SSDKeeper: Self-Adapting Channel Allocation to Improve the Performance of SSD Devices

Renping Liu*, Xianzhang Chen*, Yujuan Tan*, Runyu Zhang*, Liang Liang†, Duo Liu*
* Key Lab. of Dependable Service Computing in Cyber Physical Society (Chongqing Univ.), Ministry of Education, China
*College of Computer Science, Chongqing University, Chongqing, China
†School of Microelectronics and Communication Engineering, Chongqing University, Chongqing, China

***Abstract*—Solid state drives (SSDs) have been widely deployed in high performance data center environments, where multiple tenants usually share the same hardware. However, traditional SSDs distribute the users' incoming data uniformly across all SSD channels, which leads to numerous access conflicts. Meanwhile, SSDs that statically allocate one or several channels to one tenant sacrifice device parallelism and capacity. When SSDs are shared by tenants with different access patterns, inappropriate channel allocation results in SSDs performance degradation. In this paper, we propose a self-adapting channel allocation mechanism, named SSDKeeper, for multiple tenants to share one SSD. SSDKeeper employs a machine learning assisted algorithm to take full advantage of SSD parallelism while providing performance isolation. By collecting multi-tenant access patterns and training a model, SSDKeeper selects an optimal channel allocation strategy for multiple tenants with the lowest overall response latency. Experimental results show that SSDKeeper improves the overall performance by 24% with negligible overhead.**

I. Introduction

Solid state drives (SSDs) are widely deployed in high performance computing environments for their advanced characteristics, such as low latency, high bandwidth, shock resistant, and low power consumption [1] [2] [3] [4]. With the capacity increase, one SSD hardware equipped with multiple channels is generally shared by multiple tenants. However, traditional SSDs uniformly stripping data issued by all tenants across all flash channels, which leads to numerous access conflicts and sacrifices overall system performance.

Open-Channel SSDs which directly expose low-level flash physics to applications provide hardware isolation by assigning one or several channels to each tenant. Though Open-Channel SSDs provide more control flexibility, allocating SSD channels blindly for multiple tenants could results in sub-optimal utilization of storage resources. Therefore, the SSD channel allocation strategies should be carefully optimized when serving multiple tenants.

Recently, many researchers have proposed various methods to improve SSD performance by reducing access conflicts or making full use of the raw resources in SSDs. Mao et al. [5] and Gao et al. [6] propose to reduce the potential access conflicts of SSDs by scheduling the I/O requests in the Linux kernel. Li et al. [7] optimize the latency of access conflicts by reducing the data transfer and flash access time. Although these methods alleviate access conflicts to a certain extent, they do not consider the situation where multiple tenants share the same SSD device. Access conflicts are further exacerbated due to the interference (i.e., the so-called noisy neighbor effect [8]) among multiple tenants. To mitigate the interference among multiple tenants, several recent works have suggested using Open-Channel SSDs to physically isolate each tenant so that they can only access the data on their own channels [9] [10]. However, these methods partition storage resources statically, resulting in sub-optional utilization of storage resources. Kim et al. [8] have proposed a method to improve the quality of service by making full use of the resources of each flash chip. However, allocating the appropriate number of channels for each tenant according to their workload access patterns to reduce access conflicts and improve the utilization of storage resources is rarely considered.

In this paper, we propose binding different numbers of SSD channels for different tenants depending on the access patterns when multiple tenants are sharing the same hardware. The fact that SSD channels can be operated independently helps avoid adverse impacts on each others performance [10]. However, allocating more channels to tenants with less access intensity can waste the storage resources. To allocate proper SSD resources to each tenant, we propose SSDKeeper, a self-adapting channel allocation mechanism for multiple tenants to share the same SSD device. SSDKeeper takes the access patterns as guides to allocate channels for multiple tenants and minimize access conflicts, which usually happen when multiple I/O requests access the same flash chip in a short time interval [6] [8] [11]. Appropriate channel allocation strategy can effectively reduce access conflicts in the case of multi-tenant shared SSDs.

