BlockFlex: Enabling Storage Harvesting with Software-Defined Flash in Modern Cloud Platforms

Benjamin Reidys* Jinghan Sun* Anirudh Badam† Shadi Noghabi† Jian Huang

University of Illinois at Urbana-Champaign †*Microsoft Research*

Abstract

Cloud platforms today make efficient use of storage resources by slicing them among multi-tenant applications on demand. However, our study discloses that cloud storage is still seriously underutilized for both allocated and unallocated storage. Although cloud providers have developed harvesting techniques to allow evictable virtual machines (VMs) to use unallocated resources, these techniques cannot be directly applied to storage resources, due to the lack of systematic support for the isolation of space, bandwidth, and data security in storage devices.

In this paper, we present BlockFlex, a learning-based storage harvesting framework, which can harvest available flash-based storage resources at a fine-grained granularity in modern cloud platforms. We rethink the abstractions of storage virtualization and enable transparent harvesting of both allocated and unallocated storage for evictable VMs. BlockFlex explores both heuristics and learning-based approaches to maximize the storage utilization, while ensuring the performance and security isolation between regular and evictable VMs at the storage device level. We develop BlockFlex with programmable solid-state drives (SSDs) and demonstrate its efficiency with various datacenter workloads.

1 Introduction

In modern cloud platforms, storage devices such as flash-based solid-state drives (SSDs) have been virtualized as system-wide shared resources to provide storage services across multiple application instances [6, 10, 15, 30, 39, 65]. This enables cloud platforms to make efficient use of storage capacity and bandwidth by slicing them among multiple multi-tenant virtual machines (VMs) [44, 61, 75]. However, our study of the event traces collected from popular cloud platforms [3, 10, 23] reveals that storage I/O is still significantly underutilized for both unallocated (unsold) and allocated storage. For instance, we find that 40% of the cloud storage servers have 25% of their storage unallocated, and the I/O utilization of allocated storage is under 33% on average (see Figure 1 and §2.1).

To improve the resource efficiency in the cloud, providers offer evictable VMs (i.e., Spot VMs or Harvest VMs) [4, 5, 24]. These evictable VMs allow users to use unallocated resources with low priority, i.e., the resources of evictable VMs can be reclaimed by regular VMs at any time. Recent studies [6, 49, 69, 76] advanced this technique by improving the resource allocation and scheduling for evictable VMs with heuristic-based harvesting approaches.

However, prior work on resource harvesting mainly focused on CPU and memory resources, which cannot be directly be applied to cloud storage for three reasons. First, current cloud storage virtualization approaches do not support storage harvesting, and dynamic reallocation of resources is not feasible. Second, cloud storage usually stores sensitive application data, which requires careful management for storage allocation and deallocation. Third, cloud storage can suffer from significant harvesting overhead due to the block erasure and metadata updates, which requires specific optimizations for enabling efficient storage harvesting.

In this paper, we present BlockFlex, the first learning-based storage harvesting framework, which enables transparent storage harvesting for both allocated and unallocated storage at a fine-grained granularity, while ensuring data privacy for cloud users with low harvesting overhead.

To develop BlockFlex, we first conduct a characterization study of storage resources that could be harvested in a cloud platform. According to our study (see §2), we find that for unallocated VMs configured with 512GB SSD, 78%, 43%, and 25% of them can be harvested and used for 1 hour, 6 hours, and 12 hours, respectively. This provides us the heuristic information about how these unallocated storage resources can be utilized. As for the allocated storage for VMs, an average of 70% can be harvested, however, the time available for harvesting varies depending on the workloads running in the VMs. Our study discloses the dynamics of available storage resources, which drives us to develop a learning-based approach for assisting the storage harvesting.

*Co-primary authors.

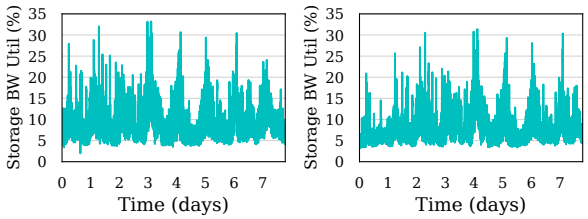
To enable transparent and fine-grained storage harvesting, we rethink the abstractions of storage virtualization for flash-based SSDs. The recent development of software-defined flash (SDF) in datacenters [30, 52] allows VMs to map their storage to dedicated flash channels. We build on top of the SDF abstraction and propose a new class of virtualized SSDs, named *ghost vSSD*. A ghost vSSD is created by harvesting free flash blocks from either unallocated or allocated but unused storage. The ghost vSSD provides the flexibility for fine-grained storage allocation and deallocation as well as block-level state tracking. It enables storage harvesting at the device level, which is transparent to the upper-level applications running on the VMs. Each ghost vSSD aims to meet the storage capacity and bandwidth requests from an evictable VM, however if needed, they can be reclaimed by regular VMs at any time.

However, frequent preemption and harvesting will inevitably introduce performance overheads to both regular VMs and evictable VMs, and even cause VM recreations. Therefore, it is desirable to provision the best-fit storage resource for an evictable VM. To achieve this, we develop learning-based techniques to predict the storage demands as well as the storage resources available for harvesting, in terms of storage capacity, bandwidth, and the duration available for harvesting. With these predictions, for each ghost vSSD, BlockFlex ensures the harvested storage resource will maximally meet the requirements of evictable VMs, while minimizing the opportunity of being preempted unexpectedly by regular VMs.

BlockFlex uses the Long Short-Term Memory (LSTM) network for online predictions at runtime, because of its low overhead and ability to make time-series predictions. We improve the prediction accuracy by developing different LSTM models for different dimensions of storage properties. For the predictions of storage capacity, bandwidth, and the time available for harvesting, BlockFlex can reach at 94.1%, 95.3%, and 93.1% accuracy, respectively, with slight over-provisioning. Upon mispredictions, BlockFlex implements different exception handlers for different cases (see the details in Table 1). As mispredictions do not happen frequently, the performance impact of misprediction handling is negligible in BlockFlex.

To minimize the performance interference between the regular VM and evictable VM caused by the storage harvesting, we assign higher priority to I/O requests from regular VMs when sharing the same SSDs with evictable VMs. When the harvested storage needs to be reclaimed, its flash blocks will be erased first to ensure data security, and then returned back to the corresponding regular VMs. Overall, we make the following contributions in this paper.

- We conduct a characterization study of the storage efficiency in different cloud platforms, our observations motivate the desirable need for storage harvesting.
- We rethink the abstractions of storage virtualization in modern cloud platforms for enabling fine-grained storage harvesting with software-defined flash.



(a) Storage utilization per VM. (b) Storage utilization per server.

Figure 1: The bandwidth utilization of allocated cloud storage.

- We build a learning-based storage harvesting framework named BlockFlex that can harvest both unallocated and allocated storage resources.
- We develop lightweight predictors that can make efficient predictions for both storage demand and availability in terms of storage capacity, bandwidth, and the time available for harvesting.
- We implement BlockFlex with real programmable SSDs and show its efficiency with various datacenter workloads.

Our experiments show that BlockFlex can improve the overall storage utilization by up to 1.75× in cloud platforms. BlockFlex is lightweight, it incurs trivial additional overheads to cloud platforms. BlockFlex can improve the performance of evictable VMs running with batch-processing workloads by 1.68× on average, while having negligible negative impact on the performance of regular VMs. The codebase of BlockFlex is available at https://github.com/platformxlab/blockflex.

2 Characterization for Storage Harvesting

Although storage virtualization has been widely deployed in cloud platforms, we observe that storage devices are still significantly underutilized, in terms of both storage bandwidth and capacity. In this section, we first quantify the cloud storage utilization, and then we conduct a hypothetical analysis of the opportunities for storage harvesting.

2.1 Cloud Storage Utilization

The storage underutilization in cloud platforms is due to both the poor utilization of allocated storage resources and the large portion of unallocated resources, as we discuss below.

Allocated storage resources. We conduct the storage utilization study based on the open-source cloud traces from Alibaba [3] and Google [23]. These traces track the usage of allocated storage resources across both VMs and physical servers. Alibaba cloud traces contain the VM utilization logs of 4K servers over 8 days, and Google cloud traces were collected from 12.5K servers over 29 days. As different cloud traces emphasize different aspects of the cloud storage usage (e.g., storage capacity, I/O bandwidth, server utilization, and

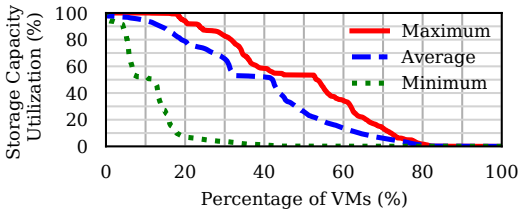


Figure 2: The capacity utilization of allocated cloud storage.

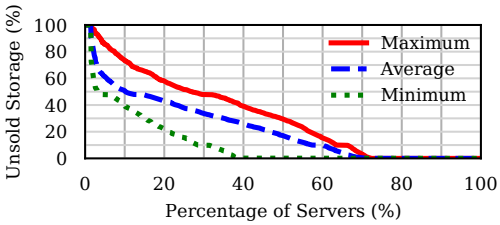


Figure 3: Unallocated storage in cloud servers.

VM utilization), we analyze both traces. We summarize our study results as follows:

- *Storage bandwidth:* We show the bandwidth utilization of Alibaba cloud [3] in Figure 1. The bandwidth utilization of allocated storage across all VMs is below 33%, and the average bandwidth utilization across all VMs over their entire lifetime is 9.2%. For physical servers that usually host multiple VMs, we obtain a similar trend: the bandwidth utilization of the physical storage devices is below 31%, and the average bandwidth utilization is 8.6%.
- *Storage capacity:* We present the cumulative distribution of storage capacity across the VMs of Google cloud [23] in Figure 2. We find that 20% of the VMs almost did not use their allocated storage capacity, 50% of the VMs used only 26.4% of the allocated storage capacity on average, and only 20% of the VMs used up to 90% of their allocated storage. Although different VMs may allocate different storage capacities, our study shows that their capacity utilization is surprisingly low.

