2018

I/O Workload Management for All-Flash Datacenter Storage Systems Based on Total Cost of Ownership

Zhengyu Yang, *Northeastern University*
Manul Awasthi
Mrinmoy Ghosh
Janki Bhimani, *Northeastern University*
Ningfang Mi

I/O Workload Management for All-Flash Datacenter Storage Systems Based on Total Cost of Ownership

Zhengyu Yang[1], Manu Awasthi[2], Mrinmoy Ghosh[3], Janki Bhimani[1] and Ningfang Mi[1]

[1]*Northeastern University, USA,* [2]*Ashoka University, India,* [3]*Samsung Semiconductor Inc., USA*

Abstract—Recently, the capital expenditure of flash-based Solid State Driver (SSDs) keeps declining and the storage capacity of SSDs keeps increasing. As a result, all-flash storage systems have started to become more economically viable for large shared storage installations in datacenters, where metrics like Total Cost of Ownership (TCO) are of paramount importance. On the other hand, flash devices suffer from write amplification, which, if unaccounted, can substantially increase the TCO of a storage system. In this paper, we first develop a TCO model for datacenter all-flash storage systems, and then plug a Write Amplification model (WAF) of NVMe SSDs we build based on empirical data into this TCO model. Our new WAF model accounts for workload characteristics like write rate and percentage of sequential writes. Furthermore, using both the TCO and WAF models as the optimization criterion, we design new flash resource management schemes (MINTCO) to guide datacenter managers to make workload allocation decisions under the consideration of TCO for SSDs. Based on that, we also develop MINTCO-RAID to support RAID SSDs and MINTCO-OFFLINE to optimize the offline workload-disk deployment problem during the initialization phase. Experimental results show that MINTCO can reduce the TCO and keep relatively high throughput and space utilization of the entire datacenter storage resources.

Index Terms—Flash Resource Management, Total Cost of Ownership Model, SSD Write Amplification, NVMe, Wearout Prediction, Workload Sequentiality Pattern, Data Center Storage System, RAID

1 INTRODUCTION

The world has entered the era of "Big Data", when large amount of data is being collected from a variety of sources, including computing devices of all types, shapes and forms. This data is then being pushed back to large, back-end datacenters where it is processed to extract relevant information. As a result of this transformation, a large number of server-side applications are becoming increasingly I/O intensive. Furthermore, with the amount of data being gathered increasing with every passing day, the pressure on the I/O subsystem will continue to keep on increasing [1].

To handle this high I/O traffic, datacenter servers are being equipped with the best possible hardware available encompassing compute, memory, networking and storage domains. Traditionally, I/O has been handled by hard disk drives (HDDs). HDDs have the benefit of providing an excellent economic value ($/GB), but being built with mechanical moving parts, they suffer from inherent physical throughput limitations, especially for random I/Os. To counter these performance limitations, solid state devices (SSDs) have recently begun to emerge as a viable storage alternative to HDDs. In the recent past, SSDs have gained widespread adoption owing to reduced costs from the economies of scale. Datacenters, especially popular public cloud providers (e.g., [2], [3]) have been at the forefront of adopting flash technology.

Nevertheless, during this revolutionary change in cloud storage systems, flash-based SSDs face two major concerns: cost and write amplification (WA). Firstly, the costs of owning (purchasing and maintaining) SSDs can still be very high. Balancing the trade-off between performance and economy is still an uphill battle. Currently, Total Cost of Ownership (TCO), comprising of two major costs, i.e., Capital and Operating Expenditures, remains a popular metric. However, only a few prior studies have focused on the TCO of SSDs in datacenters, especially with the consideration of the cost of SSD's wornout.

Secondly, SSDs have limited write cycles and also suffer from write amplification which is caused by a number of factors specific to flash devices including erase-before-rewrite, background garbage collection, and wear leveling. In fact, the Write Amplification Factor (WAF, will be defined in Sec. 2) of an SSD is a direct function of the I/O traffic it experiences. The I/O traffic, in turn, comprises of a number of different factors like the fraction of writes (as opposed to reads), the average size of I/O requests, the arrival rate of I/Os, and the ratio of sequential I/O patterns (as opposed to random I/O) in the overall I/O stream. Greater WAF can significantly reduce the lifetime and increase the ownership cost of flash devices.

Therefore, in this paper, we focus on the problem of deploying and allocating applications to a shared all-flash storage system of modern datacenters in

This work was completed during Zhengyu Yang's internship at Samsung Semiconductor Inc. This work was partially supported by National Science Foundation Career Award CNS-1452751 and AFOSR grant FA9550-14-1-0160.

order to reduce WAF and TCO. In detail, workloads (streams from an application) have different features, but in a long term of view, the I/O pattern of the same application can be characterized. Thus, we can address the above two concerns by investigating the relationship between workload patterns and WAF and then leveraging the relationship to develop new TCO models. In our experiments, we found that workloads with different sequential ratios have varying write amplifications even on the same SSD, which changes the lifetime of the device and eventually affects the TCO [4], [5]. We are thus motivated to evaluate storage systems from a cost perspective that includes many dimensions such as maintenance and purchase cost, device wornout, workload characteristics, and total data amount that can be written to the disk, etc. Therefore, in this paper, we make the following contributions to achieve this goal.

- We conduct real experiments to measure and characterize the write amplification under different workloads, and reveal the relationship between write amplification and workload sequential ratio for each disk with fixed Flash Translation Layer (FTL) specs.
- We propose a new TCO model while considering multiple factors like SSD lifetime, workload sequentiality, write wornout and the total writes.
- We propose statistical approaches for calculating components that are essential for computing the TCO but cannot (practically) be measured from SSDs during runtime, such as write amplification and wornout of each SSD.
- Based on our TCO model, we develop a set of new online adaptive flash allocation managers called "MINTCO", which leverage our TCO model to dynamically assign workloads to the SSD disk pool. The goals of MINTCO are: (1) to minimize the TCO, (2) to maximize client throughput as many as possible, and (3) to balance the load among SSD devices and best utilize SSD resources.
- We present MINTCO-RAID to support RAID mode SSD arrays through an approximation approach.
- We develop MINTCO-OFFLINE to support *offline* allocation scenario, where the datacenter manager needs to decide how many disks the datacenter needs and how to allocate workloads to the datacenter.

Lastly, we evaluate our new models and approaches using real world trace-driven simulation. Our experimental results show that MINTCO can reduce the TCO by up to 90.47% compared to other traditional algorithms. Meanwhile, it guarantees relatively high throughput and spatial utilization of the entire SSD-based datacenter. Moreover, MINTCO-OFFLINE can also reduce TCO by up to 83.53% TCO compared to the naive greedy allocation.

The remainder of this paper is organized as follows. Sec. 2 investigates the cause of write amplification on SSDs. Sec. 3 presents the details of our TCO models. Sec. 4 proposes two versions of our MINTCO allocation algorithms. Sec. 5 measures the WAF of NVMe disks under different workload patterns, and evaluates our allocation algorithms. Sec. 6 describes the related work. Finally, we summarize the paper and discuss the limitations and future work plan of this research in Sec. 7.

2 WRITE AMPLIFICATION FACTOR

Write amplification factor ("WAF", henceforth referred to as "A") is a commonly used metric to measure the write amplification degree. WAF is an undesirable phenomenon associated with flash devices where the actual amount of data written to the device is larger than the logical amount of data written by a workload. We define WAF as the ratio between the total physical write data written by the SSD and the total logical data written by the workload: $A = \frac{W_P}{W_L}$, where W_L denotes the logical write amount (in bytes), and W_P denotes the device-level physical I/O writes as seen by the SSD. Fig. 1 illustrates the logical and physical writes all the way from the application, through the OS, to the SSD. Large values of A lead to increase I/O latency, shorten the SSD's working lifetime, and increase power consumption.

Where does the SSD write amplification come from? Flash devices have an unique property that they cannot be re-written unless they have been erased. Also, the minimum granularity of an erase operation is in the order of MBs (e.g., blocks), while the granularity of writes is much smaller, in the order of KBs (e.g., pages). Meanwhile, flash devices have limited write life cycles. Thus, for the purpose of wear-leveling, the logical address space in flash devices is dynamically mapped to the physical space and the mapping changes with every write. Flash devices have a software called FTL (Flash Translation Layer) running on them to manage the erase before re-write and wear-leveling requirements. The FTLs have to schedule periodic garbage collection events to de-fragment their write data. These garbage collection events can lead to extra writes that have not been generated by the host. Additionally, SSD reserves a user-invisible space (i.e., over-provision), which is helpful to reduce the WAF during these above-mentioned events to some extent. However, since flash devices have limited write-erase cycles, the mismatch between the two (logical and physical) types of writes can still cause the SSD to fail much more quickly than expected. On the other hand, besides these device hardware related factors, write amplification is also affected by I/O workload-related factors, such as mounted file systems and workload traffic patterns. Notice that in this paper, we are not aiming to change SSD's FTL algorithms, instead, we mainly focus on the impact of different workload patterns to the WAF under SSDs (i.e., FTLs of SSDs are fixed in the datacenter after deployment).

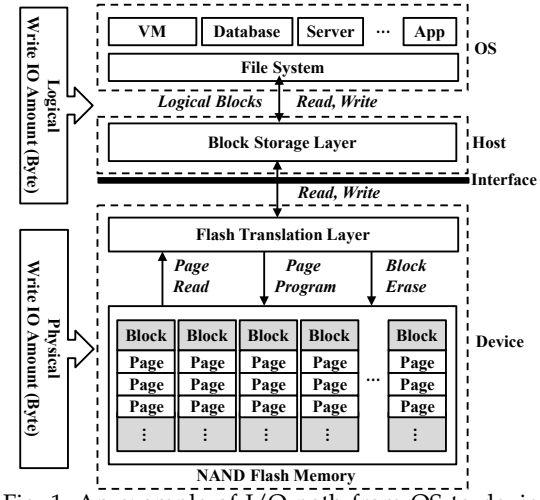

Fig. 1: An example of I/O path from OS to device.

The existing analytical models [6] for WAF build the relationship between workload characteristics and WAF based on the different garbage collection policies (i.e., cleaning algorithms) and the impacts of the hot and cold data distribution [7]. However, these models ignore a factor that is becoming increasingly important, especially in the NoSQL database community, traffic patterns of workloads in terms of sequential and random ratio experienced by the SSD [8]. With the proliferation of log structured merge tree (LSM tree) based NoSQL databases, there is a lot of uptick in the amount of sequential traffic being sent to the SSDs. LSM-tree based databases capture all the writes into a large in-memory buffer, and when the buffer is full, it is flushed to disk as a large, multi-gigabyte sequential write. Another similar case is write-intensive workloads that execute within virtual machines, where most of the write traffic to the SSD is usually sent out as large, sequential writes [9]. Hence, it is becoming increasingly essential to understand the WAF, performance of SSDs, device worn-out, and most importantly, the total owning cost of datacenters from a workload-centric view.

