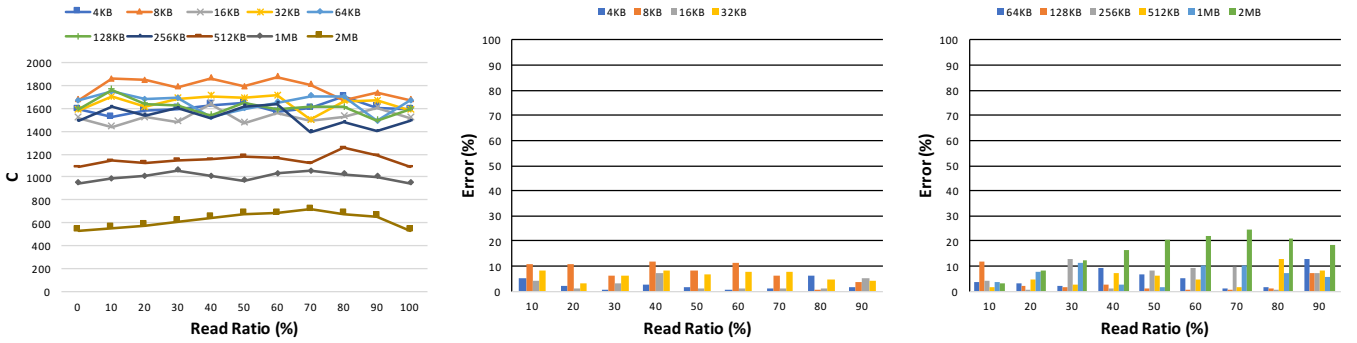
[3]One 25G port is enabled during the experiments.



(a) Ceph: C values

(b) Ceph: IOPS performance estimation errors for small IOs

(c) Ceph: IOPS performance estimation errors for large IOs



(d) Swift: C values

(e) Swift: IOPS performance estimation errors for small IOs

(f) Swift: IOPS performance estimation errors for large IOs

Fig. 2. C values and estimated performance: in x-axis, read ratio 20 means 20% read and 80% write.

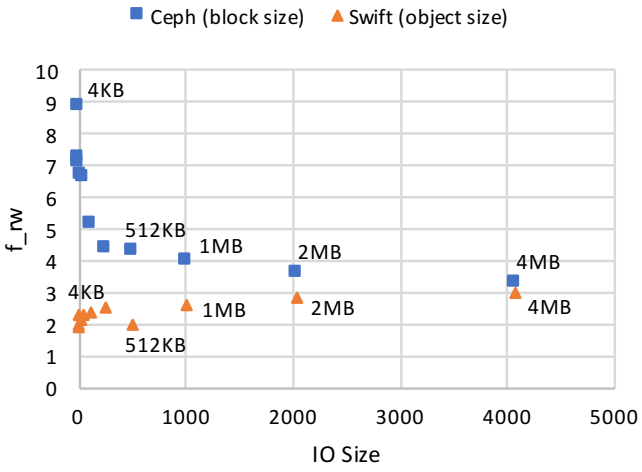


Fig. 3. f_{rw} ($= \frac{T_{100\%read}}{T_{100\%write}}$) reflects the load difference between read and write IO operation. The amount of work required for read and write operations can be very different per storage system implementation and their configurations.