The low utilization of allocated cloud storage resources is mainly due to two major reasons. First, cloud platforms usually allocate storage resource associated with each VM at a coarse-grained granularity for simplified storage management. For instance, the storage capacity of a VM in the Azure Cloud is linearly proportional to the number of allocated processor cores [6, 76], no matter whether the VM is I/O-intensive or CPU-intensive. Second, storage allocation is usually conducted in a static manner, while the storage usage of the workloads running in each VM changes dynamically over time. Therefore, the user of a VM has to over-provision sufficient storage for the peak demand upon VM creation.

Unallocated (unsold) storage resource. Beyond the allocated storage, the unallocated (unsold) storage in cloud platforms is another source for storage underutilization. This is because cloud providers usually over-provision VMs in their resource pool to satisfy the elasticity requirement from customers [6]. As each unsold VM consumes a fixed amount

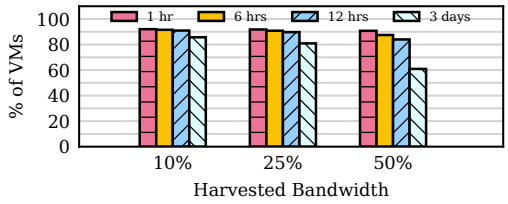


Figure 4: The availability of allocated storage for harvesting.

of resources (e.g., processor cores, memory, and storage), it will result in storage resources unallocated.

To further understand the unallocated storage, we analyze the cloud traces of unsold storage resources from Azure Cloud [6]. The traces include the VM allocation/deallocation logs for about 1,400 servers over 24 hours. As shown in Figure 3, nearly 70% of cloud servers have unsold storage resources, 50% of the servers have an average of 17.3% of their storage unallocated, and 20% of the servers have at least 20.1% of their storage unallocated. Given that a datacenter has thousands of servers, the unallocated storage is another critical source for the storage underutilization.

2.2 Opportunities for Storage Harvesting

As discussed in §2.1, we identify two sources for storage harvesting: unallocated storage and allocated storage. In this part, we conduct a hypothetical analysis of these storage resources to understand their potential for storage harvesting.

Analysis methodology. We study the cloud traces as discussed in §2.1, with a focus on the storage resource allocation and deallocation. We analyze the available storage in allocated and unallocated VMs over time, and check (1) whether we can harvest storage from them for a hypothetical harvest VM requesting a certain amount of storage capacity; (2) how long the harvested storage can last; (3) how many storage resources we can potentially harvest for the hypothetical harvest VMs. Note that Google and Alibaba cloud traces only report normalized numbers, so we use percentages rather than absolute numbers in our analysis.

Allocated storage resource. We first apply the hypothetical analysis on the allocated storage resource. Given a hypothetical harvest VM requesting different percentages (10%, 25%, and 50%) of storage bandwidth from a regular VM, we investigate how many servers have such available bandwidth, and how long these resources are available for harvesting. We report the average percentage across the entire trace. The results are summarized in Figure 4. We observe that more than 91% of the servers have harvestable bandwidth for 12 hours, and about 76% of the servers have harvestable bandwidth for 3 days. As we harvest storage for a shorter time (i.e., less than 12 hours), the portion of the available servers is consistently high. This is due to the constant low storage utilization of allocated VMs, as shown in Figure 1.

Unallocated storage resource. We now explore the unallocated storage resource. Given a hypothetical harvest VM that

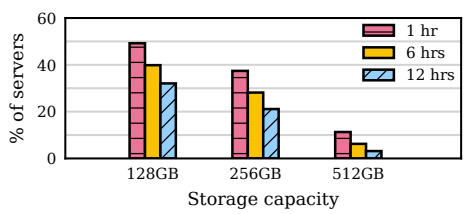


Figure 5: The availability of unallocated storage for harvesting.

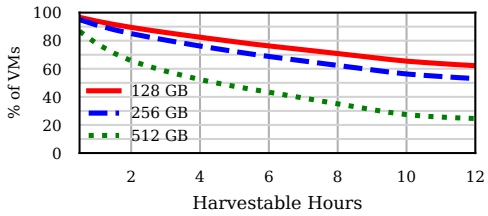


Figure 6: The availability of unsold regular VMs for storage harvesting with different capacities.

requests different storage capacities (128GB, 256GB, and 512GB), we analyze how many servers can satisfy the request from this harvest VM, and how long the available storage can last. We present the study results in Figure 5. Our study finds that 32% of the servers can satisfy the requirement of 128GB storage capacity for 12 hours. If the harvest VM requests storage for a shorter time, such as 1 hour, 50% of the servers can meet the request. As harvest VM increases the requested storage capacity, the number of harvestable servers decreases.

We also study unsold regular VMs. We vary the storage capacity request from 128GB to 512GB for the hypothetical harvest VM, and demonstrate our study results in Figure 6. For a hypothetical harvest VM of 128GB storage capacity, 94%, 76%, and 62% of the unsold regular VMs can be harvested for 1 hours, 6 hours, and 12 hours, respectively. As we increase the requested storage capacity for the harvest VM, the percentage of available unsold regular VMs drops. However, we still find a decent amount of unsold regular VMs can be harvested. For instance, for the harvest VM that requests 512GB storage capacity, 43% and 24% of the unsold VMs are available for 6 hours and 12 hours, respectively.

It is worth noting that the storage bandwidth is usually allocated proportionally with storage capacity in cloud platforms [20, 30]. This is also reflected in the cloud traces we studied in this paper. For instance, as for the VM with 128GB, 256GB, and 512GB, the storage bandwidth is 192 MB/s, 384 MB/s, and 768 MB/s, respectively. Thus, our study on the unsold storage capacity also applies to the storage bandwidth.

Takeaways. Our characterization study shows that:

- Both unallocated and allocated storage have sufficient storage capacity and bandwidth for harvesting, and they are available long enough to facilitate harvesting.
- The harvestable storage resource varies depending on the storage capacity and time available for harvesting. Harvesting a large storage capacity for a longer time has a lower chance of identifying the available storage resource.
- The harvestable storage resource from unallocated VMs and allocated VMs shows different availability patterns and trade-offs. We have a larger chance to harvest storage in allocated VMs, but this may have interference with the regular VM. The harvestable storage from unallocated VMs is limited, but it has no impact on the performance of regular VMs.

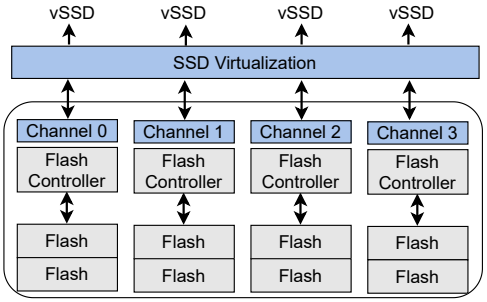


Figure 7: Storage virtualization with software-defined flash.

With BlockFlex, we aim to improve the cloud storage utilization by harvesting the available storage resources from both allocated and unallocated storage.

3 Technical Background

To facilitate our discussion, we first present the essential technical background of storage virtualization in cloud platforms, and then discuss the harvest VMs that will benefit from storage harvesting.

3.1 Storage Virtualization and SDF

In modern cloud platforms, storage virtualization has become the backbone of the storage infrastructures, in which storage devices such as flash-based solid-state drives (SSDs) are virtualized and shared by multiple VMs in order to improve storage utilization [30, 39, 61, 65]. The storage virtualization layer provides the system abstraction of virtualized storage devices (e.g., virtual disks) and hides the underlying hardware complexities from upper-level VMs. Each VM can have one or more virtual storage devices, and each virtual storage device can be mapped to one or more physical storage devices.

At the same time, SSDs are increasingly being adopted by cloud providers for their low latency and high throughput [1, 30, 31, 47]. Internally, an SSD consists of multiple flash channels, each channel has multiple flash chips, and each chip has thousands of flash blocks (see Figure 7). Each channel can issue I/O requests independently, thus, offering high parallelism and performance isolation. SSDs can only write data to free blocks, and once a free block is written, it is no longer available for future writes until it is erased. However, the erase operation is time-consuming. Thus, writes are issued to

flash blocks that have been erased in advance (i.e., out-of-place update). Because of this, SSDs employ a flash translation layer (FTL) to maintain the logical-to-physical address mapping, and manage the garbage collection (GC) operations.

To ultimately exploit the performance benefits of SSDs in the cloud, software-defined flash (SDF) was developed [40, 52]. In the context of SSD virtualization, SDF allows the upper-level VM to map its virtual SSD (vSSD) to a set of flash channels, as shown in Figure 7. Therefore, cloud providers can allocate storage capacity and bandwidth to each vSSD per its request by allocating fewer/more flash channels, following the pay-as-you-go model, while enabling the device-level performance isolation between vSSDs. The vSSD performs like a conventional storage disk, it provides the block interface to upper-level software, and uses a mapping table to index the logical-to-physical block address mappings [30, 52]. As SDF enables various cloud services such as Database-as-a-Service (DaaS) and Infrastructure-as-a-Service (IaaS) to achieve predictable storage performance and satisfy their service level objectives (SLOs), it has become an essential component in modern cloud platforms [17, 32, 56, 57, 64]. In this work, we develop BlockFlex based on the software-defined flash infrastructure.

3.2 Harvest Virtual Machine

To improve the resource utilization in cloud platforms, a few VM techniques have been developed recently [4–6, 13, 21, 59, 69]. Cloud providers offer *evictable VMs* or *Spot VMs* that run with lower priority than regular VMs, they can be evicted if resources are needed by a regular VM [4, 5]. With evictable VMs, cloud providers can sell unsold resources at a lower price while providing resource guarantees for regular VMs. Therefore, cloud customers usually rent evictable VMs to run batch-processing workloads or similar applications that have lower requirements on resource guarantees. Based on the evictable VMs, researchers developed *harvest VM* [6], *elastic VM* [69], and *memory-harvesting VM* [21], which further improve the cloud resource utilization by enabling flexible and dynamic harvesting of unallocated resources. To simplify the discussion, we will use harvest VM to represent these aforementioned VMs for resource harvesting in the remainder of the paper.

A majority of these harvest VMs were developed to harvest CPU and memory resources, and none of them can be directly applied to the storage resources. Additionally, prior work proposed various VM scheduling techniques by co-locating multi-tenant applications on the shared bare-metal servers to improve the resource efficiency [42, 44, 66, 71]. However, our study of various cloud traces discloses that the storage utilization is still a severe issue within modern cloud platforms. Since storage virtualization today assumes exclusive ownership of storage resources for each VM, it inevitably causes storage underutilization. In this work, we enable the storage harvesting for harvest VMs to improve the cloud storage utilization.