3 TCO Models of Storage Systems in All-Flash Datacenters

TCO of an all-flash datacenter's storage system is a mix of a large number of items. Broadly speaking, these items can be broken down into two major categories: (1) Capital Expenditure (CapEx), and (2) Operating Expenditure (OpEx). Capital Expenditure refers to the amount of money that needs to be spent in setting up a facility. These include the cost of buying individual components of the servers, power supplies, racks that house the servers, among other things. OpEx, on the other hand, is the amount of money that is spent in the day-to-day operation of a datacenter. Examples of OpEx include electricity costs and personnel costs (required for maintaining the datacenter). CapEx is traditionally a large, one time expenditure [10] while OpEx consists of small(er), recurring expenditures. In this section, we develop a

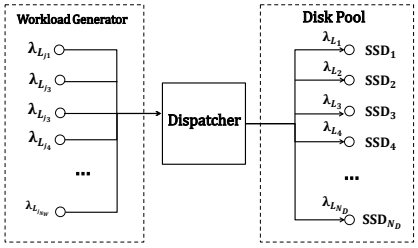

Fig. 2: Model of a datacenter storage system.

TCO model for an SSD intensive datacenter, based on the characteristics of the workloads (i.e., application I/O streams) that are scheduled on to those SSD devices. Our TCO model focuses specifically on the costs related to acquiring (i.e., CapEx) and maintaining (i.e., OpEx) SSDs in a datacenter.

3.1 Workload and Storage Models

First, we briefly explain our assumptions about datacenters, their workloads and storage systems. We assume the datacenter storage system to be a large pool of SSD devices. This helps us abstract the problem of modeling SSDs from a per-server resource to a pool of datacenter-wide resources. We then model the storage system of such a datacenter as a *workload-to-disk allocation problem*, as shown in Fig. 2. In this model, we have a pool of N_D SSDs as shown in Fig. 2. Meanwhile, there are N_W applications (workloads) that submit I/O requests with logical write rates $\lambda_{L_J i}$ (where $1 \leq i \leq N_W$, and "L_J" stands for "logical" and "job"), as seen in the left hand box of Fig. 2. To complete the connection between I/O requests and the SSDs, we assume an allocation algorithm that is used by the dispatcher to assign I/O workloads to different SSDs. Each workload has its own characteristic, and arrives at the dispatcher at different times. Multiple workloads can be assigned to the same disk as long as the capacity (e.g., space and throughput) of the disk is sufficient, such that the logical write rate (λ_{L_i}) to disk i is the summation of logical write rates from the workloads in the set $\mathbb{J}_i$ that are allocated to that SSD, i.e., $\lambda_{L_i} = \sum_{j \in \mathbb{J}_i} \lambda_{L_J j}$. We summarize our main assumptions in the following subsections.

3.1.1 I/O Workload with Certain Properties

"Workload" is defined as an endless logical I/O stream issued by applications. Particularly, from a long-term view (e.g., years), characteristics of workloads, such as sequential ratio, daily write rate, read-write ratio, working set size, and re-access ratio, can be abstracted as (almost) fixed values. Workloads may arrive to the datacenter at different times. Once a workload arrives, the dispatcher assigns it to one certain disk (or multiple disks for RAID mode SSDs, see Sec. 4.3), and the disk(s) will execute this workload until the disk(s) is (are) "dead" (i.e., SSD write cycle limit is reached), or the workload finishes. We ignore the overhead (such as time and energy consumption) during the workload deployment.

3.1.2 Isolation among Multiple Workloads on a Single SSD

Multiple workloads can be assigned to a single SSD, and have separate and independent working sets (i.e., address spaces and segments are isolated). Therefore, the cross-workloads effects along I/O path due to interleaving working sets are negligible.

3.1.3 SSD's Write Amplification Model

We use the WAF model to capture the behavior of an SSD under a workload with a specific I/O pattern. Our WAF model can estimate the WAF of each disk by using the sequentiality information of multiple workloads that are concurrently executing at a particular SSD. The write wornout of each SSD can further be estimated using the runtime status of that SSD's WAF.

3.2 Total Cost of Ownership Model

Owning and maintaining a low cost SSD-intensive datacenter is critically important. TCO has been widely adopted to evaluate and assess storage subsystem solutions for traditional hard drives. However, to the best of our knowledge, there is no standard formula for calculating the TCO of the SSD-intensive storage subsystem. In order to comprehensively access the expenditure of a datacenter, a generic TCO model should consider purchasing and maintenance costs, service time, served I/O amount and device wornout. We present the following models to calculate the TCO. As we mentioned, two major types of costs: CapEx (C_{I_i}) and OpEx (C_{M_i}) are considered in the basic TCO model. In detail, $C_{I_i} = C_{Purchase_i} + C_{Setup_i}$ and $C'_{M_i} = C'_{Power_i} + C'_{Labor_i}$, where $C_{Purchase_i}$ and C_{Setup_i} are one-time cost ($) of device purchase and device setup, and C'_{Power_i} and C'_{Labor_i} are power and maintenance labor cost rate ($/day). Although CapEx ($C'_{I_i}$) is one time cost, OpEx (C'_M) is a daily rate and the TCO should be depend on the amount of time that an SSD has been used. Therefore, we need to attach a notion of *time* to TCO. Assume we know the expected life time (T_{Lf_i}) of each disk (i.e., $T_{Lf_i} = T_{D_i} - T_{I_i}$, where T_{D_i} and T_{I_i} are the time when the disk i is completely worn out and the time when it was started accepting its first request, respectively), the total cost for purchasing and maintaining a pool of SSDs can be calculated as:

$$TCO = \sum_{i=1}^{N_D} (C_{I_i} + C'_{M_i} \cdot T_{Lf_i}), \quad (1)$$

where N_D is the number of disks in the pool. Fig. 3(a) also illustrates an example from time stamp view, where I/O workloads keep arriving and thus the physical write rate of disk i increases accordingly. However, Eq. 1 does not reflect SSD wornout at all, which is highly coupled with the workload arrival distribution. For instance, in a datacenter consisted of the same type of SSDs, the SSD running workloads with the highest physical write rate may always have the highest TCO value (i.e., its $C'_{M_i} \cdot T_{Lf_i}$ is the greatest among all) according to Eq. 1, since this SSD is probably the last one to be worn out. However, this SSD may have a larger WAF due to the workload pattern, while others that died earlier may have smaller WAFs and can serve more logical write I/O amounts under the same cost. Therefore, to conduct a fair and meaningful comparison, we introduce the data-averaged TCO rate (TCO') from the perspective of the cost vs. the total amount of (logical) data served (written) to an SSD as follows.

$$TCO' = \frac{\sum_{i=1}^{N_D} (C_{I_i} + C'_{M_i} \cdot T_{Lf_i})}{\sum_{j=1}^{N_W} D_j}, \quad (2)$$

where $\sum_{j=1}^{N_W} D_j$ is the total *logical* data write amount for all N_W workloads. Again, we use logical writes as a proxy for physical writes not only because the former is much easier to obtain for most workloads, but also because by being normalized by the logical writes, the TCO' is able to reflect the WAF and judge the disk-level wear leveling performance of different allocation algorithms.

3.3 Calibrating TCO Models

The models developed in the prior section have all assumed that certain parameters for TCO calculation (e.g., total logical data write amount, expected lifetime, etc.) are readily available or measurable. However, it is impractical to measure some parameters that are necessary in our TCO models. In this section, we propose a mathematical approach of estimating those hard to be directly measured parameters.

3.3.1 Total Logical Data Writes $\sum_{j=1}^{N_W} D_j$

Given workload j's logical write rate λ_{L_j}, arrival time T_{A_j} and estimated time of death ($T_{D(j)}$) of workload j 's host disk $D(j)$, we can calculate the total amount of data written by all the jobs over their course of execution as: $\sum_{j=1}^{N_W} D_j = \sum_{j=1}^{N_W} \lambda_{L_j}(T_{D(j)} - T_{A_j})$, where λ_{L_j} is the logical data write rate of workload j. The only unknown parameter left is $T_{D(j)}$, which can be obtained by calculating each host disk's expected lifetime.

3.3.2 Expected Lifetime T_{Lf_i}

The lifetime of a disk depends not only on the write traffic from the currently executing jobs, but also on those jobs that have already been deployed on the disk. Furthermore, we also need to account for the effects of the *combined* write traffic of the workloads that are concurrently executing on a disk. As shown in Fig. 3(a), the lifetime of disk i is the period from T_{I_i} to T_{D_i}. We further split the lifetime into two phases: (1) all (accumulated) working epochs (T_{W_i}) until the last workload arrives, i.e., $T_{W_i} = T_{R_i} - T_{I_i}$, where T_{R_i} is the assigned time of the most recent workload; and (2) expected work lifetime (T_{E_i}) from T_{R_i} to the expected death time, i.e., $T_{E_i} = T_{D_i} - T_{R_i}$. The former is easy to monitor, and the latter is further estimated as the available remaining write cycles of disk i divided by the physical data write rate (λ_{P_i}) of disk i from T_{Ri}. Moreover, λ_{P_i} can be calculated as disk i's logical data write rate (λ_{L_i}) times disk i's WAF (A_i). Thus, we have $T_{Lf_i} = T_{D_i} - T_{I_i} = T_{W_i} + T_{E_i} = (T_{R_i} - T_{I_i}) + \frac{W_i - w_i}{\lambda_{P_i}} = (T_{R_i} - T_{I_i}) + \frac{W_i - w_i}{\lambda_{L_i} \cdot A_i} = (T_{R_i} - T_{I_i}) + \frac{W_i - w_i}{\lambda_{L_i} \cdot f_{seq}(S_i)}$, where

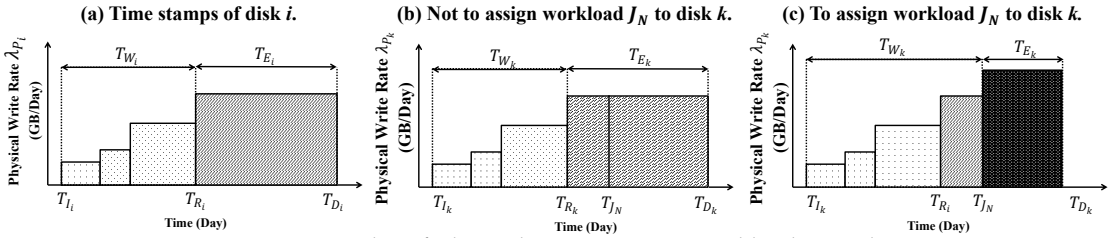

Fig. 3: Examples of physical write rates vs. workload arrivals.

A_i, W_i, w_i and S_i are the WAF function, the total write limit, current write count (wornout), and sequential ratio of all running workloads of disk i, respectively. Since the SSDs' hardware are fixed, we denote A_i as a function of workload's write I/O sequential ratio (f_{seq}) of disk i, which will be validated and regressed in our experimental section (Sec. 5.1.5). In fact, we can plug any WAF model into this TCO model. As of now, we also know T_{R_i}, T_{I_i} and W_i, and what we need to estimate next are the remaining parameters, i.e., λ_{L_i}, S_i and w_i.

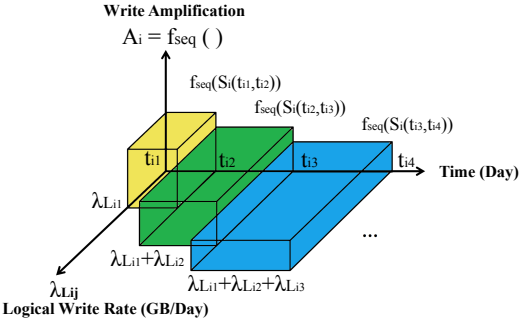

Fig. 4: An example of write worn out count estimation.