To mitigate access conflicts among multiple tenants and improve the utilization of storage resources, SSDKeeper first trains a neural network model based on the response time of various mixed workloads running with different channel allocation strategies. Based on the model, SSDKeeper manages channel allocation for tenants running different workloads. To further improve the overall performance of SSD, SSDKeeper utilizes a hybrid page allocation approach. In all, SSDKeeper is compoased of a features collector, a strategy learner, a channel allocator, and a hybrid page allocator. Experimental results show that SSDKeeper improves the overall performance by 24% with negligible overhead. In summary, we make the

Corresponding Authors: Duo Liu and Xianzhang Chen.
E-mail: {liuduo, xzchen}@cqu.edu.cn.

1530-2075/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPS47924.2020.00103

following major contributions:

- We propose an efficient channel allocation mechanism for flash storage system, named SSDKeeper, taking the access patterns as guides to allocate channels for multiple tenants and improving the overall performance of SSDs.
- We collect a large number of optimal channel allocation strategies as the data set, and trained a neural network model based on that. We further utilize the model and propose a machine learning-based channel allocation strategy with self-adapting capability.
- We implement a prototype of SSDKeeper based on a universal flash simulator and evaluate the effectiveness of our proposed design.

The remainder of this paper is organized as follows. Section II and Section III present the background and the motivation of this paper, respectively. In Section IV, we describe the details of SSDKeeper. Section V presents the experiments and evaluations. Finally we discuss related work in Section VI, and conclude this paper in Section VII respectively.

II. Background

In this section, we introduce some background information about flash-based SSDs including traditional SSDs and Open-Channel SSDs. Then, we present artificial neural network with several optimizations.

A. SSDs Architecture

SSDs are composed by a controller, multiple flash chips, and some DRAM buffer as shown in Figure 1. In order to improve the performance, an SSD controller includes multiple channels to read/store data in parallel. Each channel is connected to several flash chips, and each flash chip contains multiple dies. A die is composed of several planes, and each plan is further divided into blocks. Each block consists of hundreds of pages. Each page corresponds to a word line, which has thousands of storage units. Die is the basic unit to accept and execute flash commands. Cache register and page register in each plane are used to cache data to balance the speed difference between the host and the flash media. In addition, a block is the basic erase unit and a page is the read and write unit.

SSDs suffer from two constraints, erase-before-write and endurance problem, which make SSD can not directly provide a block device interface to the upper layer like the traditional disk. To hide these shortcomings, flash translation layer (FTL) is employed. FTL is mainly composed of address mapping, garbage collection, and wear-leveling. Address mapping [12] is used to record the mapping between logical flash pages and physical flash pages. Garbage collection is triggered to reclaim invalid space in the allocated flash blocks [13]. Wear leveling technique is used to erase all flash blocks evenly to prolong the lifetime of SSDs. FTL abstracts SSDs as a traditional block device that is the same as the disk from the top view. However, FTL is a black box and uses shared channels mapping schemes without considering access patterns, which results in the number of access conflicts. In addition, 3D stacking makes SSDs capacity increasing and multiple tenants can share the same SSD device, which further aggravates access conflicts.

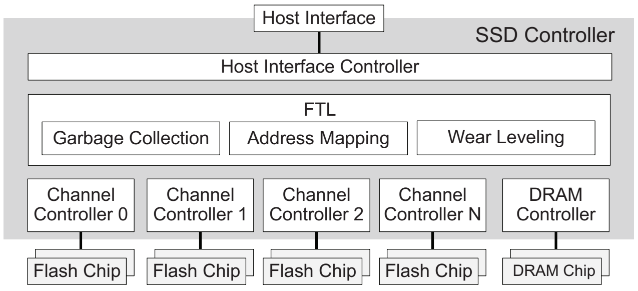


Figure 1: Structure of SSDs.

A new class of SSDs named Open-Channel SSDs has been developed to expose the internal parallelism to the host. The host is allowed to access physical address directly. I/O isolation, one of the most important properties of Open-Channel SSDs, provides a method to divide the capacity of the SSD into several isolated I/O channels [14]. Multiple tenants share the same SSD with different channels makes performance isolation. However, blindly partitioning internal storage resources for each tenant can not make full use of storage resources. In this paper, to improve the overall performance of the storage system, SSDKeeper ensures full use of storage resources while reducing access conflicts when multiple tenants share the same SSD device.