4 Design and Implementation

In this section, we first discuss the design goals and challenges of BlockFlex. After that, we will present the overview of the system as well as the design and implementation details of each component.

4.1 Design Goals and Challenges

As we develop BlockFlex to enable efficient storage harvesting, we aim to achieve the following goals:

- The storage harvesting should satisfy the storage requirements from harvest VMs while minimizing unexpected preemptions by regular VMs.
- The storage harvesting should be transparent to the upper-level VM to minimize changes to the VM and applications, as well as facilitate its production deployment.
- The storage harvesting should have minimal negative impact on the regular VMs to guarantee the quality of cloud services as we improve the global storage utilization.
- The storage harvesting should ensure the data safety, when it temporarily allocates unused data blocks from both allocated and unallocated storage to the harvest VMs.

Since cloud platforms today do not provide system support for storage harvesting, it is not easy to achieve the above goals. Additionally, existing resource harvesting techniques cannot be directly applied to storage resources. Specifically, we have to overcome the following challenges. First, cloud customers usually rely on the storage to permanently store their data, the data durability and availability are critical for storage services. This makes the storage harvesting fundamentally more challenging than the harvesting of CPU and memory resources. For example, shrinking available storage (upon reclamation) may result in data loss, while reclaiming memory and CPU resources mainly causes reduced performance. Second, the storage virtualization and management are different from that of CPU and memory resources, especially for SSDs that have intrinsic properties (see §3.1). Therefore, sharing storage resources while maintaining isolation among tenants needs new techniques. Third, storage allocation and deallocation usually incur more performance overhead than the context switch overhead caused by harvesting CPU and memory resources, which requires special development efforts for enabling the deployment of storage harvesting in cloud platforms.

4.2 System Overview

To the best of our knowledge, BlockFlex is the first storage harvesting framework built based on modern software-defined storage infrastructure. We present the system architecture of BlockFlex in Figure 8. To manage the harvested storage, we propose a new abstraction, named ghost vSSD (gSSD), on top of software-defined flash (§4.3). The ghost vSSDs

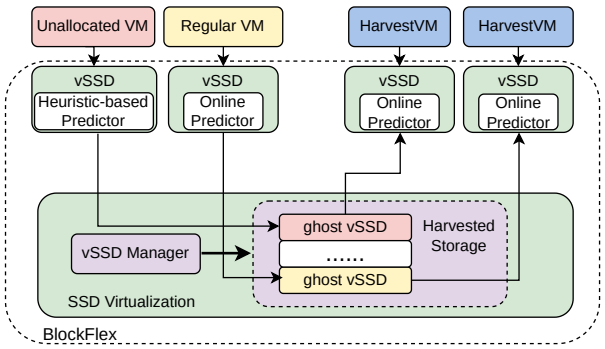
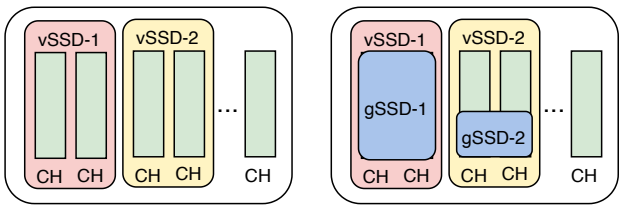


Figure 8: System overview of BlockFlex.



(a) Allocated and Unallocated vSSDs (b) Harvest available storage

Figure 9: Examples of harvesting storage. CH: flash channel; vSSD-1: unsold storage; vSSD-2: allocated storage; gSSD-1: harvest unsold storage; gSSD-2: harvest allocated storage.

can be attached to created vSSDs , therefore, no changes are required to VMs. BlockFlex will deploy a predictor in each vSSD (§4.4). For harvest VMs, BlockFlex will predict their demanded storage capacity and bandwidth, as well as how long the demand will last. For regular VMs, BlockFlex will predict their available storage capacity and bandwidth, as well as their available time. For unused storage resources, BlockFlex will use both heuristic-based approaches to predict the duration time available for harvesting. Based on the prediction, BlockFlex will make a best-fit match and allocate unused storage to the harvest VM. In case resource preemption happens to the harvest VM (caused by misprediction), BlockFlex will release the harvested storage to the regular VM and handle the exceptions for different scenarios (§4.5).

Since BlockFlex enables storage harvesting at the system virtualization level, it does not change the upper-level durability model (e.g., data replication) offered by current cloud storage infrastructures. For harvest VMs, cloud platforms assume their end users are aware of the relaxed durability guarantees and their applications may suffer from early reclamations. BlockFlex makes the best effort to allocate new gSSDs to ensure the data durability for harvest VMs. However, similar to Spot VMs [63], the owners of harvest VMs should be aware of the risk and take responsibility for their data as the cost of harvest VMs is much lower than regular VMs. As BlockFlex is deployed on top of existing software-defined storage infrastructure, it runs in a distributed environment where the global control plane manages the gSSDs and their allocations/deallocations. In the following section, we will discuss each technique proposed in BlockFlex, respectively.

4.3 New Abstraction for Storage Harvesting

As discussed in §3.1, with software-defined flash, the storage virtualization can map each virtual SSD to a number of flash channels depending on the storage capacity and bandwidth requested by the associated VM. We show two typical examples in Figure 9. Suppose we have a 1TB SSD that contains 16 channels. Each channel has 64GB and delivers a bandwidth of 70MB/s. As shown in Figure 9 (a), the cloud platform allocates two flash channels to vSSD-2 (128GB), leaving other flash channels temporarily unused (e.g., vSSD-1). Both vSSD-1 and vSSD-2 could provide opportunities for storage harvesting. For example, as shown in Figure 9(b), the entire unsold vSSD-1 (gSSD-1) and part of the allocated vSSD-2 (gSSD-2) could be harvested depending on their availability. The SDF offers the flexibility to allocate fewer/more resources to each gSSD.

However, the harvested storage still belongs to the original vSSDs, which could be preempted by existing or newly allocated regular VMs. Since the availability of harvested storage varies depending on the workloads in the cloud platform, it increases the complexity of storage harvesting.

4.3.1 Definition of Ghost vSSD

To simplify the management of harvested storage, we develop the gSSD abstraction. Its block interface is the same as that of the regular vSSD. Therefore, no code modifications are required for the VMs. Similar to vSSDs, each gSSD has a block-level mapping table to index the mappings of logical block addresses to physical block addresses , and a free block list to manage the free flash blocks. However, since each gSSD is created/borrowed from regular vSSDs and has a different lifetime (the time available for harvesting), we maintain a metadata structure for each gSSD, as shown in Figure 10.

The metadata of a gSSD includes its maximum bandwidth and capacity. We use the number of flash channels to represent the storage bandwidth, and the number of flash blocks to represent the storage capacity. As the actual storage bandwidth and capacity offered by a gSSD could vary at runtime, we use their maximum values because they are provided on a best-effort basis. We use the *expire* to indicate when the gSSD will no longer be available for use. This value is predicted with our duration predictor (see the detailed discussion in §4.4). The metadata structure also has a bit *in_use* to indicate whether the gSSD has been assigned to a harvest VM or not. If yes, the *vm_id* stores the ID of the corresponding harvest VM. The *home* pointer points to the regular vSSD from which the blocks in the gSSD are harvested. The *ghost* points to the created ghost vSSD after storage harvesting. The metadata is stored in the gSSD. It is initialized when the gSSD is created and updated when the gSSD is harvested/reclaimed.

```
typedef struct vmeta {
    int bandwidth          ; maximum bandwidth of gSSD
    int capacity           ; maximum capacity of gSSD
    int expire             ; how long the gSSD lasts
    boolean in_use         ; used by harvest VM or not
    string vm_id           ; harvest VM ID
    struct vssd* home      ; vSSD that owns these blocks
    struct vssd* ghost     ; points to the attached gSSD
} vmeta_t;
```

Figure 10: Metadata of a ghost vSSD in BlockFlex.

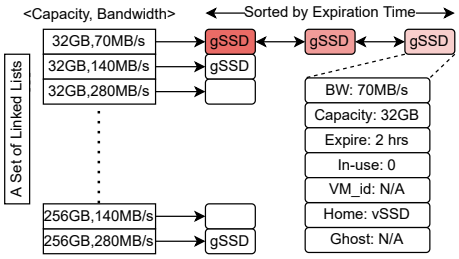


Figure 11: The organization of the gSSD pool in BlockFlex.

4.3.2 Management of Ghost vSSDs

We now discuss the gSSD creation and management.

Creating gSSDs. Instead of harvesting storage upon requests, BlockFlex allows regular vSSDs to proactively create gSSDs and add them into the gSSD pool managed by the vSSD manager (see Figure 8). This removes the harvesting procedure from the critical path. A vSSD creates a gSSD when its predictor predicts that it will have available storage resources for harvesting. These predictions occur at regular intervals (every three minutes by default). In order to create a new gSSD, BlockFlex will harvest free blocks from the vSSD and create a mapping table for them. Following our prior study on SDF [30], we use block-level address mapping tables for indexing flash blocks in the gSSDs/vSSDs. We align the address mapping granularity and flash erase granularity to simplify the storage management with improved efficiency. And each gSSD/vSSD has its own mapping table. Although the flash blocks of a gSSD could be harvested from a vSSD, the corresponding gSSD and vSSD will not share these harvested flash blocks. Therefore, we do not need to synchronize the mapping table entries between the gSSD and vSSD at runtime.

The metadata of a gSSD (Figure 10) is initialized with the number of flash channels harvested (*bandwidth*), the number of free blocks (*capacity*), and the predicted time the resources will be available for use (*expire*). The *home* of the gSSD will point to the regular vSSD, and the *ghost* will point to the newly created gSSD. At the same time, the gSSD will be added to the gSSD pool for serving future harvesting requests.

To simplify the management of gSSDs, we only create a gSSD when harvesting a chunk of resources. BlockFlex enables the storage harvesting at the granularity of a flash channel, 16GB size, and 30-minute for storage bandwidth, capacity, and duration time, respectively. To ensure reasonable performance isolation between regular VMs and harvest VMs, we restrict each vSSD to provide only one gSSD.