3.3.3 *Logical Write Rate of Workloads on Disk* λ_{L_i}

For disk i, its logical write rate λ_{L_i} should be the sum of all its assigned workloads' logical write rates, i.e., $\lambda_{L_i} = \sum_{j \in \mathbb{J}_i} \lambda_{L_{ij}}$, where $\mathbb{J}_i$ is the set of workloads running on disk i. Notice that there is a boundary case during the very early stage when no workloads have been assigned to the disk i (i.e., $\mathbb{J}_i = \emptyset$), such that $\frac{W_i - w_i}{\lambda_{L_i} \cdot f_{seq}(S_i)}$ becomes infinite. To avoid such an extreme case, we conduct a warming up process that assigns at least one workload to each disk. Only after this warming up phase is done, we start to calculate T_{Lf_i}.

3.3.4 *Sequential Ratio of Workloads on Disk* S_i

In order to calculate the write amplification A_i in Sec. 3.3.2, we need to know the sequential ratio of multiple workloads that are assigned to one disk. Unlike the logical write rate, the combined sequential ratio of multiple workloads is not equal to the sum of sequential ratios of all workloads. Our estimating solution is to assign a weight to each workload stream's sequential ratio and set the weight equal to the workload's logical data write rate. Hence, for multiple workloads running on the disk, we can calculate the overall sequential ratio as: $S_i = \frac{\sum_{j \in \mathbb{J}_i} \lambda_{L_{ij}} S_{ij}}{\sum_{j \in \mathbb{J}_i} \lambda_{L_{ij}}}$. where $\lambda_{L_{ij}}$ and S_{ij} are the logical write rate and sequential ratio of jth workload running on disk i. Appendix 1 shows the details of our implementation of sequential ratio estimator.

3.3.5 *Write Wornout Count of Disk* w_i

The last item we need to estimate is the current physical write count w_i (in Sec. 3.3.2) inside each SSD device. It is hard to exactly measure the overall write count of an SSD during its lifetime. However, we can estimate the current write count by adding the estimated write counts of all the workloads over all past epochs. For each epoch, we multiply the total logical write rate by the corresponding WAF to get the physical write rate. By iterating this process for all epochs, we can finally get the total write wornout count for each disk. Fig. 4 shows a simple example of estimating a disk's write wornout when there are multiple workloads executing on disk i. Each brick represents an epoch, which is bounded by its workloads' allocation times. The volume of all these bricks gives the total write wornout count (w_i) for disk i. To calculate w_i, we further convert above-mentioned λ_{L_i} and S_i to the total logical data write rate function and the sequential ratio function during each epoch $[t_{ix}, t_{i(x+1)})$, respectively: $\lambda_{L_i}(t_{ix}, t_{i(x+1)}) = \sum_{j \in \mathbb{J}_i(t_{ix}, t_{i(x+1)})} \lambda_{L_{ij}}$, and $S_i(t_{ix}, t_{i(x+1)}) = \frac{\sum_{j \in \mathbb{J}_i(t_{ix}, t_{i(x+1)})} \lambda_{L_{ij}} S_{ij}}{\sum_{j \in \mathbb{J}_i(t_{ix}, t_{i(x+1)})} \lambda_{L_{ij}}}$, where x is the number of workloads executing on disk i, and t_{ix} is the arrival time of disk i's x^{th} workload. $\mathbb{J}_i(t_{ix}, t_{i(x+1)})$ is the set of workloads running on disk i during t_{ix} and $t_{i(x+1)}$ epoch. $\lambda_{L_{ij}}$ and S_{ij} are the write rate and sequential ratio of j^{th} workload in $\mathbb{J}_i(t_{ix}, t_{i(x+1)})$. Therefore, wornout w_i can be calculated as: $w_i = \sum_{t_{ix} \in \mathbb{T}_i} [\lambda_{L_i}(t_{ix}, t_{i(x+1)}) \cdot f_{seq}(S_i(t_{ix}, t_{i(x+1)})) \cdot (t_{i(x+1)} - t_{ix})]$. Here, t_{ix} is the starting time of disk i's x^{th} epoch. At the sample moment $t_{i(x+1)}$, we assume there are x workloads running on disk i. $\mathbb{T}_i$ is the set of arrival times of each workload running on disk i. The three parts (λ, WAF and time) match the three axes from Fig. 4, where each brick stands for each epoch, and the total volume of these bricks is the accumulated write count value of that SSD disk. Therefore, the data-avg TCO rate TCO' in Eq. 2 can be calibrated as:

$$TCO' = \frac{\sum_{i=1}^{N_D} [C_{I_i} + C'_{M_i}(T_{W_i} + \frac{W_i - w_i}{\lambda_i \cdot A_i})]}{\sum_{j=1}^{N_W} \lambda_j (T_{Lf_{D(j)}} - T_{I_j})}. \quad (3)$$

4 Algorithm Design

Based on the proposed TCO model, we further design a set of online allocation algorithms, called "MINTCO", which adaptively allocate new workloads

Algorithm 1: minTCO

```
1  Procedure minTCO()
2      for incoming new workload J_N do
3          for i ← 1 to N_D do
4              TCO_List[i] = TCO_Assign(i, J_N);
5          SelectedDisk = TCO_List.minValueIndex();
6          Disk[SelectedDisk].addJob(J_N);
7      return;
8  Procedure TCO_Assign(i,J_N)
9      for k ← 1 to N_D do
10         C_I = getCostInit(k);
11         C_M = getCostMaint(k);
12         if k == i then
13             T_{W_k} = T_{J_N} − T_{I_k};
14             T_{E_k} = getExpFutureWorkTime(k, J_N);
15             Data = getTotalLogWriteAmt(T_{W_k} + T_{E_k}) + (T_{W_k} + T_{E_k} − T_{J_N}) * λ_{J_N};
16         else
17             T_{W_k} = T_{R_k} − T_{I_k};
18             T_{E_k} = getExpFutureWorkTime();
19             Data = getTotalLogWriteAmt(T_{W_k} + T_{E_k});
20         TCO+ = C_I + C_M * (T_{W_k} + T_{E_k});
21         TotalData+ = Data;
22     return TCO/TotalData;
```

to SSDs in the disk pool. The main goal is to minimize the data-avg TCO rate (TCO') of the storage pool and also to conduct disk-level wear leveling during workload deployment and allocation.

4.1 Baseline minTCO

The main functionality of the baseline version of MINTCO is presented in Alg. 1. When a new workload arrives, MINTCO calculates the data-avg TCO rate for the entire disk pool, and then allocates the workload to the SSD that makes the lowest data-avg TCO rate of the entire disk pool. Specifically, there are two cases during the calculation of expected lifetime and total logical write amount. The first case is that when a new workload is assigned to disk k, we use this new workload's arrival time as the boundary between T_{W_k} and T_{E_k} phases, as shown in Alg. 1 lines 13 to 15 and Fig. 3(c). The second case is that when the new workload is not assigned to disk k (Alg. 1 lines 17 to 19), we use T_{R_k} (the arrival time of the most recent workload on disk k) as the boundary between T_{W_k} and T_{E_k} phases, as shown in Fig. 3(b). As discussed previously, our TCO model is compatible with any WAF models. Here we adopt the WAF model described in Eq. 3 to implement the functions in Alg. 1 line 14, 15, 18 and 19. The baseline MINTCO also needs to consider other resource constraints. For example, MINTCO needs to further check if the available spatial (in GB) and throughput (in IOPS) capacities of the chosen SSD are large enough to hold the new workload's working set. If not, MINTCO moves to the next SSD which has the second lowest data-avg TCO rate. If no disks have enough capacity, then the workload will be rejected.

4.2 Performance Enhanced minTCO

One limitation of the baseline MINTCO is that it does not balance the load across the SSDs in the pool and thus cannot achieve optimal resources utilization. However, best using of resources (e.g., I/O throughput and disk capacity) is an important goal in real storage system management. To address this limitation, we further develop the performance enhanced manager, namely MINTCO-PERF, which considers statistical metrics (i.e., load balancing and resource utilization) as the performance factors in workload allocation.

4.2.1 *System Resource Utilization*

We consider two types of resources, throughput (IOPS) and space capacity (GB), and calculate the utilization ($U(i,k)$) of disk i when disk k is selected to serve the new workload J_N, as:

$$U(i,k) = \begin{cases} \frac{R_U(i)}{R(i)}, & i \neq k \\ \frac{R_U(i)+R(J_N)}{R_i}, & i = k \end{cases}, \tag{4}$$

where $R_U(i)$, $R(i)$ and $R(J_N)$ represent the amount of *used* resource on disk k, the *total* amount of resource of disk i, and the resource *requirement* of workload J_N, respectively. When i is equal to k, we have the new requirement (i.e., $R(J_N)$) as extra resources needed on that disk. This equation can be used to calculate either throughput utilization (i.e., $U_p(i,k)$) or space capacity utilization (i.e. $U_s(i,k)$). The average utilization can be calculated to represent the system utilization of the entire disk pool: $\overline{U(k)} = \frac{\sum_{k=1}^{N_D} U(i,k)}{N_D}$. Our goal is to increase either average throughput utilization (i.e., $U_p(i,k)$) or average space utilization (i.e. $U_s(i,k)$).

4.2.2 *Load Balancing*

We use coefficient of variation (CV) of throughput (or space) utilizations among all disks to ensure the load balancing. Specifically, when assigning the workload J_N to disk k, we calculate expected $CV(k)$ as: $CV(k) = \frac{\sqrt{\frac{\sum_{i=1}^{N_D}[U(i,k)-\overline{U(k)}]^2}{N_D}}}{\overline{U(k)}}$. A smaller $CV(k)$ indicates better load balancing in the datacenter.

Then, we have our MINTCO-PERF algorithm which aims to minimize the data-avg TCO rate, while achieving best resource utilization and load balancing among disks. MINTCO-PERF uses an optimization framework to minimize the objective function under constrains listed in Eq. 5.

Minimize:

$$\begin{aligned} & f(R_w) \cdot TCO'(k) \\ & - g_s(R_r) \cdot \overline{U_s(k)} + h_s(R_r) \cdot CV_s(k) \\ & - g_p(R_r) \cdot \overline{U_p(k)} + h_p(R_r) \cdot CV_p(k) \end{aligned}$$

Subject to:

$$\begin{aligned} & i \in D, k \in D \\ & 0 \leq TCO'(i,k) \leq Th_c \\ & 0 \leq U_s(i,k) \leq Th_s \\ & 0 \leq U_p(i,k) \leq Th_p \end{aligned} \tag{5}$$

Upon the arrival of a new workload J_N, we calculate the "enhanced cost" of the the disk pool. The object function in Eq. 5 contains the TCO rate cost ($f(R_w) \cdot TCO'(k)$), the resource utilization reward ($g_s(R_r) \cdot \overline{U_s(k)}$ and $g_p(R_r) \cdot \overline{U_p(k)}$), and the load unbalancing penalty ($h_s(R_r) \cdot CV_s(k)$ and $h_p(R_r) \cdot CV_p(k)$). Notice that $TCO'(k)$ and $TCO'(i,k)$ represent the TCO rate of the entire disk pool and the TCO rate of disk i, respectively, when disk k is selected to take the workload. Non-negative parameters in Eq 5 (e.g., $f(R_w)$, $g_s(R_r)$, $g_p(R_r)$, $h_s(R_r)$ and $h_p(R_r)$) are the weight functions that are related with the read ratio ($R_r = \frac{readIO\#}{totalIO\#}$) and write ratio ($R_w = \frac{writeIO\#}{totalIO\#}$) of workloads. Finally, the disk with the lowest enhanced cost will be selected for the new workload. The reason behind this is that in the real world, write intensive workloads affect WAF and TCO, and read intensive workloads are sensitive to load balancing degree. In addition, Th_c, Th_s and Th_p are used as the upper bounds for TCO, space and throughput resources utilization ratios, respectively.