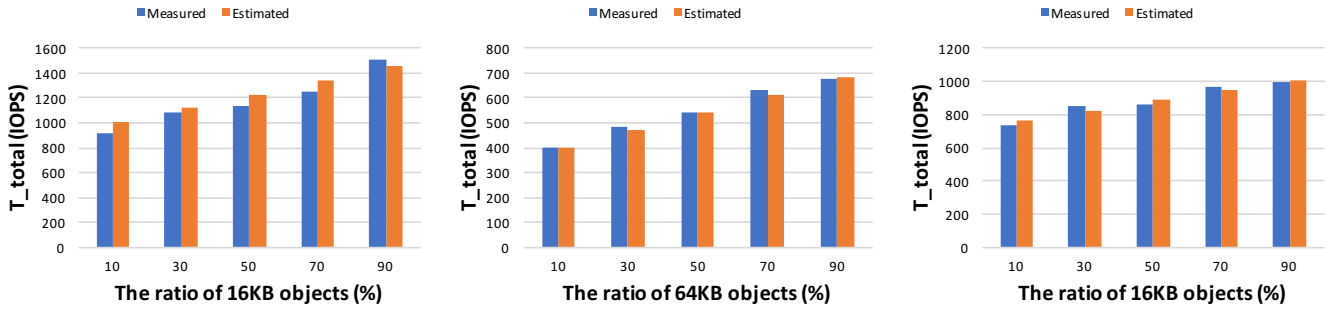
cluster and storage systems, for the block/object sizes equal or less than 32KB, the accuracy is ∼90% or above. Similarly for larger block/object sizes, most of the time the estimation accuracy is 80% or higher. The reason for having a little lower accuracy for larger block size could be the difference in IO size between the application generating IOs and physical medium underneath. A typical sector size of disk drives is 512 bytes in Linux and max IO size from the perspective of block subsystem is less than the object sizes we tried in our evaluation, e.g., hundreds of KB range in our corporate environment. Due to this mismatch, the operating system needs to do more IO operations to physical media than the number of IOs issued by applications, and consequently we get more variance in the estimation results.

In summary, it is worth emphasizing again that our proposed technique requires only two data points (i.e., 100% read IOPS and 100% write IOPS) to get the f_{rw} value. After getting f_{rw}, we will be able to get a fairly accurate estimate for the performance of a mixed read/write IO workload.

Mixed IO sizes: We evaluated how well our model in Sec. II-B can predict total IOPS when the workload has multiple IO sizes. We used the Swift cluster for this evaluation item. From a client, we generate IO workloads with the following mixes:

- case 1: 16KB 100% reads and 1MB 100% reads
- case 2: 64KB 100% writes and 1MB 100% writes
- case 3: 16KB 50/50% r/w and 512KB 50/50% r/w

To estimate total IOPS ($= T_{total}$ in Eq. 7), for each object size 16KB, 64KB, 512KB, and 1MB, we first calculate f_{rw}^{16KB}, f_{rw}^{64KB}, f_{rw}^{512KB}, and f_{rw}^{1MB} by using $f_{rw} = \frac{T_{100\%read}}{T_{100\%write}}$



(a) 16KB Reads + 1MB Reads (b) 64KB Writes + 1MB Writes (c) 16KB R/Ws + 512KB R/Ws

Fig. 4. IO workloads with mixed object sizes using COSBench on the Swift cluster

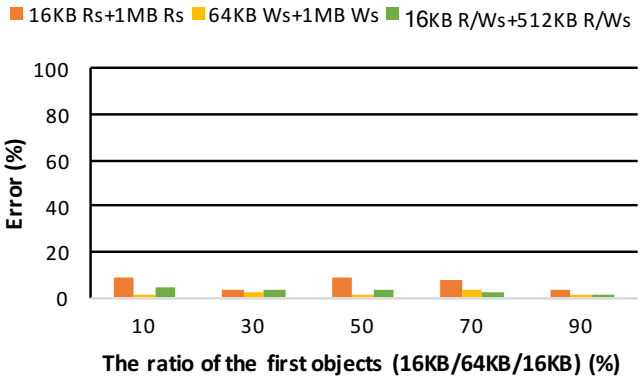


Fig. 5. Estimation Error

based on measured data. We then calculate k^{S_i} for each object size S_i. For example, for 16KB objects $k^{16KB} = \frac{T^{16KB}_{100\%read}}{R_{read}+\{100-R_{read}\}\cdot f^{16KB}_{rw}}$. In case 1, for 16KB reads, $R_{read} = 100$, so $k^{16KB} = \frac{T^{16KB}_{100\%read}}{100}$. In case 3, for 16KB 50% reads, $k^{16KB} = \frac{T^{16KB}_{100\%read}}{50+50\cdot f^{16KB}_{rw}}$.