Managing gSSDs. To facilitate fast gSSD lookup, we organize gSSDs in a set of lists in the vSSD manager with considering the sorting in three dimensions: storage bandwidth, capacity, and time available for harvesting. We optimize the lists based on our observations that (1) the storage bandwidth and capacity are correlated with the number of channels available in a vSSD; (2) the time available for harvesting for each gSSD needs to be updated at regular intervals; and (3) we will not update the maximum storage capacity and bandwidth over the lifetime of a gSSD. Therefore, as shown in Figure 11, BlockFlex maintains a set of gSSD lists sorted by <capacity, bandwidth>. In each list, the gSSDs are sorted by their expiration time from the farthest one to the nearest one. There is a timer running periodically (per 15 minutes by default) to update the expire time in the gSSD pool. For the expired gSSDs but have not been allocated to any harvest VM, BlockFlex will remove them from the list.

Harvesting gSSDs. Upon receiving a request for storage harvesting, BlockFlex will check the gSSD pool to identify a best-fit match for the requested storage capacity, bandwidth, and time available for harvesting. BlockFlex uses the best-fit matching policy to minimize the waste of storage resources. These requested parameters are obtained from the predictors deployed in the vSSD of the corresponding harvest VM (see §4.4). Since the gSSD pool is sorted, we use the binary search to first locate the corresponding list that matches with the requested storage capacity and bandwidth. After that, we walk through the list until identifying an available gSSD whose expire time matches with the requested harvestable time.

Once a gSSD is identified in the pool, we set its *in_use* to 1 to indicate this gSSD has been assigned to a harvest VM and the corresponding harvest VM ID is recorded. BlockFlex supports concurrent gSSD allocations by managing the gSSD lists using non-blocking linked-lists implementation with the compare-and-swap operations [28]. Compared to the lifetime of a gSSD (hours or even days), the gSSD allocation overhead (a few microseconds) is trivial.

With a harvested gSSD, BlockFlex will assign its flash blocks to the vSSD of the corresponding harvest VM. This harvesting procedure is transparent to the harvest VM, as we track these blocks in the mapping table of the vSSD of the harvest VM, as shown in Figure 12. The address mapping table in the vSSD is extended to include the ID of the harvested gSSD. Therefore, upon data accesses from the harvest VM, its vSSD will conduct the address translation to translate the logical block address (LBA) to [gSSD-ID, gLBA]. With the obtained gSSD-ID, the corresponding gSSD will translate the gLBA to the physical block address (PBA). This enables BlockFlex to harvest multiple gSSDs for a harvest VM. With the expanded vSSD, the harvest VM can resize the vSSD and its file system with existing virtual disk and file system tools [14, 19, 67]. Note that the address mapping of a vSSD will also index the

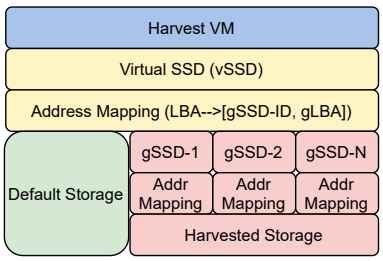


Figure 12: Harvesting multiple ghost vSSDs for a harvest VM.

default storage allocated when a harvest VM is created.

We assume each harvest VM will not request more than 256 gSSDs, so 1 byte is used to index the gSSDs. In total, each address mapping entry takes 9 bytes (4 bytes for LBA and 4 bytes for PBA). Given a harvest VM that requests 128GB storage, and each flash block is 4MB, the block-level address mapping of a vSSD will take only 288KB.

Reclaiming gSSDs. When a harvest VM finishes its jobs, the harvested gSSDs will be reclaimed to the pool in the vSSD manager. Upon the gSSD reclamation, the corresponding entries in the address mapping table of the vSSD will be removed. BlockFlex will check whether a gSSD will expire soon or not (i.e., in 30 minutes by default). If yes, BlockFlex will erase the flash blocks for data safety, and remove the gSSD instance. Otherwise, BlockFlex will add the gSSD into the gSSD pool for future harvesting. Since the erase operation is expensive, BlockFlex leverages the channel parallelism of an SSD to execute them in parallel.

The additional erase operations caused by gSSD reclamation has minimal impact on the lifetime of SSDs. This is for two major reasons. First, BlockFlex ensures wear leveling of SSDs by following a relaxed wear-leveling scheme proposed in our prior study [30]. It showed that SDF can achieve near-ideal SSD lifetime by swapping channels every 19 days on average for data center workloads, and 12 days on average for the worst case of erasing channels at full bandwidth. The wear leveling plays a fundamental role of ensuring the device lifetime, no matter whether flash blocks are used by regular VMs or harvest VMs. Second, the harvesting procedure itself only introduces erases when harvested storage is reclaimed, and it happens infrequently. Based on our study, for a given vSSD, it is harvested about every 2.1 days and consumes an average of 25% of the SSD (see §5.2), meaning the entire vSSD is erased once per 8.4 days. For modern SSDs that usually have 10K P/E cycles and can last 5-year lifetime, the storage harvesting operations will consume about 2% of the device lifetime, which is acceptable in practice.

In addition, with the assistance of predictors (see §4.4), BlockFlex minimizes the chances of early reclamation, and takes the erase operations from the critical path. However, a reclamation would still happen, even though a gSSD is in use by a harvest VM. This could be caused by the resource preemption issued by a regular VM. We will discuss how BlockFlex handles this in details in §4.5.

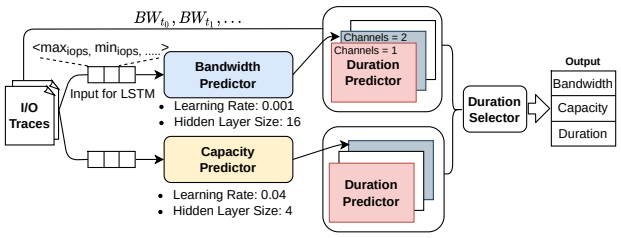


Figure 13: The workflow of the predictors used in BlockFlex.

4.4 Predictions for Storage Harvesting

Instead of relying on the cloud customers or VM users to specify their demanded or unused storage resource, we use a lightweight online learning approach to predict them. As discussed in §4.2, each vSSD has an online predictor, except those for the unallocated (unsold) VMs.

4.4.1 Heuristic-based Prediction for Unsold VMs

For the unallocated (unsold) VMs, we use a heuristic-based approach, based on our study characterizing the unallocated storage in cloud platforms (see §2). Recall that cloud providers usually over-provision VMs to provide the elasticity for their services. They reserve different regular VMs with various storage capacities. The common sizes include 128GB, 256GB, and 512GB for simplified VM management and deployment. According to our study in Figure 6, their availability for harvesting varies by their capacities.

Previous harvesting studies have identified that past values are a useful indicator for the available time of unsold storage [6]. In our study of unsold storage resource, we confirm that the available time of unsold storage for harvesting is stable. For this reason, we tag each unsold VM with a predicted duration time using the histogram of previous available times for the unsold VM with the same storage capacity. For instance, for the unsold VMs with 512GB storage capacity, we can use 20% of them as gSSDs that would be available for 12 hours, 20% for 6 hours, and the remaining for 1 hour. This distribution could change depending on the heuristic study of the corresponding cloud platform. The distribution of these gSSD sizes depends on the configured storage capacities for the unsold VMs.

4.4.2 Online Learning for Allocated and Harvest VMs

We predict the harvestable storage resource for allocated VMs, and demanded storage resource for harvest VMs. Since the predictions for allocated VMs and harvest VMs are both determined by their workloads, they use the same learning-based approach but different learning parameters.

We show the entire prediction workflow of BlockFlex in Figure 13. In each vSSD, we collect the read, write, and erase operations at the block layer for online predictions, therefore, we do not rely on the systems software running on top of

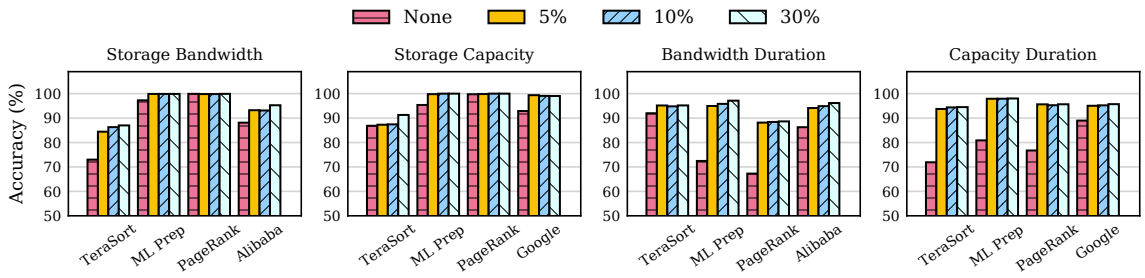


Figure 14: Prediction accuracy of storage bandwidth, capacity, and duration time available for harvesting, with various over-provisioning ratios. A slight over-provisioning for storage harvesting can significantly improve the prediction accuracy.

the vSSD. Based on these I/O traces, we infer the bandwidth, throughput (IOPS), and current storage utilization.

We use Long-Short Term Memory (LSTM) models [29] to develop our predictors, because of their strength in time-series predictions and relatively low overhead. The inputs for LSTMs are statistical measures gathered from the bandwidth, IOPS (e.g., max_{iops}, min_{iops}), and storage utilization. By default, BlockFlex trains the models every three minutes using the collected statistics from the preceding 15 minutes. This introduces minimal performance and memory overhead. Both bandwidth predictor and capacity predictor use the same LSTM model, but we tune their learning rate and hidden layer size slightly differently for improved accuracy (see Figure 13). These predictors will generate the predicted bandwidth (in channels) and predicted capacity (in GB), respectively.

The predictions of storage bandwidth and capacity are passed to their respective duration predictors. Each duration predictor consists of a collection of individual sub-predictors. Each sub-predictor is responsible for a possible output from the bandwidth/capacity predictors. For instance, if the output of the bandwidth predictor ranges from 1 to 16 channels, we will have 16 duration sub-predictors, each sub-predictor will predict its corresponding duration time by using the history of previous durations at that demand.