4.3 RAID Mode minTCO

In the era of flash drivers, there are also lots of commercial available solutions using RAID mode flash disk arrays in data center to further accelerate the I/O speeds, such as Diff-RAID [11] and RamSan-500 [12]. RAID technique transforms a number of independent disks into a larger and more reliable logical single entity [13]. Motivated by this state-of-the-art trend, we extend MINTCO-PERF to support storage systems with RAID mode flash disk arrays. The major problem during this transition is that it is almost *impossible* to calculate (or even monitor) the exact WAF of RAID flash disk arrays during runtime in real implementation. Moreover, for systems with different RAID setups, getting accurate WAF value will be more complicated. Therefore, we present an approximate approach that aims to *estimate* WAF and TCO to guide the data center manager to make allocation decisions from a long-term datacenter operation of view.

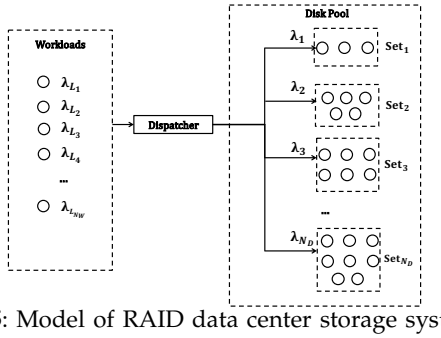

Fig. 5: Model of RAID data center storage system.

As shown in Fig. 5, the main idea of our approximation is to treat each RAID disk array set as a single "pseudo disk". A "pseudo disk" can consist of multiple SSDs, which are disks in the same RAID array. Similar to Fig. 2, the "pseudo disk" is the minimum unit to be allowed to accept workloads. To further simplify the problem, we focus on the cases that disks within each RAID disk set are homogeneous, and different sets can have heterogeneous RAID modes. Since other parts in the non-RAID model (i.e., Fig. 2) have not been modified in the our RAID model, MINTCO-RAID can use the same method to solve the RAID scenario problem by converting TCO and performance metrics of "pseudo disks". Specifically, Table 1 lists the corresponding adjustment parameters for the conversion.

Specifically, some of the factors can be ported from the realm of one disk to a "pseudo disk" relatively easily. For example, the costs (C_I,C'_M) and total write cycle limits (W) of a disk set are the sum of those values for each individual disk. An N-disk RAID-1 set has CapEx $C_{RAID_I} = N \cdot C_I$, OpEx $C'_{RAID_M} = N \cdot C'_M$, and total write cycle limits $W_{RAID} = N \cdot W$.

However, the estimation of a WAF function for a disk-set is not simple, since WAF is heavily dependent on the actual implementation [14]. From this point on, we focus on a simple implementation which is abstracted for a long-term and large-scale view of the datacenter. We observed that the striping parts of RAID-0 and RAID-5 are following the *same I/O locality and behavior* as the non-striping case, and thus the subset of workload on each disk keeps the same sequential ratio as the non-striped workload [15]. For example, after striping a 100% sequential stream with <0∼80> pages into four disks (with 10-page striping granularity), the four disks will be assigned pages <0∼10,41∼50>, <11∼20,51∼60>, <21∼30,61∼70> and <31∼40,71∼80>, respectively. Importantly, those striped subsets of a sequential stream will be physically written continuously on the disk (e.g., <41∼50> will be physically continuous to <0∼10> on disk 1), therefore the sequentialities will be kept, and thus WAF functions of those striping disks will be the same as that of a single disk [16]. For the parity disk of RAID-5, we observed that its I/O behavior follows the same locality as the original workload. Therefore, WAFs of RAID-0 and RAID-5 are still the same as a single disk [17]–[19]. It is worth mentioning that for an RAID-1 dis sets with 2 or more than 2 disks (we only allow *even* number of disks), this RAID-1 set will mirror each I/O to two "equal" RAID-0 groups, instead of replicating one single disk to all others which is a huge waste. Thus, the WAF function of each (homogeneous) disk set under RAID-1 mode remains the same as that of each individual disk.

Meanwhile, λ_L (logical data write rate) and S (spatial capacity) of each disk set also vary due to different RAID modes that might be used across these sets. Specifically, RAID-0 stripes the write and does not trigger any additional logical writes to the disk set. So, the logical data write rate of the disk set is the

RAID	Mode	TCO					Perf	
		C_I	C'_M	W	A	λ_L	S	ρ
0	Strip	N	N	N	1	1	N	1
1	Mirror	N	N	N	1	2	$N/2$	2
5	Pair	N	N	N	1	$\frac{N}{N-1}$	$N-1$	4

TABLE 1: Conversion table for different RAID modes.

same as that of non-RAID mode single disk, and the S should be multiplied by N. RAID-1 mirrors the write (with *two* copies), so the logical write rate is doubled compared with the workload's logical write rate. RAID-5 mode is relatively complex. For each logical write, $N-1$ disks are assigned for striping write, and 1 disk is left for parity write. Therefore, the overall logical data write rate of the disk set should be scaled by the workload's logical data write rate multiplying $\frac{N}{N-1}$.

Lastly, RAID-1 and RAID-5 introduce a write penalty (ρ) [20]. RAID-1 has two IOPS per write operation, while one RAID-5 write requires four IOPS per write operation [21]. We need to convert an incoming workload's original throughput requirement P_J to its RAID version P_{RAID_J}:

$$P_{RAID_J}(i) = P_J(i) \cdot R_W(i) \cdot \rho(i) + P_J(i) \cdot R_R(i), \quad (6)$$

where $\rho(i)$ is the write penalty for disk i, which can be obtained from Tab. 1. For example, a disk set with RAID-1 mode has 4 disks, each of them has $6,000$ IOPS, then the disk set's throughput capacity is $P_{RAID}(i) = 6,000 \times 4 = 24,000$ IOPS. For a new incoming workload which requires 30 IOPS, with write ratio of 40%, we can obtain its real requirement on RAID-1 mode by as: $P_{RAID_J} = 30 \times 40\% \times 2 + 30 \times (1 - 40\%) = 42$ IOPS. Similarly, if this new coming workload has a logical data write rate of 200 GB per day, then the equivalent logical data write rate in this RAID-1 disk set is doubled ($200 \times 2 = 400$ GB per day) since it replicates each write I/O. Furthermore, the actual physical write rate can be estimated based on equivalent logical data write rate and the corresponding write amplification function of the disk.

4.4 Offline Mode MINTCO

MINTCO, MINTCO-PERF and MINTCO-RAID are working in the so-called *online* scenario where the number and specs of disks in the disk pool are known and fixed, while the system has no foreknowledge about those I/O workloads that are keeping coming and waiting to be assigned to the disk pool. To achieve a lower TCO rate goal, these three algorithms are following the greedy-based approach to obtain the global best solution by choosing the "local" best solution. In real world, there exists another scenario called "*offline*" scenario, where the datacenter manager needs to allocate *all* known workloads to an undecided disk pool in the beginning. In detail, the manager needs to decide the number and type of disks to run the I/O workloads, and to allocate each workload to disks aiming to get a global best data-avg TCO rate solution. Unfortunately, we cannot directly use these three online-greedy algorithms to solve this offline problem. Although it is on theory possible to convert the offline problem into an NP-hard backpack problem with relatively complicated weight and value functions, to solve this problem in field is yet computationally time-consuming. For instance, due to the core feature of modern datacenters such as elasticity and scalability, both the numbers of workloads and disks are too "huge" for existing solutions like *branch and bound method* [22] or *multiple-objective genetic local search* [23].

Motivated by this challenge, in this section, we develop a less-costly allocation algorithm MINTCO-OFFLINE, which works in datacenters consisting of *homogeneous* disks. This algorithm starts from rethinking the problem from the relationship between sequential ratio, logical write rate and TCO rate. In Appendix 2, we prove that sorting all workloads by their sequential ratio and sending them to each disk in the order of sequential ratio is the best solution under ideal conditions. However, in real implementation, to avoid the case that purely based on sequential ratio may lead to capacity or I/O throughput unbalance, we need to manually set up two (or more than two) disk zones and then balance the write rate inside each zone.

As shown in Alg. 2 line 3 to 8, MINTCO-OFFLINE first groups the all workloads ($\mathbb{J}$) into high and low sequential ratio groups ($\mathbb{J}_H$ and $\mathbb{J}_L$) by comparing each workload's sequential ratio with a preset threshold ε. In fact, MINTCO-OFFLINE can be extended to support more than two workload groups and disk zones simply by using a more fine-grained sequential ratio threshold vector $\overrightarrow{\varepsilon}$ to replace the single value threshold ε. Meanwhile, MINTCO-OFFLINE also calculates the total write rate of each group. If write rates of these two zones are close to each other (by comparing their normalized difference with the approach switching threshold δ in line 9), the algorithm chooses the grouping approach to select a disk among *all* disks in the array to allocate the workload. Otherwise, the algorithm selects the greedy approach to allocate the workload in a certain disk group (e.g., either the high or low sequential ratio disk zone in our implementation). Lines 10 and 11 show the greedy approach, which simply calls the distribute function. Specifically, the distribute function first checks whether this workload's spatial and throughput capacity requirements exceeds a brand new (empty) disk. If yes, then this workload will be rejected, see in line 21 to 22, where C and P (resp. C_J and P_J) are total spatial and I/O throughput capacity of an empty disk (resp. of the current workload), respectively. Otherwise, it iterates to calculate the CV of write rates among the disk pool if this workload is added to each disk (line 26 to 30, where $remC_d$ and $remP_d$ are remaining spatial and throughput capacity of a certain disk d). It attempts

Algorithm 2: OfflineDeploy

```
1  Procedure minTCO-Offline(𝕁,𝔻)
2      for each workload J ∈ 𝕁 do
3          if S_J ≥ ε then
4              𝕁_H.add(J);
5              λ_H += λ_J;
6          else
7              𝕁_L.add(J);
8              λ_L += λ_J;
9      if |(λ_H − λ_L)/(λ_H + λ_L)| ≥ δ then
10         /* Greedy Approach */
11         Distribute(𝕁,𝔻);
12     else
13         /* Grouping Approach */
14         𝕁_H = 𝕁_H.descendingSortBySeqRatio();
15         𝕁_L = 𝕁_L.descendingSortBySeqRatio();
16         Distribute(𝕁_H,𝔻_H);
17         Distribute(𝕁_L,𝔻_L);
18     return;
19 Procedure Distribute(𝕁,𝔻)
20     for each workload J ∈ 𝕁 do
21         if C < C_J or P < P_J then
22             return Error: "Even empty disk cannot run
                 this workload.";
23         else if size(𝔻) == 0 then
24             𝔻.addNewDisk().addJob(J);
25         else
26             for each disk d ∈ 𝔻 do
27                 if remC_d < C_J or remP_d < P_J then
28                     CV[d] = ∞;
29                 else
30                     updateWriteRateCV(CV, d, λ_J);
31             diskID=minArg(CV);
32             if CV[diskID] = ∞ then
33                 /* No d ∈ 𝔻 can run this workload.
                     */
34                 𝔻.addNewDisk().addJob(J);
35             else
36                 𝔻[diskID].addJob(J);
37     return;
```

to balance the write rate of each disk in the disk zone. It finally selects the disk that will result in a minimum CV of write rates (line 31). Similarly, for the grouping approach, MINTCO-OFFLINE sorts each workload group by sequential ratio in the deceasing order (line 14 and 15), and then calls the distribute function to allocate these two groups of workloads to their corresponding zones, i.e., $\mathbb{D}_H$ and $\mathbb{D}_L$ as high and low sequential ratio disk zones, respectively (line 16 and 17).