In each case, we use 10/90, 30/70, 50/50, 70/30, and 90/10 for the ratio of the first IO sizes (i.e., 16KB, 64KB, and 16KB) to the second IO sizes (i.e., 1MB, 1MB, and 512KB), respectively. For example, in case 1, for $P_1 = 0.1$ and $P_2 = 0.9$, we can calculate total IOPS by $T_{total} = 100\{0.1 \cdot k^{16KB} + 0.9 \cdot k^{1MB}\}$. In a similar fashion, we calculate total IOPS for all three cases.

The results depicted in Fig. 4 were surprisingly good! Overall, the errors between estimated and measured total IOPS are less than 9%.

IV. Scope and Limitations of our Approach

Our evaluation results are very promising as shown in Sec. III and we expect the technique to greatly reduce the amount of effort and time for some practical tasks, e.g., capacity planning for production deployment. In order to understand the applicable scope of our proposed technique, we discuss the limitations of our technique in this section.

First, it is worth noting that our technique relies on a linear behavior of the underlying distributed storage system. So if the system behavior becomes non-linear, our technique is unlikely to work well or we may need to re-sample required measurement points again to maintain accuracy. A good example is to incorporate an impact of performance interference under a heavily loaded environment. Our technique is not intended to solve this loaded scenario. A summary of this research area can be found in our prior work [13]. Aging issues of storage hardware [14], [15] and/or file system [16], [17] can be mitigated by re-sampling, i.e., running 100% read/write for the interested IO sizes, if performance degradation does not cause too much noisy behavior.

Second, our work is not tailored to any specific workload that might have some dependencies among IOs. There is a large body of work whose design goal is to optimize the storage system so as to best exploit the temporal/spatial locality of a given workload often with a sort of caching mechanism. Our work, however, aims to provide a baseline performance cap for generic cloud environments where all types of workloads can co-exist. Moreover, note that our technique's target storage systems are highly distributed by design, so most workloads become random IOs when each request reaches the underlying medium.

V. Related Work

Performance modeling of storage systems are extensively studied in the past couple of decades. Due to the space constraint, we cover only a part of such research activities. Compared to most work in this space, our technique is much simpler to apply for a real world scenario, while potentially achieving accurate estimation results in many use cases.

Analytical modeling: If one has a deep domain knowledge of a certain system, he could construct a model to capture every single details of the dynamics. Examples of such efforts are as follows. Shriver et al. [5] models disk performance in the presence of caches and IO reordering. Kelly et al. [18] devises a probabilistic model to estimate IO request latency and verify the model with simulation. BASIL [19] models a linear region of disk-based storage systems for load balancing tasks in order to aid live migration tasks of virtual hard disks.

IRONModel [6] identifies that a model-based approach is brittle in real world, and provides a way to localize the problem derived from the redundancy identified between the model and system-specific implementation details. Similar to IRONModel in some sense, our proposed technique provides a close-form equation to estimate IO performance and, when it is applied to any real system, the results will be steered by a few measured data points for the IO size of interest.

Machine learning approach: There is a body of work that essentially treats storage system as black box and applies statistical machine learning techniques based on performance data collected from the system. Inside-out [8] deals with the performance of distributed storage systems such as Ceph. One natural and popular approach often relies on a broad category of regression techniques [20]–[23]. Ganapathi [7] explores three classes of data- and compute-intensive applications in his PhD thesis, using more involved machine learning techniques such as KCCA (Kernel Canonical Correlation Analysis) and presents promising results with anecdotal success stories. In general, these techniques require a large volume of data to bootstrap and reach a desirable level of prediction performance.