B. Artificial Neural Network

SSDKeeper is a machine learning-based mechanism that allocates SSDs' channels according to the access patterns of multiple tenants, resulting in the lowest overall response latency of the flash storage system. Since it is difficult to find a way to adapt to varieties of multi-tenant access patterns, machine learning provides SSDKeeper the ability to make a decision to allocate flash channels for tenants with different access patterns. Artificial Neural Network (ANN) is a class of machine learning algorithm. An ANN consists of multiple layers, and every layer contains many neurons. The non-linear activation function of an ANN makes it different from the linear perceptron. ANN can distinguish data that is not linearly separable. SSDKeeper utilizes this feature to distinguish tenants with different access patterns, and select the channel allocation strategy with the lowest overall response latency.

Backpropagation algorithm is used to efficiently train a high accuracy ANN model following a gradient-based optimization algorithm that exploits the chain rule. In the phase of learning, the weight and the bias of neurons are updated by calculating the gradient of the loss function. The update function is shown as:

$$\begin{aligned} w^l_{jk} &:= w^l_{jk} - \alpha \frac{\partial C}{\partial w^l_{jk}} \\ b^l_{jk} &:= b^l_{jk} - \alpha \frac{\partial C}{\partial b^l_{jk}} \end{aligned} \tag{1}$$

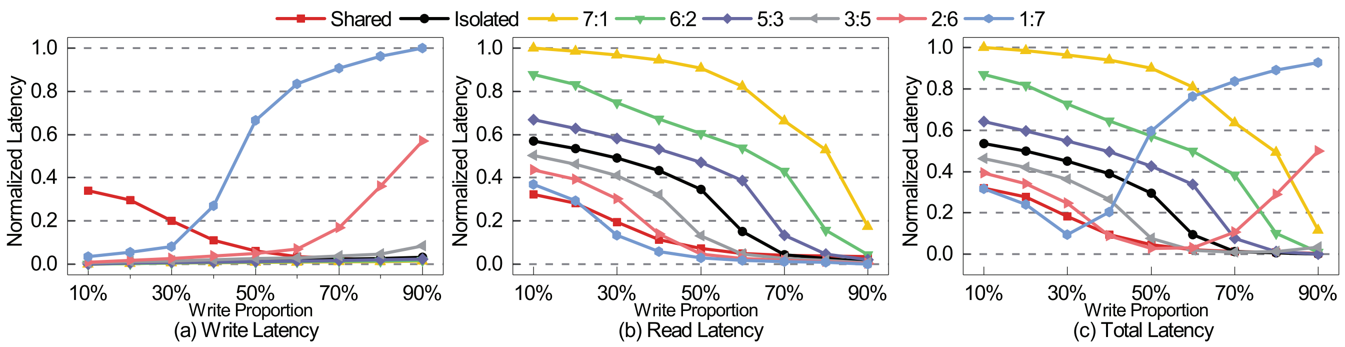


Figure 2: Normalized read, write, and total response latency when two tenants with different write/read proportion share the same SSD device.

where w^l_{jk} and b^l_{jk} are the weight and the bias at position (j,k) of l^{th} layer. C represents cost function and α is the learning rate. According to Equation 1, ANN can choose a direction that would decrease the error rate, and continue iterating until the cost function converges to the minimum. Stochastic gradient descent (SGD) is a variant of gradient descent. SGD does not calculate the entire data set (which is redundant and inefficient), but only a small portion of the data sample or random selection. When the learning rate is low, SGD produces the same performance as regular gradient descent. However, in recent years, many new optimizations have been proposed to solve the complex training scenarios with the poor performance of the gradient descent method. One of the most widely used and practical optimizations for training neural network models is Adam. It combines the advantages of two SGD extensions: Adaptive Gradient Algorithm (AdaGrad), which works well with sparse gradients, and Root Mean Square Propagation (RMSProp), which works well in on-line and non-stationary settings [15]. SSDKeeper explores several optimizations including SGD with momentum and Adam with different activation functions.