5 EVALUATION

5.1 Write Amplification Measurement & Modeling

5.1.1 *Hardware Testbed*

Most SSD vendors do not provide APIs or performance counters to measure this physical write quantity. Hence, many prior studies (e.g., [6], [7]) have tried to develop models for *estimating* the WAF of an SSD based on a certain criterion. In this paper, we propose to leverage the data *directly measured* from SSDs to calculate a WAF function for an SSD. Our goal is to characterize the effects of write traffic from multiple workloads on the WAF, and see if such a characterization can be generalized as a mathematical model. Tab. 2 and Fig. 6(a) shows the testbed specification and the WAF measurement workflow, receptively.

Component	Specs
Processor	Xeon E5-2690, 2.9GHz
Processor Cores	Dual Socket-8 Cores
Memory Capacity	64 GB ECC DDR3 R-DIMMs
Memory Bandwidth	$102.4GB/s$
RAID Controller	LSI SAS 2008
Network	10 Gigabit Ethernet NIC
Operating system	Ubuntu 12.04.5
Linux Kernel	3.14 Mainline
FIO Version	2.1.10 run with direct I/O
Storage Type	NVMe SSD (Released in 2014)
Storage Capacity	1.6 TB

TABLE 2: Server node configuration.

5.1.2 *Filesystem*

We test two representative scenarios: "formatting with no file system" and "formatting with Ext4 file system". (1) "No file system" mimics the use case like a swap partition, where avoiding a filesystem mainly has three advantages: making more of the disk usable, since a filesystem always has some bookkeeping overhead; making the disks more easily compatible with other systems (e.g., the *tar* file format is the same on all systems, while filesystems are different on most systems.); and enabling enterprise user to deploy customized block manager running on the hypervisor without mounting a traditional filesystem. (2) "Ext4 file system" is a very solid file system which has been widely used for SSDs in datacenters using linux distributions for the past few years. The journaling that comes with Ext4 is a very important feature for system crash recovery although it causes some acceptable write activity.

5.1.3 *Precondition*

In order to ensure the SSD in the same state and stimulate the drive to the same performance state at the beginning of each measurement, we also conduct a 9 hour "preconditioning process". Tab. 3 shows the detail of our preconditioning setups. In detail, we have the following operations: "Sequential precondition" is that, between each measurement, the SSD is completely fulfilled with sequentially I/Os so that all write I/Os in the measurement workloads are overwrite operations, and WAF results will not be independent on garbage collection. "Random precondition" will further conduct an additional complete overwrite to the device with random I/Os with 4KB granularity after the sequential preconditioning

TABLE 3: I/O Setups of Preconditioning.

	Precon. Seq Fill	Precon. Rand Fill
RW	Write	Write
IODepth	16	32
BlockSize	1MB	4KB
Job Number	1	4

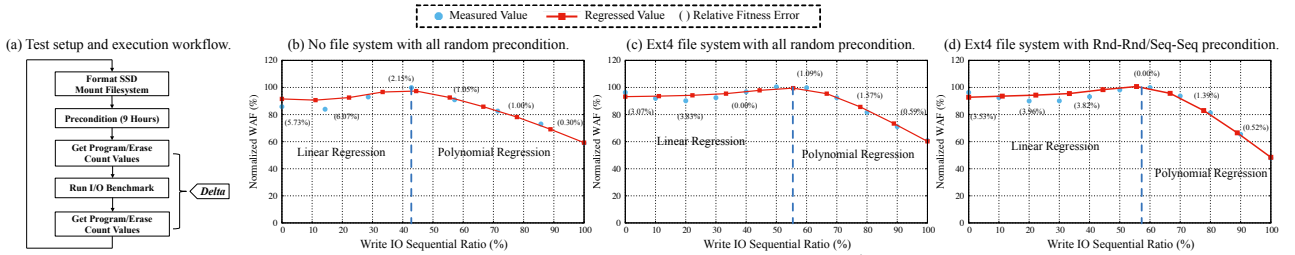

Fig. 6: WAF measurement experiment results.

process to randomize the workset distribution. "Rnd-Rnd/Seq-Seq precondition" is the policy that we use the random and sequential precondition for non-100% sequential and 100% sequential I/O workloads, respectively. We attempt to use these workloads to observe the ideal write performance (i.e., steady write performance). These two precondition operations can help us simulate different scenarios.

5.1.4 *I/O Workloads*

TABLE 4: Statistics for *part* of I/O workloads we use.

Trace Name	S (%)	λ (GB/day)	P_{Pk} $(IOPS)$	R_W (%)	WSs (GB)
mds0	31.52	21.04	207.02	88.11	6.43
prn0	39.13	131.33	254.55	89.21	32.74
proj3	72.06	7.50	345.52	5.18	14.35
stg0	35.92	43.11	187.01	84.81	13.21
usr0	28.06	37.36	138.28	59.58	7.49
usr2	46.10	75.63	584.50	18.87	763.12
wdv0	30.78	20.42	55.84	79.92	3.18
web0	34.56	33.35	249.67	70.12	14.91
hm1	25.15	139.40	298.33	90.45	20.16
hm2	10.20	73.12	77.52	98.53	2.28
hm3	10.21	86.28	76.11	99.86	1.74
onl2	74.41	15.01	292.69	64.25	3.44
Fin1	35.92	575.94	218.59	76.84	1.08
Fin2	24.13	76.60	159.94	17.65	1.11
Web1	7.46	0.95	355.38	0.02	18.37
Web3	69.70	0.18	245.09	0.03	19.21

In order to study the effects of sequential traffic on WAF, we conduct an experiment that can control the amount of sequential traffic being sent to an SSD. Most workloads in real systems are a certain mixture of sequential and random I/Os. To mimic such real situations, we generate mixed workloads by using an I/O testing tool FIO [24]. We also make changes to an 1.6TB NVMe SSD firmware to parse the value of (page) program and (block) erase counters. We investigate the write amplification factor as the ratio of sequential and random accesses, and the changes of these counters, as shown as "delta" in Fig. 6(a).

5.1.5 *WAF Results and Modeling*

Fig. 6(b)-(d) show three representative cases from our WAF experimental results, which present the normalized WAF as a function of different sequential ratios on write I/Os. The WAF data points are normalized by the largest WAF across different workload sequential ratios (e.g., the WAF under 40.22% sequential ratio in Fig. 6 (b)). Thus, the original WAF is $\in [1, +\infty)$, while the normalized WAF is $\in [0, 1]$.

First, we can see that WAF curves in the three figures are similar, i.e., the curves can be regressed into two stages: a flat linear regression stage and a dramatically decreasing polynomial regression stage. The former part shows that the write amplification factor of mixed workloads is almost identical to that of a pure random workload and keeps almost constant before a turning point (around 40% to 60%). But, after this turning point, the WAF dramatically decreases. In other word, a small fraction of random accesses is necessary to intensify the write amplification factor.

We further regress the WAF (A) as a piecewise function of sequentiality of I/O operations in the workload as shown in Eq. 7, where α, β, γ, μ and ε are parameters, and S is the sequential ratio.

$$A = f_{seq}(S) = \begin{cases} \alpha S + \beta, & S \in [0, \varepsilon] \\ \eta S^2 + \mu S + \gamma, & S \in (\varepsilon, 1] \end{cases} \quad (7)$$

At the turning point $S = \varepsilon$, we have $\alpha\varepsilon + \beta = \eta\varepsilon^2 + \mu\varepsilon + \gamma$. Additionally, α is close to zero since the linear regression stage is relatively smooth. We carry out these experiments multiple times on a number of NVMe SSDs, and draw our conclusions as follows. We believe that such a mathematical model of sequential write traffic vs. WAF can be constructed for most devices, and each SSD can have its own unique version of WAF function, depending on a number of its hardware-related factors (FTL, wear leveling, over-provisioning etc.). However, being able to regress a mathematical model for the problem forms the basis of the rest of this paper. Additionally, we also observe that the regression turning point of the non-filesystem case (Fig. 6(b)) comes earlier than Ext4's (Fig. 6(c) and (d)). This validates the fact that Ext4's (bookkeeping) overhead is heavier than the raw disk. Moreover, when the sequential ratio is 100%, the WAF under "Rnd-Rnd/Seq-Seq precondition" case (Fig. 6(d)) is lower than that under the "All-Rnd precondition" case (Fig. 6(c)). This validates that in the former case, the steady write performance can be reached.

5.2 TCO Evaluation

5.2.1 *Benchmarks and Metrics*

In this section, we plug the WAF model regressed from the real experiments to our TCO model, and then evaluate our MINTCO algorithms. Our trace-driven

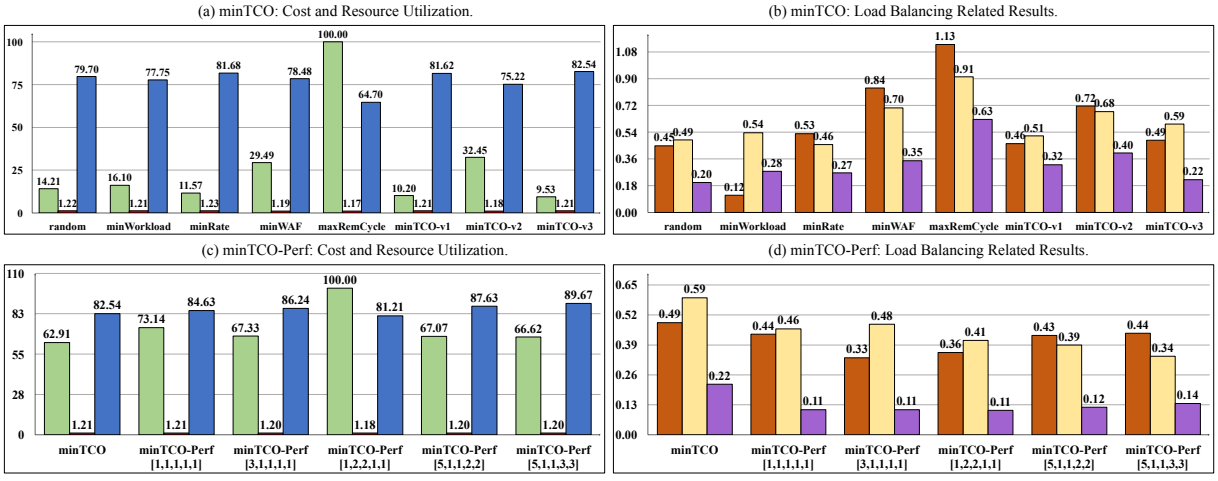

Fig. 7: TCO rate, resource utilization, and load balancing results under MINTCO and MINTCO-PERF.

simulation experiments are conducted based on the spec of real NVMe disks and enterprise I/O workloads. Specifically, we evaluate more than one hundred enterprise workloads from MSR-Cambridge [25], FIU [26] and UMass [27] trace repositories. These workloads represent applications widely used in real cloud storage systems, such as financial applications, web mail servers, search engines, etc.