VI. Conclusion and future work

In this paper, we proposed a novel technique to accurately estimate the performance of an arbitrarily mixed workload, in terms of read/write ratio and IO size. Our surprisingly simple technique requires only a few data points to provide its intended accuracy and can be applicable to any system as long as one can run a configurable workload a priori. Much work remains. In the future, we will collect more measurement data points using different storage system implementations, e.g., Cassandra, to further verify whether our technique is generally applicable. In addition, we plan to explore the applicability of our technique to a close-loop feedback controller for software-defined storage ecosystem in an enterprise environment.

References

[1] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "Crush: Controlled, scalable, decentralized placement of replicated data," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 2006, p. 122.

[2] "OpenStack Object Storage (Swift)," http://swift.openstack.org.

[3] "Apache Cassandra," http://cassandra.apache.org/.

[4] M.-R. Ra, "Understanding the performance of ceph block storage for hyper-converged cloud with all flash storage," arXiv preprint arXiv:1802.08102, Tech. Rep., 2018.

[5] E. Shriver, A. Merchant, and J. Wilkes, "An analytic behavior model for disk drives with readahead caches and request reordering," pp. 182–191.

[6] E. Thereska and G. R. Ganger, "Ironmodel: Robust performance models in the wild," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, pp. 253–264, 2008.

[7] A. Ganapathi, "Predicting and optimizing system utilization and performance via statistical machine learning," Ph.D. dissertation, University of California at Berkeley, 2009.

[8] C.-J. Hsu, R. K. Panta, M.-R. Ra, and V. W. Freeh, "Inside-out: Reliable performance prediction for distributed storage systems in the cloud," in *Reliable Distributed Systems (SRDS), 2016 IEEE 35th Symposium on*. IEEE, 2016, pp. 127–136.

[9] "Ceph," http://ceph.com/.

[10] "OpenStack: Open source software for creating private and public clouds," http://www.openstack.org/.

[11] "FIO: Flexible I/O Tester," https://github.com/axboe/fio.

[12] "COSBench - Cloud Object Storage Benchmark," https://github.com/intel-cloud/cosbench.

[13] H. W. Lee and M.-R. Ra, "Mist: Mitigating host-side interference for storage traffic in virtualized data centers," in *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*. IEEE, 2016, pp. 268–275.

[14] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population." in *FAST*, vol. 7, no. 1, 2007, pp. 17–23.

[15] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending ssd lifetimes with disk-based write caches." in *FAST*, vol. 10, 2010, pp. 101–114.

[16] K. A. Smith and M. I. Seltzer, "File system aging – increasing the relevance of file system benchmarks," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 25, no. 1. ACM, 1997, pp. 203–213.

[17] A. Conway, A. Bakshi, Y. Jiao, W. Jannen, Y. Zhan, J. Yuan, M. A. Bender, R. Johnson, B. C. Kuszmaul, D. E. Porter *et al.*, "File systems fated for senescence? nonsense, says science!" in *FAST*, 2017, pp. 45–58.

[18] T. Kelly, I. Cohen, M. Goldszmidt, and K. Keeton, "Inducing Models of Black-Box Storage Arrays Inducing Models of Black-Box Storage Arrays," HP Laboratories, Tech. Rep., 2004.

[19] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar, "Basil: Automated io load balancing across storage devices." in *FAST*, vol. 10, 2010, pp. 13–13.

[20] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. Ganger, "Storage device performance prediction with CART models," pp. 588–595.

[21] M. Wang, "Performance Modeling of Storage Devices using Machine Learning," Ph.D. dissertation, Carnegie Mellon University, 2006.

[22] Q. Noorshams, D. Bruhn, S. Kounev, and R. Reussner, "Predictive performance modeling of virtualized storage systems using optimized statistical regression techniques," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ACM, 2013, pp. 283–294.

[23] L. Y. L. Yin, S. Uttamchandani, and R. Katz, "An Empirical Exploration of Black-Box Performance Models for Storage Systems," pp. 433–440.