III. Motivation

When multiple tenants sharing the same SSD device, the I/O requests of different tenants exacerbate access conflicts [8]. Since an I/O request of a tenant may access multiple flash chips, and the latency of the request depends on the slowest chip access. The larger size of the request is, the higher probability that it will be blocked by the request of other tenants. Although Open-Channel SSDs provide a new approach to isolate multi-tenant I/O requests to avoid access conflicts among multiple tenants, allocating SSD channels blindly for tenants could result in the waste of storage resources. For example, allocating more channels to tenants with less access intensity could waste the storage resources. On the contrary, traditional SSDs can take advantage of the chip parallelism to respond to I/O requests by sharing all channels among tenants, but a large number of access conflicts are generated. Therefore, these two approaches are hard to adapt to varying access patterns of multiple tenants.

To understand this problem, we compare those two different approaches for sharing SSDs. The first approach stripes data from all the tenants across all the flash channels, just as exiting SSDs do (shared SSDs). This strategy provides the maximum parallelism for each I/O. The second approach uses the feature from Open-Channel SSDs that provide the hardware isolation by assigning a certain number of channels to each workload (isolated SSDs). In this paper, *Shared* represents channel allocation strategy like traditional SSDs, and *Isolated* means all tenants split the channels equally. We investigate how the performance change when multiple tenants are concurrently accessing the same SSD device. We assume that a tenant corresponds to a workload. We conduct a set of experiments. In each experiment, there are two workloads that always keep the total number of I/O requests fixed. One workload only contains write operations. The other is all read operations. We control the write proportion of synthetic workloads. Two different workloads represent two tenants with different access patterns. The detailed experimental setup of the SSD device can be found in Section V.

A. The drawbacks of shared SSDs and isolated SSDs

Figure 2 shows normalized read, write, and total response latency when two tenants with different write/read proportion share the same SSD device. We set the SSD device with 8 channels, and we have listed all channel allocation strategies in Figure 2. In this paper, *7:1* expresses allocating seven channels to the write-oriented workload, the other one channel to the read-oriented workload. The other channel allocation strategies have the same meaning, with the first number representing the number of channels allocated to the write-oriented workload.

We sweep nine different values for the write proportion, ranging from 10% to 90%. As shown in Figure. 2 (a), when the write proportion is above 60%, write latency of *Shared* is close to *7:1*, *6:2*, *5:3* and *Isolated*. Because of the configuration of the SSD, four channels are enough to handle those write requests. If we reduce channels for write workload, like *3:5*, *2:6* and *1:7*, write latency increase dramatically. However, write latency of *Shared* is much higher as write proportion below 50% due to the amount number of access conflicts with read operations. Read operation must read data from the specific flash chips which hold the data and has priority to respond because of the lower flash chip accessing time. Response latency of the write operations is extended by plenty of the read requests in shared SSDs.

Figure 2 (b) shows the read response latency of all channel allocation strategies when two tenants sharing the same SSD device. As the number of channels allocated to the read-oriented workload increases, read response latency decreases. In the case of *3:5*, it always keeps the lower read latency compared with *Isolated*. Note that *3:5* and *Isolated* have similar write response latency as shown in Figure2 (a). In this situation, we conclude that allocating one more channel to the read-oriented workload is better than allocating this channel to the write-oriented workload. Therefore, isolated SSDs fail to make full use of storage resources.

B. Single channel allocation strategy and self-adjusting channel allocation strategy

We utilize the sum of write response latency and read response latency to evaluate the overall performance, as shown in Figure 2 (c). We can observe that the total response latency of *1:7* is the lowest when write proportion is around 30%. As the write ratio increases, *Shared*, *2:6* and *5:3* have the lowest total latency respectively. Therefore, single channel allocation method (like shared SSDs or isolated SSDs) can not adapt to variable mixed workloads. Furthermore, the performance of different methods varies by up to 10.6 times, such as *7:1* and *2:6* at 50% in Figure 2 (c). These observations motivate us to find a self-adjusting channel allocation strategy to copy with varying multi-tenant access characteristics.