Tab. 4 shows the statistics for some of these workloads (out of more than 100 workloads that we are using), where S is the sequential ratio of write I/O (i.e., the ratio between the amount of sequential write I/Os and the amount of total write I/Os), λ is the daily logical write rate (GB/day), P_{Pk} is the peak throughput demand with the $5min$ statistical analyses window, R_W is the write I/O ratio (i.e., the ratio between the amount of write I/Os and the amount of total I/Os), and WSs is the workings set size (i.e., the spatial capacity demand). The arrival process of these workloads is drawn from an exponential distribution. We use the following metrics to evaluate our MINTCO algorithms: (1) cost per GB during the expected lifetime: the total logical data-averaged TCO during the expected lifetime ($TCO'_{LfPerData}$); (2) resource utilization: the average throughput and space capacity utilization ratios among all disks; and (3) load balancing: the CV of resource utilization ratio across all disks.

5.2.2 TCO Experimental Results

We implement both baseline MINTCO and the performance enhanced MINTCO-PERF. Additionally, three versions of MINTCO are considered, such that MINTCO-$v1$ uses the TCO of expected lifetime ($\sum_{i=1}^{N_D}(C_{I_i}+C'_{M_i}\cdot T_{Lf_i})$), MINTCO-$v2$ uses the TCO model of expected lifetime per day ($\frac{\sum_{i=1}^{N_D}(C_{I_i}+C'_{M_i}\cdot T_{Lf_i})}{\sum_{i=1}^{N_D} T_{Lf_i}}$), and MINTCO-$v3$ uses the TCO model of expected lifetime per GB amount ($\frac{\sum_{i=1}^{N_D}(C_{I_i}+C'_{M_i}\cdot T_{Lf_i})}{\sum_{j=1}^{N_W} D_j}$). As expected, none of these baseline MINTCO algorithms consider load balancing and resource utilization during allocation. For comparison, we also implement other widely used allocation algorithms, including $maxRemCycle$ which selects the disk with the greatest number of remaining write cycles, $minWAF$ which chooses the disk with the lowest estimated WAF value *after* adding the newly incoming workload, $minRate$ which chooses the disk with the smallest sum of all its workloads' logical write rates, and $minWorkloadNum$ which selects the disk with the smallest number of workloads.

(1) minTCO

We first conduct the experiments running on the disk pool which consists of 20 disks, with 9 different models of NVMe SSDs (available on market in fall 2015). In our implementation, we mix about 100 workloads from MSR, FIU, and Umass with exponentially distributed arrival times in 525 days. Fig. 7(a) and (c) show the results of data-avg TCO rates and resource (I/O throughput and space capacity) utilizations under different allocation algorithms. Fig. 7(b) and (d) further present the performance of load balancing, e.g., CVs of workload number and resource utilizations. First, as shown in Fig. 7(a) and (c), MINTCO-v3 achieves the lowest data-avg TCO rate($/GB). We also observe that among the MINTCO family, MINTCO-$v2$ performs the worst, see Fig. 7(a), and obtains the largest CVs of allocated workload numbers. The reason is that, to some extent, MINTCO-$v2$ aims at maximizing the expected life time by sending almost all workloads to a single disk, in order to avoid "damaging" disks and increasing the TCO.Therefore, it cannot "evenly" allocate the workloads. We further find that $maxRemCycle$ performs the worst among all allocation algorithms, because it does not consider the TCO as well as the varying WAF due to different sequentialities of the running workloads. In summary, $minTCO$-$v3$ is the best choice which

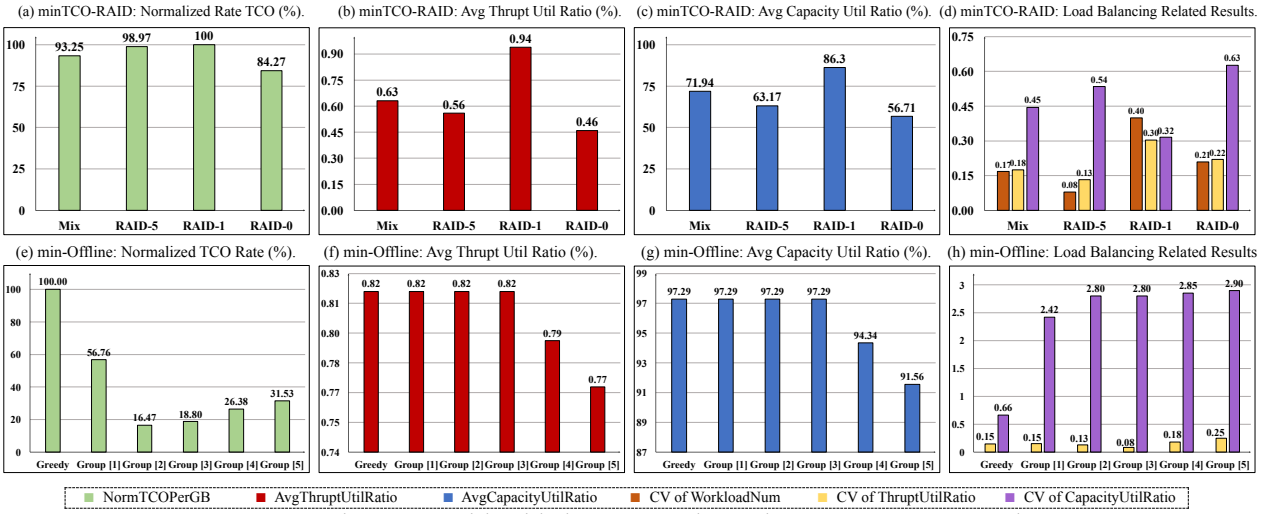

Fig. 8: TCO rate, resource utilization, and load balancing results under MINTCO-RAID and MINTCO-OFFLINE.

considers expected life time, cost and expected logical data amount that can be written to each disk.

(2) minTCO-Perf

We next implement MINTCO-PERF which is based on MINTCO-$v3$, and considers the data-avg TCO rate as the criterion to choose the disk for the new workload. As described in Sec. 4.2.2, MINTCO-PERF uses Eq. 5 to find the best candidate under the goal of minimizing TCO, maximizing resource utilization, and balancing the load. There are a set of weight functions (i.e., $f(R_w)$, $g_s(R_r)$, $g_p(R_r)$, $h_s(R_r)$ and $h_p(R_r)$) used in Eq. 5 . To investigate the effects of these weight functions, we conduct sensitivity analysis on the average values for the five weight functions in Eq. 5. After trying different approaches, and choose the linear function approach to implement weight functions. We show the over-time average value of each function normalized by the minimum function one. For example, "[5,1,1,2,2]" means that all values are normalized by the second weight function ($g_s(R_r)$). In Fig. 7(c) and (g), we observe that space capacity (instead of I/O throughput) is always the system bottleneck (i.e., with high utilization) across different approaches. This is because NVMe SSDs support up to $64K$ I/O queues and up to $64K$ commands per queue (i.e., an aggregation of $2K$ MSI-X interrupts). Meanwhile, workloads we are using here are collected from traditional enterprise servers, which have not been optimized for NVMe's revolutionary throughput capacity improvement. We also find that with a slight sacrifice in TCO, MINTCO-PERF can improve both resource utilization and load balancing. Fig. 7(c) further show that "[5,1,1,3,3]" is the best choice among all cases, which is 3.71% more expensive than the baseline $minTCO$, but increases the space utilization ratio by 7.13%, and reduces CV of throughput and space capacity utilization by 0.25 and 0.8, respectively. This is because MINTCO-PERF sets TCO and space capacity higher priorities.

(3) minTCO-RAID

Our third experiment is for different RAID modes. We group 6 same-model NVMe disks into a set (internal homogeneous), but allow various sets to have different models (external heterogeneous). In total, we have 8 sets with 48 disks. Fig. 8(a) to (d) compare results of RAID-0 striping, RAID-1 mirroring, RAID-5 pairing and mix of these three modes under MINTCO-RAID. RAID-1 has the highest TCO since it duplicates each I/O, followed by RAID-5 which has $1/N$ space used for replica. RAID-0 has zero replica so its TCO per data is the lowest. The results of the mix case is in between RAID-1 and RAID-5. No doubt that the average utilization ratios of both throughput and capacity of RAID-1 are the greatest among all, since it mirrors each I/O. Since we assign a larger weight for spatial capacity in the objective function than others, RAID-1, as the most spatial-capacity-sensitive RAID mode, obtains the most benefit.

(4) minTCO-Offline

The last experiment is to investigate the performance of MINTCO-OFFLINE in the offline scenario. We attempt to allocate 1359 MRS, FIU and Umass I/O workloads to a datacenter with homogeneous disks. Specifically, MINTCO-OFFLINE needs to decide how many disks we need and how to allocate workloads to the disk pool to minimized the data-avg TCO rate. In our implementation, we build both the greedy and grouping allocators to evaluate the performance as well as to tune the switching threshold δ. According to our rationales in Sec. 4.4, the best use case for the grouping approach is to sort and allocate workloads to disks by their sequential ratios while keeping write rates of each disk similar. In other words, the number of zones should be the same as the number of disks. But in real implementation, we need to consider the spatial capacity and I/O throughput constrains, therefore we tune different numbers of zones to investigate the performance. As shown in Fig. 8(e) to (h), we

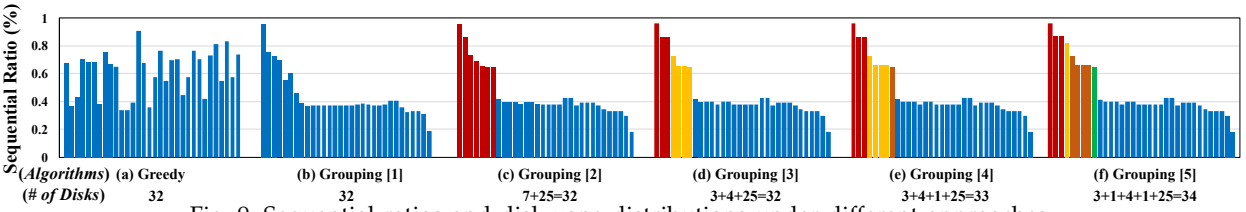

Fig. 9: Sequential ratios and disk-zone distributions under different approaches.

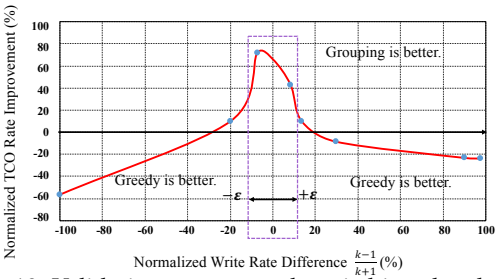

Fig. 10: Validation on approach switching threshold.

observe that 2-zone grouping approach has the lowest TCO rate; and the greedy and grouping approaches with 1 to 3 zones have almost the same throughput and capacity utilization ratios. In terms of load balancing, the 3-zone grouping case's I/O throughput is best balanced, and the greedy approach achieving the best balance in spatial capacity.

To further investigate the reason of performance difference, Fig. 9 illustrates the disk-zone distributions and the sequential ratios of each disk under different algorithms. The greedy approach's sequential ratio distribution is similar to a randomized curve, while, all these grouping approaches unsurprisingly have monotone decreasing curves. Additionally, the larger the number of disk zones is, the better the disk pool is sorted by sequential ratios. However, the trade-off of allocating more disk zones is that it triggers more "unnecessary" disks compared with less disk zones, when the sequential ratio distribution of the entire workload set is not equalikely matching the sequential ratio threshold vector $\vec{\varepsilon}$ (i.e., some disk zones have very rare workloads). For example, as shown in Fig. 9, if we divide workloads into more than three disk zones, we need 33 disks (4 zones, see Fig. 9(e)) and 34 disks (5 zones, see Fig. 9(f)), on the other hand, we only need 3 disks when we consider 3 or less zones (see Fig. 9(b) to (d)), or even the greedy approach (Fig. 9(a)). The reason is that as indicated in WAF measurement, NVMe SSD's WAF is almost identical and keeps almost constant before a turning point (around 40% to 60%), so there is no need to "fine-grainedly" divide the sequential ratio range from zero to the turning point. Therefore, based on these results, we choose the number of zones as 2 since it has the lowest data-avg TCO and relatively good resource utilization and load balance performance.