To ensure a gSSD can satisfy both the storage and bandwidth requirements from a harvest VM, the duration selector takes the maximum duration for demanded storage resources. To ensure a regular VM will not reclaim resources early, the selector takes the minimum duration for the harvestable storage resources. The final output delivered by the predictors in BlockFlex is presented in a tuple of <bandwidth, capacity, duration>. We describe the details of each predictor as follows.

Storage bandwidth: For the prediction of storage bandwidth, we use six inputs for the LSTM model: the *maximum, minimum, and average* for both bandwidth and IOPS. We do not use other statistical measures as inputs because they do not improve the prediction accuracy and slow down the convergence of the model. As the number of flash channels is proportional to the storage bandwidth, we use the number of channels as the bandwidth metric to simplify the bandwidth prediction.

Storage capacity: The prediction model for the storage capacity is similar to the model used for the storage bandwidth. We use the *maximum, minimum, and average* of past storage utilizations, and the *current changes in storage utilization* as the inputs. We find that using the changes in storage utilization helps differentiate long periods of sequential writes against shorter changes. We use the number of flash blocks as the output of the capacity predictor.

Duration: For the duration, we make the predictions for storage bandwidth and capacity separately. For allocated VMs, we predict how long their available storage capacity and bandwidth can be used by harvest VMs; for harvest VMs, we predict how long a demand of storage capacity and bandwidth will last before more resources are needed. As discussed, a set of sub-predictors are used for each demanded bandwidth/capacity. BlockFlex updates and maintains the history of durations for model training and inference.

4.4.3 Resource Provisioning for Improved Accuracy

We examine the accuracy of the LSTM models we develop for the aforementioned predictors using various cloud workloads (see their descriptions in Table 2). A prediction for storage bandwidth and capacity is considered accurate if the predictor predicts at least as much as the actually demanded/available storage. A prediction of duration time is considered accurate if the predicted storage bandwidth and capacity lasts as long as the actual demand/availability. We track the actual storage demand/availability and predicted storage demand/availability to calculate the prediction accuracies.

As shown in Figure 14, the average accuracies of predicting storage bandwidth, capacity, and their durations are 89%, 93%, 79%, and 79% on average. Their accuracy varies for different workloads. To further improve the prediction accuracy and avoid resource preemptions (see §4.5), we use a simple yet effective approach – over-provisioning more storage resources based on the predictions of demanded storage resources, and under-provisioning storage resources based on the predictions of harvestable storage resources. We vary the provisioning ratio from 5% to 30%, and show the updated accuracies in Figure 14. We find the accuracies of all the predictions can reach 93–96% with a provisioning ratio of 5%. As we increase the provisioning ratio, we do not see much accuracy improvement. Therefore, we use the 5% provisioning ratio in BlockFlex by default.

Table 1: Exception handling for different scenarios.

ID	Harvestable Storage	Demanded Storage	Possible Exceptions
❶	Over-predict	Over-predict	Waste or Early Reclamation or N/A
❷	Over-predict	Under-predict	Under-Harvest or Early Reclamation
❸	Under-predict	Over-predict	Waste
❹	Under-predict	Under-predict	Under-Harvest or Waste or N/A

4.5 Exception Handling in Storage Harvesting

Although the predictors in BlockFlex deliver high accuracy as discussed in §4.4, mispredictions can still happen, causing exceptions during storage harvesting. Typical exceptions include the resource preemption in which a regular VM prematurely reclaims the harvested storage from a harvest VM, and under-harvesting in which a harvest VM must request additional storage resources to satisfy the request of more storage resource than the predicted demand. VM terminations and data loss could happen if these exceptions are not handled properly.

Misprediction types. Mispredictions can be categorized into two types: over-prediction and under-prediction. As we make predictions for both harvestable storage (in the regular VMs) and demanded storage (in the harvest VMs), the two misprediction categories apply to both sides, as shown in Table 1.

An over-prediction of demanded storage means that a harvest VM harvests more storage resources than it really needs; an under-prediction of demanded storage means that a harvest VM harvests less storage resources than it really needs. In contrast, an over-prediction of harvestable storage means that a regular VM has less harvestable storage resources than predicted; an under-prediction of harvestable storage means that a regular VM has more harvestable storage resources than predicted. During storage harvesting, any misprediction or combinations of mispredictions could cause an exception. BlockFlex employs different exception handling for each scenario.

Exception handling. As shown in Table 1[1], mispredictions could mainly cause three exceptions: *waste of storage resources*, *early resource reclamation*, and *under-harvesting*.

Waste of storage resources. BlockFlex could waste storage resources when mispredictions leave them unused. In the case ❶ of Table 1, a regular VM provides the storage resource requested from the harvest VM, although the harvest VM may over-predict its demanded storage resource. In case ❸, the waste of storage resources becomes worse, because the regular VM actually has more harvestable storage resources than the requested resources from the harvest VM. As we trade the over-provisioning of demanded storage in the harvest VMs for increased prediction accuracy, it is inevitable to cause some waste of storage resources. However, since BlockFlex uses a 5% over-provisioning ratio (see §4.4) in its predictors, the waste is minimal. Compared to the cloud platforms without storage harvesting, BlockFlex still improves the storage utilization. Therefore, BlockFlex does perform special exception handling for this exception.

Early resource reclamation. This could happen when a regular VM has less harvestable storage resources than the demanded storage resources from a harvest VM. Typical examples include the case ❶ and ❷ in Table 1, in which we over-predict the harvestable storage resource in a regular VM, but in reality, the regular VM has less harvestable storage than the demanded storage from a harvest VM. In both cases, the regular VM has to reclaim its storage from the harvest VM. To handle this exception, BlockFlex will identify a new gSSD that meets the requirements for storage capacity, bandwidth, and duration. After that, BlockFlex will copy all the data from the old gSSD to the new gSSD and update the address mapping table in the vSSD of the corresponding harvest VM. BlockFlex will migrate data between gSSDs at block granularity to minimize the impact on the running applications. BlockFlex will reclaim the old gSSD while ensuring its flash blocks are erased before being used by the regular vSSD (see §4.3.2). However, if there is no satisfactory gSSD available, an exception will be reported to the end users of the harvest VM (like what is done today for spot VMs [63]).

Under-harvesting. The exception of under-harvesting could happen when a harvest VM under-predicts its demanded storage resources (i.e., it requests less storage resources than it really needs). Typical examples include the cases ❷ and ❹. For the case ❷, under-harvesting could happen when the harvest VM under-predicts its demanded storage resources. For the case ❹, although the regular VM under-predicts its harvestable storage resources, the demanded storage in reality could still be more than the available storage resources in the regular VM. To handle this exception, BlockFlex will harvest new gSSDs for the harvest VM until meeting the demand. As discussed in Figure 12, BlockFlex enables the use of multiple gSSDs in a single vSSD. However, if there is no gSSD available, BlockFlex will report an exception to the users of the harvest VM, resulting in a termination of the harvest VM or a delay of job executions in the harvest VM.

Note that mispredictions could happen along all three dimensions (i.e., storage capacity, bandwidth, and time available for harvesting) of the storage resource. The described exception handling is used in BlockFlex for mispredictions along any of the three dimensions.

4.6 Implementation Details

We implement the gSSD abstraction of BlockFlex using a programmable SSD with 1TB capacity. The SSD has 16 channels, each channel has 4 dies, each die has 4 planes, each plane has 1024 blocks. Each block consists 256 pages, each 16KB. Its controller allows read/write/erase operations against the raw flash resources and enables the host to develop their own FTL for address translation, GC, and wear leveling.

The gSSD implementation takes 4.1K lines of code (LoC) using C programming language. The vSSD used in this paper

[1] If the demanded storage resource from a harvest VM exactly matches with the harvestable storage resource in a regular VM, there is no exception (N/A).

Table 2: Workloads used in our evaluation.

Workload	Description
TeraSort [26]	Sort data generated by TeraGen.
ML Prep [2]	Preprocess images for machine learning tasks.
PageRank [25]	Compute the pagerank of a graph.
YCSB [73]	Transaction processing on a database.

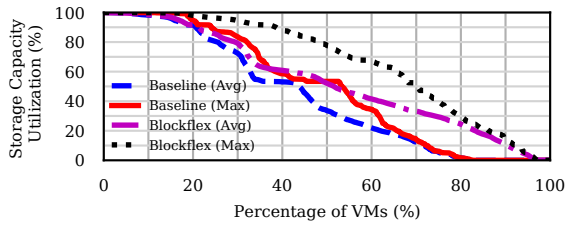


Figure 15: Improved utilization for underutilized storage.

is similar to the virtualized SSDs in our prior work [30]. BlockFlex creates different vSSDs for harvest VMs and regular VMs. It allocates physical flash channels for each vSSD to ensure performance isolation. Upon workload execution, BlockFlex handles the logical block I/O requests received by the vSSDs with actual read and write operations to the allocated physical flash blocks. We run BlockFlex on a real server with 8 Intel(R) Xeon(R) CPU E3-1240 v5 cores running at 3.5 GHz.

BlockFlex's predictors are implemented using PyTorch v1.9.0 [53] in 2.8K LoC using Python. Each model is implemented with one hidden LSTM layer fully connected with the input and output layers. The bandwidth and space predictors have an additional softmax layer applied to the output. All models use *adam* [37] as an optimizer and mean squared error as a loss function. We vary the learning rate and sizes of the hidden layer. Bandwidth prediction uses a learning rate of 0.005 and 16 hidden nodes. Capacity prediction uses a learning rate of 0.04 and 4 hidden nodes. Bandwidth duration uses a learning rate of 0.006 and 50 hidden nodes. Capacity duration uses a learning rate of 0.001 and 50 hidden nodes.