IV. SSDKeeper Design

In this section, we describe the details of SSDKeeper, a self-adapting channel allocation mechanism for sharing SSD settings. The basic idea of SSDKeeper is to allocate an appropriate number of channels for tenants to reduce access conflicts. SSDKeeper can also optimize resource allocation and improves the utilization of storage resources. To achieve this goal, SSDKeeper uses a machine learning-assisted method to learn channel allocation strategies. SSDKeeper firstly collects the response latency by running mixed synthetic workloads under varying channel allocation strategies as the data set. Then a model is trained with the data set. The inputs of the model are characteristics of a mixed workload, and the output is the best channel allocation strategy that makes the total response latency lowest. The well-trained model makes it easy to cope with mixed workloads with different access patterns. To further improve the overall performance of the SSD device, SSDKeeper utilizes a hybrid page allocation method. This method uses different page allocation modes for workloads with different read/write patterns. SSDKeeper runs in FTL, so it can be adapted to traditional SSDs and Open-Channel SSDs depending on where FTL is running.

A. Overview

Traditional shared SSDs cause a large number of access conflicts extending total response latency, while isolated SSDs

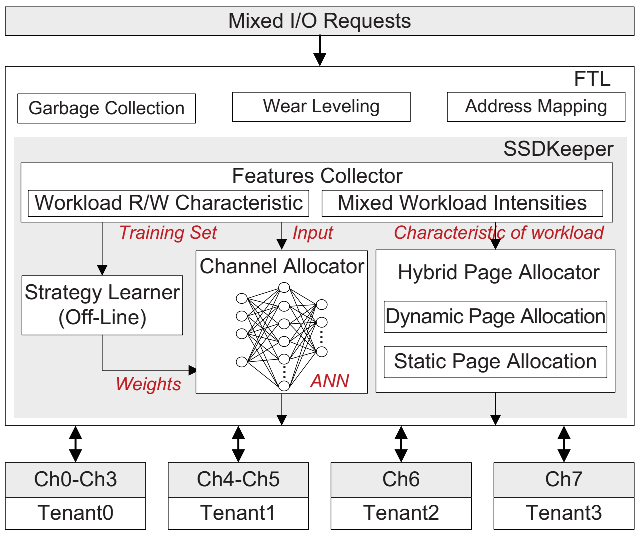


Figure 3: SSDKeeper Design.

cannot fully exploit storage sources of SSDs. SSDKeeper can effectively alleviate these problems. Figure 3 shows the design of SSDKeeper. SSDKeeper allocates channels for multiple tenants including several components: features collector, strategy learner, and channel allocator. In order to further improve the overall performance of the SSD, hybrid page allocator is deployed to SSDKeeper.

Features collector is designed to collect training data set and access patterns from the current storage system when a mixed workload runs on the same SSD device.

Strategy learner utilizes the access patterns of a mixed workload as features and regards the channel allocation strategy with the lowest overall response latency as the label. Based on a mass of features-label pairs, SSDKeeper has the ability to train a model to cope with varying access patterns of multiple tenants.

Channel allocator uses the well-trained model to allocate channels for multiple tenants. The well-trained model has observed the optimal channel allocation strategies for multiple tenants with different access patterns in the current SSD configuration. The output of channel allocator is an optimal channel allocation strategy that minimizes the overall response latency of the SSD hardware.

Hybrid page allocator adjusts the page allocation mode based on the read/write characteristic of different tenants to further improve the overall performance of the SSD hardware.

B. Features Collector

In order to allocate channels for multiple tenants to minimize access conflicts and improve the utilization of storage resources, SSDKeeper needs to collect features of the running mixed workload currently. Therefore, we design a features collector to assist SSDKeeper in making the decision to allocate channels. The features collector collects the state information of the running mixed workload, including read/write characteristic of each workload and intensities of the mixed workload.

Read/Write Characteristic of Each Workload. Since the performance of read operations affected by write operations, read/write characteristic of a workload plays an important role when multiple tenants share the same SSD device. In this paper, we only consider two types of workload including write-oriented and read-oriented. SSDKeeper is unconscious of the type of each workload at first. After a period of time to run the mixed workload concurrently, SSDKeeper can judge the type of the running workloads according to record their I/O read/write characteristics. In addition, we utilize workloadID to distinguish the different workloads. The method of obtaining the workloadID inside the SSD can refer to [16] [17].