Additionally, we further conduct a set of sensitivity analysis on different logical write rates to decide the threshold of switching by using different I/O workloads with different overall sequential ratios. Fig. 10 shows the normalized TCO rate improvement degree (i.e., $\frac{TCO'(Greedy)-TCO'(Grouping)}{TCO'(Greedy)}$) vs. normalized write rate difference of two workload groups (i.e., $\frac{\lambda_L-\lambda_H}{\lambda_H+\lambda_L} = \frac{k-1}{k+1} \in [-100\%, 100\%]$, where $k \in [0, +\infty)$). Here we fix the sequential ratio threshold ϵ to 0.6. We observe that when k is less than 1.31 (i.e., $\delta = 13.46\%$), the grouping approach is better, which validates the "$k \rightarrow 1$" condition in Sec. 4.4. Therefore, we set MINTCO-OFFLINE's approach-switching threshold to $\delta = 13.46\%$.

6 Related Work

Few prior studies that have focused on the long-term costs of SSD-intensive storage systems with SSDs so far, especially in the context of datacenters. Majority of the existing literature investigates SSD-HDD tiering storage systems. [28] was proposed to reduce the cost of a SSD-HDD tiering storage system by increasing both temporal and spatial update granularities. The "cost" in vFRM is the I/O latency and delay, rather than price. [29] built a cost model that also considers the lifetime cost of ownership including energy and power costs, replacement cost, and more. They assume that the "trade-in" value of the disk is a linear function of its available write cycles. Our previous work [4] also reveals the relationship between WAF and write I/O sequential ratio. Meanwhile, in terms of budget-driven workload allocation method, [30] recently presents a systematic way to determine the optimal cache configuration given a fixed budget based on frequencies of read and write requests to individual data items. [31] discussed modeling NVMe workload for optimizing TCO. However, it only addressed on the online and per-I/O-request scheduling problem by minimizing the "cost" in terms of workloads' latencies, while our work focus on both online and offline allocation problem based on the dollar-per-GB TCO model in a long term view. Write amplification of an SSD depends on the FTL algorithm deployed in the controller. The studies in [32]–[35] presented FTL algorithms that are commonly adopted in public domain. However, SSD vendors do not publicly reveal the FTL algorithms to customers due to confidentiality issues.

7 Conclusion

In this paper, we characterize the write amplification (WA) of SSDs as a function of fraction of sequential

writes in a workload. We plug this WAF function into our proposed Total Cost of Ownership (TCO) model, which also considers capital and operational cost, estimated lifetime of flash under different workloads, resource restrictions and performance QoS. Based on the TCO model, we build the online workload allocation algorithm MINTCO and MINTCO-PERF. Experimental results show that MINTCO reduces the ownership cost, and MINTCO-PERF further balances the load among disks and maximize the overall resource utilization, while keeping the TCO as low as possible. MINTCO-RAID extends the above solution to RAID mode of enterprise storage environment. Last but not least, MINTCO-OFFLINE presents a less-costly solution for the offline allocation scenario to minimize the overall TCO.

REFERENCES

[1] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang *et al.*, "Bigdatabench: A big data benchmark suite from internet services," in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*. IEEE, 2014, pp. 488–499.

[2] "Amazon Web Services," http://aws.amazon.com/.

[3] "Google Computer Engine," http://cloud.google.com/compute/.

[4] Z. Yang, M. Awasthi, M. Ghosh, and N. Mi, "A Fresh Perspective on Total Cost of Ownership Models for Flash Storage in Datacenters," in *2016 IEEE 8th International Conference on Cloud Computing Technology and Science*. IEEE, 2016.

[5] Z. Yang, M. Ghosh, M. Awasthi, and V. Balakrishnan, "Online Flash Resource Allocation Manager Based on TCO Model," Patent US15/092 156, US20 170 046 089A1, 2016.

[6] P. Desnoyers, "Analytic modeling of ssd write performance," in *Proceedings of the 5th Annual International Systems and Storage Conference*. ACM, 2012, p. 12.

[7] W. Bux and I. Iliadis, "Performance of greedy garbage collection in flash-based solid-state drives," *Performance Evaluation*, vol. 67, no. 11, pp. 1172–1186, 2010.

[8] R. Zhou, M. Liu, and T. Li, "Characterizing the efficiency of data deduplication for big data storage management," in *Workload Characterization (IISWC), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 98–108.

[9] V. Tarasov, D. Hildebrand, G. Kuenning, and E. Zadok, "Virtual machine workloads: The case for new nas benchmarks," in *Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 13)*, 2013, pp. 307–320.

[10] "The mega-datacenter battle update: Internet giants increased capex in 2014," https://451research.com/report-short?entityId=84745.

[11] M. Balakrishnan, A. Kadav, V. Prabhakaran, and D. Malkhi, "Differential RAID: rethinking RAID for SSD reliability," *ACM Transactions on Storage (TOS)*, vol. 6, no. 2, p. 4, 2010.

[12] W. Hutsell, "An in-depth look at the ramsan-500 cached flash solid state disk," *Texas Memory Systems White Paper*, p. 16, 2008.

[13] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating server storage to ssds: analysis of tradeoffs," in *Proceedings of the 4th ACM European conference on Computer systems*. ACM, 2009, pp. 145–158.

[14] B. Mao, H. Jiang, S. Wu, L. Tian, D. Feng, J. Chen, and L. Zeng, "Hpda: A hybrid parity-based disk array for enhanced performance and reliability," *ACM Transactions on Storage (TOS)*, vol. 8, no. 1, p. 4, 2012.

[15] A. Busch, Q. Noorshams, S. Kounev, A. Koziolek, R. Reussner, and E. Amrehn, "Automated workload characterization for i/o performance analysis in virtualized environments," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ACM, 2015, pp. 265–276.

[16] R. Gunasekaran, S. Oral, J. Hill, R. Miller, F. Wang, and D. Leverman, "Comparative i/o workload characterization of two leadership class storage clusters," in *Proceedings of the 10th Parallel Data Storage Workshop*. ACM, 2015, pp. 31–36.

[17] "RAID Penalty," http://theithollow.com/2012/03/21/understanding-raid-penalty/.

[18] S. Lowe, "Calculate iops in a storage array," 2010, https://www.techrepublic.com/blog/the-enterprise-cloud/calculate-iops-in-a-storage-array.

[19] E. Shanks, "Understanding raid penalty," 2012, https://theithollow.com/2012/03/21/understanding-raid-penalty.

[20] S. Miller, "Understanding raid performance at various levels," https://blog.storagecraft.com/raid-performance.

[21] R. Nobel, "The write penalty of raid 5," 2011, http://rickardnobel.se/raid-5-write-penalty.

[22] W. Shih, "A branch and bound method for the multiconstraint zero-one knapsack problem," *Journal of the Operational Research Society*, vol. 30, no. 4, pp. 369–378, 1979.

[23] A. Jaszkiewicz, "On the performance of multiple-objective genetic local search on the 0/1 knapsack problem-a comparative experiment," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 402–412, 2002.

[24] "FIO: Flexible I/O Tester," http://linux.die.net/man/1/fio.

[25] "MSR Cambridge Traces," http://iotta.snia.org/traces/388.

[26] "SNIA Iotta Repository," http://iotta.snia.org/historical_section.

[27] "UMass Trace Repository," http://traces.cs.umass.edu/index.php/Storage/Storage.

[28] J. Tai, D. Liu, Z. Yang, X. Zhu, J. Lo, and N. Mi, "Improving Flash Resource Utilization at Minimal Management Cost in Virtualized Flash-based Storage Systems," *Cloud Computing, IEEE Transactions on*, no. 99, p. 1, 2015.

[29] Z. Li, A. Mukker, and E. Zadok, "On the importance of evaluating storage systems' costs," in *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage '14)*, 2014, pp. 1–5.

[30] S. Ghandeharizadeh, S. Irani, and J. Lam, "Memory hierarchy design for caching middleware in the age of nvm," 2015.

[31] B. Jun and D. Shin, "Workload-aware budget compensation scheduling for nvme solid state drives," in *Non-Volatile Memory System and Applications Symposium (NVMSA), 2015 IEEE*. IEEE, 2015, pp. 1–6.

[32] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 6, no. 3, p. 18, 2007.

[33] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compactflash systems," *Consumer Electronics, IEEE Transactions on*, vol. 48, no. 2, pp. 366–375, 2002.

[34] H. Cho, D. Shin, and Y. I. Eom, "Kast: K-associative sector translation for nand flash memory in real-time systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 507–512.

[35] S. Lee, D. Shin, Y.-J. . J. Kim, and J. Kim, "Last: locality-aware sector translation for nand flash memory-based storage systems," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 6, pp. 36–42, 2008.

Zhengyu Yang is a Senior Software Engineer at Samsung Semiconductor Inc. He received his Ph.D. degree at Northeastern University, Boston. He graduated from the Hong Kong University of Science and Technology with a M.Sc. degree in Telecommunications, and he obtained his B.Sc. degree in Communication Engineering from Tongji University, Shanghai, China. His research interests are cache algorithm, deduplication, cloud computing, datacenter storage and scheduling optimization.

Manu Awasthi is an Associate Professor at Ashoka University, India. During this work, he was a Senior Staff Engineer at Samsung Semiconductor Inc., San Jose, CA. He received his Ph.D. degree in Computer Science from University of Utah, UT in 2011. His research interests are performance evaluation, storage reference architectures, and characterization of datacenter applications.

Mrinmoy Ghosh is a Performance and Capacity Engineer at Facebook Inc., Menlo Park, CA. During this work, he was a Senior Staff Engineer at Samsung Semiconductor Inc., San Jose, CA. He received his Ph.D. degree in Computer Engineering from Georgia Institute of Technology, GA in 2008. His research interests are characterizing datacenter applications, analyzing storage stack performance, NVMe SSD performance, and remote storage performance.

Janki Bhimani is a Ph.D. candidate working with Prof. Ningfang Mi at Northeastern University, Boston. Her current research focuses on performance prediction and capacity planning for parallel computing heterogeneous platforms and backend storage. She received her M.S. from Northeastern University in Computer Engineering. She received her B.Tech. from Gitam University, India in Electrical and Electronics Engineering.

Ningfang Mi is an Associate Professor at Northeastern University, Boston. She received her Ph.D. degree in Computer Science from the College of William and Mary, VA. She received her M.S. in Computer Science from the University of Texas at Dallas, TX and her B.S. in Computer Science from Nanjing University, China. Her research interests are capacity planning, MapReduce scheduling, cloud computing, resource management, performance evaluation, workload characterization, simulation and virtualization.

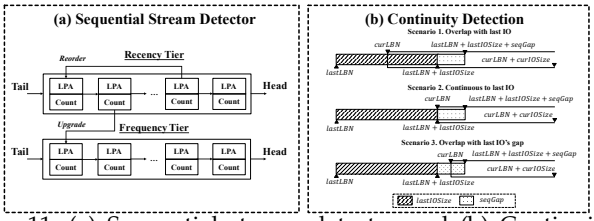

Fig. 11: (a) Sequential stream detector and (b) Continuity detection.