5 Evaluation

Our evaluation demonstrates that: (1) BlockFlex improves the storage utilization in cloud platforms by leveraging both underutilized and unallocated storage resources (§5.2); (2) BlockFlex improves the performance of harvest VMs while minimizing the impact on regular VMs (§5.3 and §5.4); (3) BlockFlex introduces negligible overhead to storage management (§5.5);

5.1 Experimental Setup

We evaluate BlockFlex with a set of synthetic workloads and real-world applications as shown in Table 2. We use Hadoop's TeraSort [26], ML Prep [2], and the PageRank implementation in GraphChi [25] to represent common applications in harvest VMs, while YCSB [73] represents common regular VM workloads. For TeraSort, we generate and sort 75GB datasets with

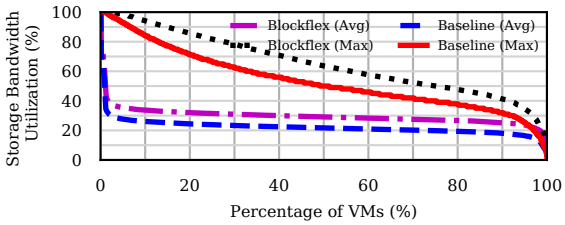


Figure 16: Improved utilization for underutilized bandwidth.

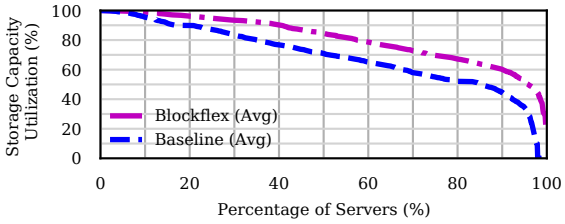


Figure 17: Improved utilization for unallocated resources.

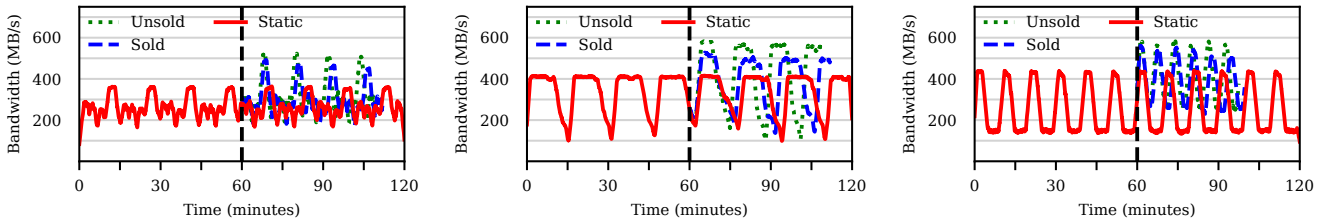
the TeraGen in Hadoop [26]. For PageRank, we use the Friendster graph (61GB) [72]. For ML Prep, we process images from the ImageNet data set (220 GB) [18]. For YCSB, we populate a key-value store RocksDB [55] with 180GB of data and run workloads A-E. In the evaluation, we report the numbers for YCSB-A since the workloads B-E deliver similar results.

5.2 Improved Storage Utilization

To evaluate the improved utilization of BlockFlex, we gather requests from 60,000 low priority VMs from Google traces to characterize the demand of harvest VMs. Their storage requests vary between 32GB and 512GB, and last between 30 minutes and 8.5 days (2.1 days on average). The demanded bandwidth is proportional to the demanded storage. We match these storage demands with harvestable storage capacity from 4,000 regular VMs. When evaluating the benefits of utilizing unallocated storage, we match them with unallocated VMs of 1,400 servers. Since VMs with low storage utilization present a greater opportunity for harvesting, we highlight the capability of utilizing the heavily underutilized storage with BlockFlex.

Underutilized Capacity. We first analyze the impact on the underutilized storage capacity, summarized in Figure 15. We compare the average and maximum utilization when using BlockFlex against the baseline utilization for VMs without harvesting (originally shown in Figure 2). We see an average improvement of 1.25× (43% vs. 54% utilization) across all VMs and an improvement of 1.75× (20% vs. 35%) for those that had less than 60% storage utilization. This shows the benefits BlockFlex can obtain, especially when harvesting flash blocks from VMs with low storage utilization.

Next, we see that the maximum utilization across all of the VMs is increased by 1.37× (49% vs. 67%). We also observe that the over-provisioning we add to the predictions ensures that we do not fully utilize any regular VM. This reinforces



(a) TeraSort (b) ML Prep (c) PageRank

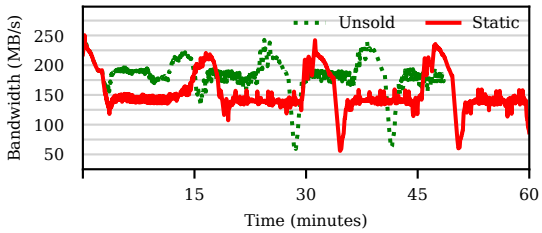
Figure 18: Performance benefits of storage harvesting for harvest VMs.

that BlockFlex has a low probability of reclamation.

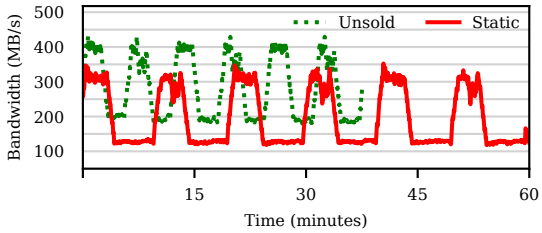
Underutilized Bandwidth. We now analyze the underutilized storage resource from a bandwidth perspective, summarized in Figure 16. Our results show a stable improvement of 1.34× (22% vs. 30%) for all VMs. BlockFlex also increases the maximum utilization by 1.27× (53% vs. 66%). As with underutilized storage, we avoid reclamations by not fully utilizing the bandwidth of regular VMs. This demonstrates that BlockFlex can improve both the bandwidth and capacity utilization of cloud storage from underutilized resources.

Unallocated Storage. We analyze the utilization improvement by harvesting unallocated VMs, presented in Figure 17. We observe that BlockFlex improves the overall utilization by 1.17× (69% vs. 81%). Servers with utilization below 60% are improved by 1.42× (45% vs. 64%).

For underutilized and unallocated storage resources, we observe 1.25× improvement on average, showing that BlockFlex can significantly use both underutilized and unsold storage resources to improve utilization. For extremely underutilized cases (under 60%), we observe 1.48× improvement on average. This shows that BlockFlex can successfully match the harvestable storage resources to the demands from harvest VMs.



(a) ML Prep Bandwidth



(b) PageRank Bandwidth

Figure 19: Read bandwidth of ML Prep and PageRank workloads after storage harvesting.

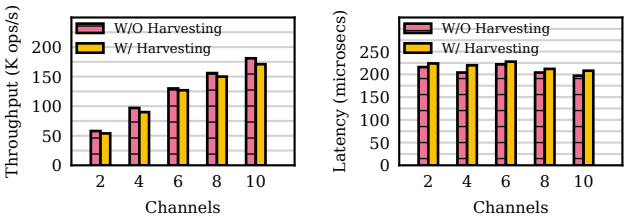
5.3 Improved Performance for Harvest VM

We examine how BlockFlex improves the performance of harvest VMs. The results are shown in Figure 18. We evaluate three different configurations: **Static:** the harvest VM is statically configured with 8 channels and does not harvest. This represents the current (baseline) storage virtualization. **Sold:** a 4-channel gSSD is allocated from channels occupied by a regular VM that uses 50% of its maximum bandwidth. **Unsold:** a 4-channel gSSD is allocated from unallocated channels. For both unsold and static, the gSSD is harvested after one hour. Before each experiment, we warm up the SSD to ensure GC will occur. We run all workloads for two hours.

By harvesting additional channels, the harvest VM has significantly improved bandwidth. As we compare the Sold scheme with the Static scheme, the workload performance is improved by 16–51% on average. For the Unsold scheme, the lack of interference with the regular VM improves the storage bandwidth by 22–60%. We observe the best improvement for PageRank, as its workload spends more time on I/O than TeraSort or ML Prep workloads. The Unsold scheme provides an additional 6% bandwidth improvement over the Sold scheme on average. As we translate this into the end-to-end execution time, we see an average performance improvement of 20% using Sold storage, and 25% improved performance using Unsold storage. This demonstrates the significant performance benefits BlockFlex can obtain for IO-intensive applications, when utilizing either sold or unsold storage.

Clearly, additional flash channels can benefit write-heavy workloads, as we increase the I/O parallelism. It is less clear whether additional channels can benefit read heavy workloads, as the harvested channels cannot immediately satisfy reads. To investigate this, we focus on the read bandwidth improvements in Figure 19. For both ML Prep and PageRank workloads, we see an increase of 10−21% after 5 minutes of harvesting. After the full 60 minutes, the average increase of the read bandwidth stabilizes and reaches an overall improvement of 22−60%.

Specifically, for ML Prep, we see a slight increase (10%) as



(a) Bandwidth (b) 95th Percentile Latency

Figure 20: Performance of a regular SSD with storage harvesting enabled.

we redirect writes to the additional channels immediately upon harvesting (0-5 minutes). Afterwards, as we start issuing writes and reads to the new channels, we see the read bandwidth benefit stabilizes at an improved level (24%). As for the PageRank workload, it shows relatively consistent benefit in the read bandwidth (60%). This is because PageRank workload is write intensive, during the first two minutes of harvesting. Thus, the PageRank data is aggressively written to the new harvested channels, which benefits the read bandwidth in return.

5.4 Performance Impact on Regular VM

To investigate the impact of storage harvesting on regular VMs, we examine the interference generated by the harvest VM. We run the YCSB workload-A with 10 threads in the regular VM, and vary the number of flash channels in its vSSD from 2 to 10. The database tables are striped across all the available channels in the vSSD. We first measure the throughput and tail latency (95th percentile latency) of the YCSB workload without enabling storage harvesting. After that, we create a harvest VM to run the ML Prep workload. The harvest VM will harvest all the channels of the regular VM, and we measure the performance of the regular VM after the harvesting.

As shown in Figure 20, the throughput of YCSB Workload-A decreases slightly, while the latency is almost constant as we increase the number of channels. The storage harvesting does not introduce much overhead (5.1% on average), since the regular VM always has the higher priority for its I/O requests and available storage bandwidth. We observe a similar overhead for the tail latency, demonstrating that the storage harvesting has negligible negative impact on the performance of regular VMs.

We also examine the interference caused by additional GC and storage reclamations. As indicated in §5.3, GC is enabled in our experiments. We believe the GC overhead can be further reduced with erase suspension available in modern SSDs [36, 70]. We wish to explore this feature in our future work. As for the overhead caused by storage reclamations, we observe that reclaiming an entire flash channel results in 1.5% slowdown in the average bandwidth of regular VMs. Such an overhead is acceptable in reality, as storage reclamations do not happen frequently over the entire lifetime of VMs.