Intensities of Mixed workload. The intensity of I/O requests is a key factor affecting SSD performance. A workload with a higher I/O request intensity is more likely to cause access conflicts. Therefore, SSDKeeper utilizes the overall intensity of the mixed workload as one of the features. SSDKeeper represents the overall intensity of a mixed workload based on the total number of I/O requests over a period of time and divides the overall intensity into multiple levels. The higher the level means a higher overall intensity. In addition, the relative intensities of each workload also affect channel allocation strategy of SSDKeeper. As shown in Figure 2(c), when the write proportion is 30%, the average response latency of *1:7* is 47.8% better than the performance of shared SSD. And after the write proportion exceeds 40%, *1:7* would no longer be considered the best. Note that SSDKeeper mainly considers read-dominated or write-dominated workloads. The proportion of read or write operations to total requests during this time can reflect the relative intensity of each workload. Therefore, allocating appropriate numbers of channels to workloads with different relative intensities can effectively improve the overall performance of the SSD. We select the overall intensity of a mixed workload and relative intensities of each workload as guides for SSDKeeper to allocate channels.

C. Strategy Learner

The main function of SSDKeeper is to select the best strategy to allocate channels for multiple tenants based on the features collector obtained. To achieve this goal, SSDKeeper utilizes a strategy learner to learn the channel allocation strategy with the lowest response latency. Due to different patterns of mixed workloads, we use machine learning to assist. The strategy learner consists of label generation and model training.

Label Generation. For collecting the training data set, SSDKeeper tries all channel allocation strategies, and then select the strategy with the lowest overall read and write response latency as the label. Different numbers of tenants sharing the same SSD device results in different allocation strategies. We assume a fixed SSD configuration with 8 channels as shown in Table I. When two tenants share an SSD device, the number of allocation strategies are 8 (*Shared*, *Isolated*, *7:1*, *6:2*, *5:3*, *3:5*, *2:6*, *1:7*), as shown in Figure 2. When four tenants share an SSD device, the number of allocation strategies are 42, including an additional 34 allocation strategies (*5:1:1:1*, *4:2:1:1*, *3:3:1:1*, *3:2:2:1* and so on). SSDKeeper divides the channels into two or four parts without considering three parts (such as *6:1:1*). During the strategy learning phase, SSDKeeper runs a mixed workload in all allocation scenarios and records the strategy with the lowest latency as the label.

Algorithm 1 Training phase of SSDKeeper

```
Input: $total\_data\_number$: the number of data set;
    $max\_iterations$: max iterations of training.
Output: Parameters of neural network.
 1: $data\_number = 0$.
 2: $iteration = 0$.
 3: while $data\_number < total\_data\_number$ do
 4:     Generate a synthetic mixed workload and fed it to the
        simulator.
 5:     Run all channel allocation strategies.
 6:     Select a strategy with the lowest total latency as the
        label.
 7:     $data\_number = data\_number + 1$.
 8: end while
 9: $Data\_preprocessing()$.
10: while $iteration < max\_iterations$ do
11:     Update parameters.
12:     $w^l_{jk} := w^l_{jk} - \alpha\frac{\partial C}{\partial w^l_{jk}}$.
13:     $b^l_{jk} := b^l_{jk} - \alpha\frac{\partial C}{\partial b^l_{jk}}$.
14:     $iteration = iteration + 1$.
15: end while
16: return parameters.
```

Model Training. Based on the input features, SSDKeeper can find the corresponding label as the best channel allocation strategy in the data set. However, features of mixed workloads are diverse. It is almost impossible to encounter two mixed workloads with the same feature. Therefore, SSDKeeper needs a large amount of data as training data set to cope with scenarios that have not been seen. SSDKeeper uses ANN to train the model. The details of training phase are described as Algorithm 1. The training phase of the model is offline without affecting the performance of the SSD hardware. We can train the model on the host side and then sends the parameters to the FTL as well as training the model when the SSD controller is idle. Compared to other machine learning algorithms such as Bayesian or k-nearest neighbors, ANN does not need to save all the training data set, only need to store a small number of parameters.