8 Appendix1: Sequential Ratio Estimator

We set the bin size as $1MB$ as the prefetching size between SSD and HDD, but the original I/O request size above SSD can be any size. Different size and address distribution will impact on sequential ratio, and thus the write amplification Therefore, we need to have a module to estimate sequential ratio on each disk. Our stream sequentiality detector considers two criteria during qualification:

[Criteria 1] Continuity: As shown in Fig. 11(a), for each incoming I/O, we check whether its "precursor neighbor" I/Os is recorded in the queue (we have 32 queues). If it exists, then we update that stream node by appending this new page's metadata to it. Otherwise, we try to create a new node in the queue for this I/O. The new stream I/O's node will be moved to the MRU position. The challenging part is to check whether the new I/O's precursor neighbor exists and whether the new I/O is sequential to it. Only the three scenarios shown in Fig. 11(b) are considered that the new I/O is (sequential) successor of a certain collected stream. Scenario 1 of Fig. 11(b) is that the current I/O's start address is within $[lastLBN, lastLBN + lastIOSize)$; Scenario 2 of Fig. 11(b) is a "perfect case" where current I/O starts exactly from last I/O's ending point. In practical, due to different write granularities along OS to device firmware I/O path, it is necessary to relax the above two scenarios' conditions a little bit. Thus we provide a reconsideration chance by extending the ending point of last I/O by an additional space which is named as $seqGap$ (preset as 32 4KB-pages, which is 128KB) as shown in Scenario 3 of Fig. 11(b). To sum up, for continuity, we are more interested in the current I/O's start address ($currLBN$) and the last I/O's coverage ($lastLBN + lastIOSize$).

[Criteria 2] Stream I/O Size: Not all streams in the queue are considered as sequential. To be qualified as a sequential stream, instead of using each single I/O's size, the collected I/O streams' total I/O working set size (deduplicated space coverage size) will be picked and compared with a preset threshold "$seqStreamSize$" (256 4KB-pages, which is $1MB$). Streams that are longer than this threshold will finally be regarded as sequential streams.

9 Appendix2: Proof of offline mode minTCO

In this section, we first formulate the problem by conducting mathematical analysis on different allocation approaches, and then prove the correctness of the design to switch between grouping and greedy approaches.

Problem Formulation

We focus on the offline scenario where all disks be *identical* and *homogeneous*, i.e., $\mathbb{D} = \{x_i | x_i = d, i \in [1, |\mathbb{D}|]\}$, where d represents a single disk, and $\mathbb{D}$ stands for the entire disk pool. At beginning we *have* the knowledge of all workloads to be deployed (i.e., the workload set $\mathbb{J}$). We denote the total sequential ratio and the total logical data write rate of $\mathbb{J}$ as $\lambda_{\mathbb{J}}$ and $S_{\mathbb{J}}$, respectively. Additionally, in real enterprise datacenters use case, the granularity of each workload is very tiny compared with that of the entire $\mathbb{J}$. Based on these assumptions and by using the technique of mathematical induction, we start our investigate on how different approaches work and which one is better under different circumstances.

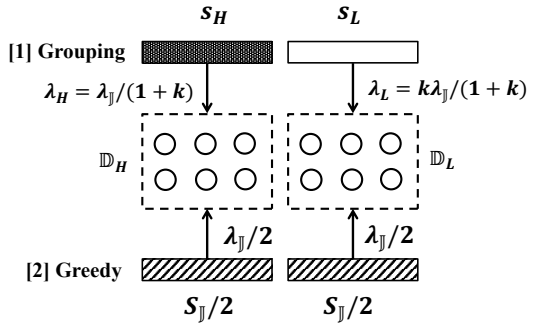

Fig. 12: Grouping and Greedy approaches.

Step 1: Base Case

minTCO-Offline is an algorithm switching between two approaches, namely "grouping" and "greedy" approaches. Fig. 12 illustrates the example of these two approaches. Assume there are two zones with same number of disks (i.e., $\mathbb{D}_L$ and $\mathbb{D}_H$), and similar to the RAID mode estimation, we hereby regard multiple disk as one "pseudo disk". Let C'_M and C_I be the maintenance and initial costs of *each zone*, W be the total cycle of *each zone*, and A be the WAF function of *each disk*.

(1) Grouping Approach: Workloads are sorted into two groups based on their sequential ratios (i.e., one group $\mathbb{J}_H$ with higher total sequential ratio S_H, and the other group $\mathbb{J}_L$ with lower total sequential ratio S_L). Let $\lambda_H = \lambda_{\mathbb{J}}/(1+k)$ and $\lambda_L = k\lambda_{\mathbb{J}}/(1+k)$ to be the total logical write rate of these two groups, respectively, where $\lambda_H + \lambda_L = \lambda_{\mathbb{J}}$. According to 3.3.4, we have:

$$S_{\mathbb{J}} = \frac{\lambda_H S_H + \lambda_L S_L}{\lambda_H + \lambda_L} = \frac{1}{k+1} S_H + \frac{k}{k+1} S_L. \tag{8}$$

It then sends these two groups of workloads to different disk zones (i.e., D_H and D_L), in order to reduce the cross-workload affect and the corresponding WAF values. Notice that since each I/O workload is so fine-grained, we can assume that there are no capacity and throughput constrain issues during this allocation.

(2) Greedy Approach: It focuses on greedily filling the workload into existing disks to best use disk resources, and balancing the logical write rate of each disk, without consideration of the sequential ratio and WAF. It is possible to set capacity or I/O throughput usages as the balance target, however according to both the following rationale and observations from our real implementation, logical write rate plays a more important role in terms of TCO rate. To be consistent with our analysis of the grouping approach, we also treat the greedy approach as it is sending two mixed workloads with same workload groups. Each workload group has same logical write rate $\lambda_M = \lambda_{\mathbb{J}}/2$, and the same sequential ratio as the entire workload set's $S_{\mathbb{J}}$.

(3) TCO Comparison: Based on the above analysis, we compare data-avg TCO rate of two approaches by using Eq. 3:

$$\begin{aligned} &Diff\_TCO' = TCO'(Greedy) - TCO'(Grouping) \\ &= \frac{2[C_I + C'_M \frac{W}{\lambda_M A(S_{\mathbb{J}})}]}{2[\lambda_M \frac{W}{\lambda_M A(S_{\mathbb{J}})}]} - \frac{C_I + C'_M \frac{W}{\lambda_H A(S_H)} + C_I + C'_M \frac{W}{\lambda_L A(S_L)}}{\lambda_H \frac{W}{\lambda_H A(S_H)} + \lambda_L \frac{W}{\lambda_L A(S_L)}} \\ &= \frac{2C_I}{W}[\frac{A(S_{\mathbb{J}})}{2} - \frac{1}{\frac{1}{A(S_H)} + \frac{1}{A(S_L)}}] + C'_M \frac{(k-1)[A(S_H) - kA(S_L)]}{\lambda k[A(S_H) + A(S_L)]}. \end{aligned} \tag{9}$$

We found that only when $k \to 1$ (i.e., for the grouping approach, its two workload groups have similar logical write rate $\lambda_H \approx \lambda_L \approx \frac{\lambda}{2}$), the second part of Eq. 9 will be zero. Thus we have:

$$lhs \approx \frac{2C_I}{W} \cdot [\frac{A(S_{\mathbb{J}})}{2} - \frac{1}{\frac{1}{A(S_H)} + \frac{1}{A(S_L)}}]. \tag{10}$$

Meanwhile, according to Eq. 8, "$k \to 1$" also causes $A(S_{\mathbb{J}}) = A(\frac{S_H + S_L}{2})$. Moreover, based on our measurement and regression results (will be discussed in Sec. 5.1.5), A is a concave function of sequential ratio and $A \geq 1$, which has the following feature:

$$A(S_{\mathbb{J}}) = A(\frac{S_H + S_L}{2}) \geq \frac{A(S_H) + A(S_L)}{2}. \tag{11}$$

An example of this feature is illustrated in Fig. 13.

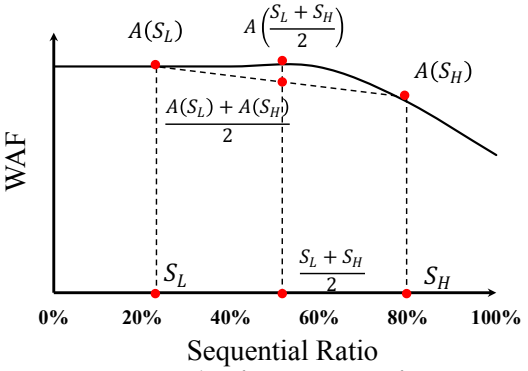

Fig. 13: Example of NVMe WAF function.

Furthermore, it is always true that harmonic mean is less than arithmetic mean, so we have:

$$\frac{A(S_H) + A(S_L)}{2} \geq \frac{2}{\frac{1}{A(S_H)} + \frac{1}{A(S_L)}} \geq 0. \tag{12}$$

Consequently, by combining Eq. 11 and 12, we get:

$$\frac{A(\frac{S_H+S_L}{2})}{2} \geq \frac{1}{\frac{1}{A(S_H)}+\frac{1}{A(S_L)}} \geq 0. \quad (13)$$

By substituting this result into Eq. 9, we have $lhs \geq 0$, only when all workloads are the same, the equal mark holds. This proves that separating sequential and random streams to different disk zones can reduce the overall TCO. Notice that if $k \gg 1$, we *cannot* guarantee that $TCO'(Greedy)$ is always worse than $TCO'(Grouping)$.

Step 2: Inductive Hypothesis

Step 1 proves that under the certain condition ($k \to 1$), it is better to group and send workloads to two homogeneous disk zones than to mix and send them to the disk pool. If we further divide each disk zone into two same size "subzones", and similarly group each workload subset into two similar-logical-data-write-rate ($\approx \frac{\lambda_\mathbb{J}}{4}$) sub-subsets with relatively high and low sequential ratio workloads. It is still true that the grouping approach is better than the greedy approach.

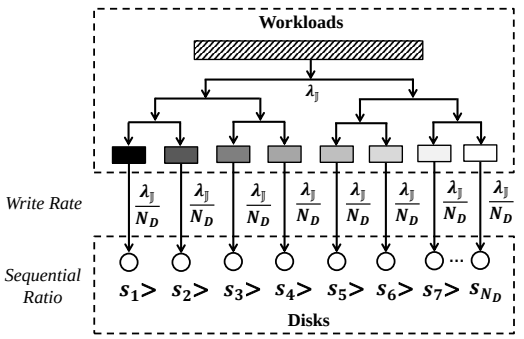

Fig. 14: Example of an ultimate allocation distribution of the best solution.

By iterating this process, we can prove that sorting all workloads and sending them to each disk by the order of different sequential ratios to balance each disk with similar write rate will lead to the best solution for minimizing data-avg TCO rate, if the workload set and homogeneous disk pool size are relatively large and workloads are fine-grained. Fig. 14 also illustrates an example of the ultimate allocation distribution of the best solution.

Step 3: Summarize

Based on steps 1 and 2, we prove that sorting all workloads and sending them to each disk by order is the best solution. However, in real implementation, to avoid the case that purely based on sequential ratio may lead to capacity or I/O throughput unbalance, we manually set up two (or more than two) disk zones and then balance the write rate inside each zone.