Table 3: Learning overheads for each iteration in our predictors

Predictor	Training Time (millisecs)	Inference Time (millisecs)	Model Size (KB)
Bandwidth	10.3	2.5	22
Space	13.0	0.4	12
Bandwidth Duration	410.0	4.1	1153
Space Duration	42.0	0.3	510
Total	**475**	**7.3**	**1697**

5.5 Overhead Sources in BlockFlex

We now profile the overheads introduced by BlockFlex. We begin by analyzing the overheads introduced by the predictors. We present the summary of these overheads in Table 3. First, we measure the time consumed by training each predictor for one iteration of online training. As discussed in §4.4, each model is trained one iteration every three minutes. Since each duration predictor has multiple models, their training is more expensive than storage bandwidth and capacity predictors. In total, training all of the predictors consumes 0.48 seconds on our multi-core server. In this case, cloud platform operators do not need powerful hardware accelerators like GPUs to deploy BlockFlex. Since input sizes and training frequency do not change by workload, the training overhead is the same across all the workloads evaluated in this paper.

For each inference, the total execution time is 7.3 milliseconds. This overhead is also incurred once every three minutes, but can be further optimized. For example, we can decrease the inference frequency when a vSSD has generated a gSSD.

To store the predictors for each vSSD, BlockFlex allocates about 1.7MB memory space. It also allocates 4KB memory to store the history of bandwidth/capacity information used for training each iteration. This demonstrates the minimal performance and storage overheads of the predictors in BlockFlex.

We also profile the overheads of gSSD creation and lookup. They include the overheads of creating a new gSSD and harvesting free blocks from a regular vSSD. Since they only involve metadata operations, the overhead is 61 μs for creating a gSSD with 64GB. As gSSDs are created in the background, their creation overhead is not on the critical path. We organize the gSSD pool in sorted lists, the gSSD lookup takes 1.2 μs on average.

As we reclaim a gSSD from a harvest VM, its primary cost is on the erase of all the written blocks. Since we can parallelize the erase operations across channels, the limiting factor is the channel with the most allocated blocks. The total overhead is 17.1, 34.2, and 68.4 seconds for a channel (64GB) with 25%, 50%, 100% harvested, respectively. According to our study of various cloud traces, we observe that storage harvesting is infrequent (once every few hours). Additionally, compared to the lifetime of VMs in the cloud, the overhead of storage reclamation is relatively small, which has negligible impact on the performance of regular VMs.

6 Discussion and Future Work

Security implications of storage harvesting. A few potential security concerns may arise when sharing physical flash blocks in a cloud environment. First, we consider whether data could be leaked via harvested blocks. Since BlockFlex erases the flash blocks before creating/reclaiming the gSSD, it guarantees that user data will not be leaked through the storage harvesting. Second, we consider whether information could be leaked through the cached data, such as LBA-PBA mappings. As existing cloud infrastructure prevents access to the SSD virtualization, device driver, and controller layers without permission checking, therefore, even though a flash channel is shared across VMs, their accesses are protected. Third, we consider whether multiple VMs sharing a physical flash channel could suffer from side-channel attacks. It is actually hard for attackers to obtain meaningful information, since the variations could be caused by many factors, such as the number of co-located VMs or the CPU/memory contention.

Compatible with compute and memory harvesting. Upon the creation of harvest VMs, cloud platforms will allocate essential compute, memory, and storage resources. BlockFlex mainly targets storage harvesting to improve the overall cloud storage utilization, and improve the performance of applications bottlenecked by storage resources. It is compatible with prior studies on compute and memory harvesting [21, 69] for improving the whole-system resource utilization.

Semantic-aware storage harvesting. BlockFlex utilizes the vSSD interface in its implementation, making it transparent to applications in VMs. However, due to the lack of semantic information from upper-level applications, BlockFlex has to rely on the predictors to decide the harvestable and demanded storage resources. Additionally, preventing data loss is one of the key challenges when developing BlockFlex, allowing systems software to manage their data in harvested storage would be an alternative solution to address this challenge. Therefore, new APIs can be developed and exposed to popular software systems such as key-value stores and Hadoop Distributed File System (HDFS), which offers more flexibility for applications to manage their data in harvested storage.

7 Related Work

Storage virtualization and efficiency. Storage devices such as SSDs have been virtualized as system-wide shared resources for improved utilization in cloud platforms [30, 35, 44, 61, 62, 75]. Based on this, most recent studies focused on improving the performance isolation between collocated applications [7, 33, 34, 43, 50, 65]. However, our study (see §2) reveals that the cloud storage is still significantly underutilized. Ouyang et al. [52] identified the resource underutilization in the SSDs and developed the software-defined flash for cloud platforms. Similar to software-defined networking, software-defined flash is becoming a backbone technique in datacenters today [17, 57, 65]. However, most of them still use a static-allocation approach, which inevitably causes the waste of both storage capacity and bandwidth [12, 52]. Disaggregated storage architectures are proposed [41, 48, 51, 58, 68]. However, they still suffer from storage underutilization when we allocate disaggregated storage to VMs, due to the dynamic workload changes in VMs. BlockFlex addresses the storage underutilization problem by enabling storage harvesting in software-defined datacenters.

Resource harvesting in cloud platforms. Harvesting resources for VMs to improve the resource utilization is not a new concept in cloud platforms. Similar to the harvest VM, many studies have been developed recently, such as Spot VMs and burstable VMs [6, 8, 9, 21, 22, 60, 69]. However, they typically harvest compute and memory resources at a VM granularity. BlockFlex is the first work that focuses on storage harvesting, and addresses the unique challenges in storage harvesting and exception handling. Beyond harvesting unsold resources [6], we can also harvest underutilized allocated storage resources, while providing the performance and security isolation between regular VMs and harvest VMs.

Learning approaches for resource efficiency. Most recently, researchers started to leverage learning techniques to improve the task scheduling [54, 69, 77], cluster resource management [6, 11, 16, 46, 74], and performance optimizations [27, 38, 45, 78]. They showed that the learning-based approach is a promising method to address system optimization problems. However, it is still unclear how they can benefit the cloud storage. In this work, we apply the learning-based approach to improve the storage utilization within our storage harvesting framework. We customize the classical LSTM models for the predictions of harvestable and demanded storage resources, and show their efficiency in our evaluation.

8 Conclusion

In this paper, we first conduct a characterization study of the cloud storage utilization, and discloses that the low storage utilization exists pervasively in modern cloud platforms. To this end, we develop a learning-based storage harvesting framework BlockFlex, which can harvest both allocated and unallocated storage for evictable VMs. Our experiments show that BlockFlex can significantly improve the cloud storage utilization, while accelerating the storage performance of harvest VMs with minimal impact on the regular VMs.

Acknowledgments

We thank the anonymous reviewers and our shepherd Swami Sundararaman for their helpful comments and feedback. We thank Íñigo Goiri for providing part of the cloud traces for our study as well as insightful discussions. This work is supported by NSF CAREER Award 2144796, CCF-1919044, CNS-1850317 and a grant from Western Digital Technologies, Inc.

References

[1] Ahmed Abulila, Vikram S Mailthody, Zaid Qureshi, Jian Huang, Nam Sung Kim, Jinjun Xiong, and Wen-mei Hwu. FlatFlash: Exploiting the Byte-Accessibility of SSDs within A Unified Memory-Storage Hierarchy. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*, Providence, RI, USA, 2019.

[2] Albumentations Image Processing. https://github.com/albumentations-team/albumentations, 2021.

[3] Alibaba Cluster Trace. https://github.com/alibaba/clusterdata/blob/master/cluster-trace-v2018/trace_2018.md.

[4] Azure Spot VM. https://azure.microsoft.com/en-us/services/virtual-machines/spot/.

[5] Amazon Elastic Compute Cloud. Amazon EC2 Spot Instances. https://aws.amazon.com/ec2/spot/.

[6] Pradeep Ambati, Inigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, Marcus Fontoura, and Ricardo Bianchini. Providing slos for resource-harvesting vms in cloud platforms. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, November 2020.

[7] Sebastian Angel, Hitesh Ballani, Thomas Karagiannis, Greg O'Shea, and Eno Thereska. End-to-end performance isolation through virtual datacenters. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*, Broomfield, CO, October 2014.

[8] Amazon AWS. Burstable performance instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances.html, 2020.

[9] Microsoft Azure. Introducing B-Series, our new burstable VM size. https://azure.microsoft.com/en-us/blog/introducing-b-series-our-new-burstable-vm-size/, 2017.

[10] Azure cloud trace. https://github.com/Azure/AzurePublicDataset, 2019.

[11] Ricardo Bianchini, Marcus Fontoura, Eli Cortez, Anand Bonde, Alexandre Muzio, Ana-Maria Constantin, Thomas Moscibroda, Gabriel Magalhaes, Girish Bablani, and Mark Russinovich. Toward ml-centric cloud platforms. *Communication of ACM*, 63(2), January 2020.

[12] Matias Bjørling, Javier Gonzalez, and Philippe Bonnet. Lightnvm: The linux open-channel SSD subsystem. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)*, Santa Clara, CA, February 2017.

[13] Amazon Elastic Compute Cloud, Burstable Performance Instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances.html, 2020.

[14] Microsoft Azure Cloud. Configure online virtual hard disk resize. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn282284(v=ws.11), 2016.

[15] Cloud flash storage: SSD options from AWS, Azure, and GCP. https://www.computerweekly.com/feature/Cloud-flash-storage-SSD-options-from\-AWS-Azure-and-GCP, 2020.

[16] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17)*, Shanghai, China, 2017.

[17] Project denali to define flexible ssds for cloud-scale applications. https://azure.microsoft.com/en-us/blog/project-denali-to-define-flexible\-ssds-for-cloud-scale-applications/.

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009.

[19] Andreas E Dilger. Online ext2 and ext3 filesystem resizing. In *Ottawa Linux Symposium*, page 117, 2002.

[20] Ev3 and Esv3-Series. https://docs.microsoft.com/en-us/azure/virtual-machines/ev3-esv3-series, 2021.

[21] Alexander Fuerst, Stanko Novaković, Íñigo Goiri, Gohar Irfan Chaudhry, Prateek Sharma, Kapil Arya, Kevin Broas, Eugene Bak, Mehmet Iyigun, and Ricardo Bianchini. Memory-harvesting vms in cloud platforms. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*, Lausanne, Switzerland, February 2022.

[22] Google. Our data centers now work harder when the sun shines and wind blows. https://blog.google/inside-google/infrastructure/data-centers-work-harder-sun-shines-wind-blows, 2020.

[23] Google Cluster Trace. https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md.

[24] Google Cloud. Preemptible VM Instances. https://cloud.google.com/compute/docs/instances/preemptible.

[25] Graphchi. https://github.com/GraphChi/graphchi-cpp, 2021.

[26] Hadoop TeraSort. https://hadoop.apache.org/docs/r3.2.0/api/org/apache/hadoop/examples/terasort/package-summary.html, 2021.

[27] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S. Gunawi. LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, November 2020.

[28] Timothy Harris. A pragmatic implementation of non-blocking linked-lists. In *Proceedings of the 15th International Symposium on Distributed Computing (DISC 2001)*, Lisbon, Portugal, 2001.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[30] Jian Huang, Anirudh Badam, Laura Caulfield, Suman Nath, Sudipta Sengupta, Bikash Sharma, and Moinuddin K. Qureshi. Flashblox: Achieving both performance isolation and uniform lifetime for virtualized ssds. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)*, Santa Clara, CA, February 2017.

[31] Jian Huang, Anirudh Badam, Moinuddin K. Qureshi, and Karsten Schwan. Unified Address Translation for Memory-Mapped SSD with FlashMap. In *Proceedings of the 42nd International Symposium on Computer Architecture (ISCA'15)*, Portland, OR, June 2015.

[32] IBM. Ibm flash storage and software defined storage. *White Paper*, 2017.

[33] Giorgos Kappes and Stergios V. Anastasiadis. Libservices: Dynamic storage provisioning for multitenant i/o isolation. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys'20)*, Tsukuba, Japan, 2020.

[34] Giorgos Kappes and Stergios V. Anastasiadis. A user-level toolkit for storage i/o isolation on multitenant hosts. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC'20)*, Virtual Event, USA, 2020.

[35] Jaeho Kim, Donghee Lee, and Sam H. Noh. Towards SLO Complying SSDs Through OPS Isolation. In *Proc. FAST'15*, Santa Clara, CA, February 2015.

[36] Shine Kim, Jonghyun Bae, Hakbeom Jang, Wenjing Jin, Jeonghun Gong, Seungyeon Lee, Tae Jun Ham, and Jae W. Lee. Practical erase suspension for modern low-latency SSDs. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC'19)*, Renton, WA, July 2019.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Daniar H. Kurniawan, Levent Toksoz, Anirudh Badam, Tim Emami, Sandeep Madireddy, Robert B. Ross, Henry Hoffmann, and Haryadi S. Gunawi. Ionet: Towards an open machine learning training ground for i/o performance prediction. *Technical Report*, 2021.

[39] Dongup Kwon, Junehyuk Boo, Dongryeong Kim, and Jangwoo Kim. FVM: Fpga-assisted virtual device emulation for fast, scalable, and flexible storage virtualization. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, November 2020.

[40] Sungjin Lee, Ming Liu, Sangwoo Jun, Shuotao Xu, Jihong Kim, and Arvind. Application-managed flash. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*, Santa Clara, CA, February 2016.

[41] Sergey Legtchenko, Hugh Williams, Kaveh Razavi, Austin Donnelly, Richard Black, Andrew Douglas, Nathanael Cheriere, Daniel Fryer, Kai Mast, Angela Demke Brown, Ana Klimovic, Andy Slowey, and Antony Rowstron. Understanding Rack-Scale disaggregated storage. In *Proceedings of the 9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'17)*, Santa Clara, CA, July 2017.

[42] Jacob Leverich and Christos Kozyrakis. Reconciling high server utilization and sub-millisecond quality-of-service. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys'14)*, Amsterdam, The Netherlands, 2014.

[43] Ning Li, Hong Jiang, Dan Feng, and Zhan Shi. PSLO: Enforcing the Xth Percentile Latency and Throughput SLOs for Consolidated VM Storage. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys'16)*, London, United Kingdom, April 2016.

[44] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA'15)*, Portland, OR, June 2015.

[45] Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, and Colin Raffel. Learning-based memory allocation for c++ server workloads. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, Lausanne, Switzerland, 2020.

[46] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM'19)*, Beijing, China, 2019.

[47] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. Taming performance variability. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*, Carlsbad, CA, October 2018. USENIX Association.

[48] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In Hwan Doh, and Arvind Krishnamurthy. Gimbal: Enabling multi-tenant storage disaggregation on smartnic jbofs. In *Proceedings of the 2021 Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'21)*, Virtual Event, USA, 2021.

[49] Pulkit A. Misra, Inigo Goiri, Jason Kace, and Ricardo Bianchini. Scaling distributed file systems in resource-harvesting datacenters. In *Proceedings of the 2017 USENIX Annual Technical Conference (ATC'17)*, Santa Clara, CA, July 2017.

[50] Dushyanth Narayanan, Eno Thereska, Austin Donelly, Sameh Elnikety, and Antony Rowstron. Migrating server storage to ssds, analysis of tradeoffs. In *Proceedings of the Fourth European Conference on Computer Systems (EuroSys'09)*, Nuremberg, Germany, March 2009.

[51] Nutanix Distributed Storage. https://www.nutanix.com/products/acropolis/distributed-storage, 2022.

[52] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, Yong Wang, and Yuanzheng Wang. Sdf: Software-defined flash for web-scale internet storage systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*, Salt Lake City, UT, 2014.

[53] PyTorch. https://pytorch.org/, 2021.

[54] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. FIRM: An intelligent fine-grained resource management framework for slo-oriented microservices. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, November 2020.

[55] RocksDB. https://github.com/facebook/rocksdb, 2021.

[56] Software-defined data center. https://en.wikipedia.org/wiki/Software-defined_data_center.

[57] Software-defined storage. https://en.wikipedia.org/wiki/Software-defined_storage.

[58] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*, Carlsbad, CA, October 2018.

[59] Prateek Sharma, Ahmed Ali-Eldin, and Prashant Shenoy. Resource deflation: A new approach for transient resource reclamation. In *Proceedings of the Fourteenth European Conference on Computer Systems (EuroSys'19)*, Dresden, Germany, 2019.

[60] Prateek Sharma, Stephen Lee, Tian Guo, David Irwin, and Prashant Shenoy. Spotcheck: Designing a derivative iaas cloud on the spot market. In *Proceedings of the 10th European Conference on Computer Systems (EuroSys'15)*, 2015.

[61] David Shue, Michael J. Freedman, and Anees Shaikh. Performance isolation and fairness for multi-tenant cloud storage. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI'12)*, Hollywood, CA, October 2012.

[62] Dharma Shukla, Shireesh Thota, Karthik Raman, Madhan Gajendran, Ankur Shah, Sergii Ziuzin, Krishnam Sundama, Miguel Gonzalez Guajardo, Anna Wawrzyniak, Samer Boshra, Renato Ferreira, Mohamed Nassar, Michael Koltachev, Ji Huang, Sudipta Sengupta, Justin Levandoski, and David Lomet. Schema-agnostic indexing with azure documentdb. In *Proceedings of the 41st International Conference on Very Large Databases (VLDB'15)*, Kohala Coast, Hawaii, September 2015.

[63] Error Messages for Azure Spot Virtual Machines and Scale Sets. https://docs.microsoft.com/en-us/azure/virtual-machines/error-codes-spot.

[64] Software-Enabled Flash for Hyperscale Data Centers. https://searchstorage.techtarget.com/post/Software-Enabled-Flash-for-Hyperscale\-Data-Centers, 2021.

[65] Eno Thereska, Hitesh Ballani, Greg O'Shea, Thomas Karagiannis, Antony Rowstron, Tom Talpey, Richard Black, and Timothy Zhu. Ioflow: A software-defined storage architecture. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP'13)*, 2013.

[66] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys'15)*, Bordeaux, France, 2015.

[67] VMware. Growing, thinning, and shrinking virtual disks in esxi. https://kb.vmware.com/s/article/1002019, 2021.

[68] VMWare VSAN. https://www.vmware.com/products/vsan.html, 2022.

[69] Yawen Wang, Kapil Arya, Marios Kogias, Manohar Vanga, Aditya Bhandari, Neeraja J. Yadwadkar, Siddhartha Sen, Sameh Elnikety, Christos Kozyrakis, and Ricardo Bianchini. Smartharvest: Harvesting idle cpus safely and efficiently in the cloud. In *Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys'21)*, 2021.

[70] Guanying Wu and Xubin He. Reducing SSD read latency via NAND flash program and erase suspension. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*, San Jose, CA, February 2012.

[71] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*, Tel-Aviv, Israel, 2013.

[72] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *arXiv preprint arXiv:1205.6233*, 2012.

[73] Yahoo! Cloud Serving Benchmark. https://github.com/brianfrankcooper/YCSB/wiki, 2021.

[74] Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. RLScheduler: An Automated HPC Batch Job Scheduler Using Reinforcement Learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'20)*, Virtual Event, November 2020.

[75] Ning Zhang, Junichi Tatemura, Jignesh M. Patel, and Hakan Hacigumus. Re-evaluating Designs for Multi-Tenant OLTP Workloads on SSD-based I/O Subsystems. In *Proceedings of the SIGMOD'14*, Snowbird, UT, June 2014.

[76] Yunqi Zhang, George Prekas, Giovanni Matteo Fumarola, Marcus Fontoura, Inigo Goiri, and Ricardo Bianchini. History-based harvesting of spare cycles and storage in large-scale datacenters. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA, November 2016.

[77] Zhiheng Zhong, Minxian Xu, Maria Alejandra Rodriguez, Chengzhong Xu, and Rajkumar Buyya. Machine learning-based orchestration of containers: A taxonomy and future directions. *Computing Research Repository (CoRR)*, abs/2106.12739, 2021.

[78] Giulio Zhou and Martin Maas. Learning on distributed traces for data center storage systems. In *Proceedings of the Machine Learning and Systems (MLSys'21)*, Austin, TX, March 2021.