D. Channel Allocator

Based on the model trained by the strategy learner, we completed a channel allocator. The channel allocator takes the features obtained by the features collector as input and outputs the optimal channel allocation strategy through the forward propagation of the neural network. The neural network is offloaded to the channel allocator, as shown in Figure 3. We assume that the trained network contains L layers and L_i represents the i^{th} layer. The layer of i^{th} has N_i neurons, and each neuron contains the weight and the bias accounting for 16 bytes. Total storage space requires $\sum_{i=1}^{L} 16 * N_i$. The number of float point multiplications required for forward propagation is $\sum_{i=0}^{L-1} N_i * N_{i+1}$, where L_0 represents the input layer. In this paper, the input layer has 9 features. The hidden layer and output layer have 64 and 42 neurons respectively. Therefore, the storage overhead and computational overhead of SSDKeeper are negligible. The accuracy of the model on the test data set reaches 94.5%.

E. Hybrid Page Allocator

Different flash page allocation modes have different impact on SSD performance [18] [19]. We implement a hybrid page allocator for SSDKeeper. Since the workloads with different patterns are allocated to the specified number of channels, the hybrid page allocator can adopt different page allocation mode for those channels, thereby further improving the overall performance of the SSD.

For write requests, dynamic page allocation can reduce response latency, because as long as there is idle channel and chip, the data that needs to be stored is immediately written to the idle channel and chip. In the case of static page allocation, the channel and chip have been allocated in advance according to the logical address of this write request. The response latency is extended if the channel and chip allocated to the write request are occupied.

For read requests, static page allocation can improve response time, because it enables successive logical pages to be distributed across different channels and chips. When a read request arrives, multiple channels and chips can simultaneously serve this read request, taking advantage of channel parallelism [18]. However, in the case of dynamic page allocation, the data of successive read requests is not completely dispersed in different channels and chips when written to the flash memory. Instead, the data may be stored on the same channel or even on the same chip. When the read request needs to be responded, the parallelism between channels cannot be utilized.

Therefore, SSDKeeper uses static page allocation mode for the read-dominated workload and selects dynamic page allocation mode for the write-dominated workload. The workflow of SSDKeeper is shown in Algorithm 2.

Algorithm 2 SSDKeeper with hybrid-page allocation

```
Input: An I/O request of mixed workload; $T$: time for collecting features; $t$: the current time.
Output: PA: physical address of the I/O request.
 1: if $t < T$ then
 2:     Collect access characteristics of mixed workload.
 3:     $workloadID = get\_workloadID()$.
 4:     $PA = hybrid\_page\_alloc(shared, workloadID)$.
 5: else
 6:     if $t == T$ then
 7:         $features = get\_features()$.
 8:         $strategy = predict(features)$.
 9:     else
10:         $workloadID = get\_workloadID()$.
11:         $PA = hybrid\_page\_alloc(strategy, workloadID)$.
12:     end if
13: end if
14: return $PA$.
```

V. Evaluation

To evaluate the proposed SSDKeeper, we modified the universal flash simulator to implement features collector, strategy learner, channel allocator, and hybrid page allocator. Our approach is general enough and can be used in traditional SSDs. It can be also used in Open-Channel SSDs by modifying the file system or calling the library in userspace. In the rest of this section, we first give the experimental setup, then discuss the loss and accuracy of the model training phase. At last, we analyze the performance of SSDKeeper.

A. Experimental Setup

We use a trace driven simulator named SSDSim [18] to evaluate the proposed SSDKeeper. SSDSim has been widely used in the exploration of SSDs. SSDKeeper is implemented by modifying the simulator source code. For the features collector, we record the read/write characteristic of each workload over a period of time. We utilize workloadID to distinguish the different workloads. The method of obtaining the workloadID inside the SSD can refer to [16] [17]. We utilize scikit-learn for strategy learning to train a neural network model. Scikit-learn is a simple and efficient toolset for data mining and data analysis in python. Evaluation experiments are conducted on a workstation equipped with two Intel(R) Xeon(R) E5-2640 processors and 256GB DRAM. The operating system is Ubuntu 16.04 with Linux kernel 4.4.4.

In order to better verify the proposed mechanism, we construct a modest SSD with 8 channels and 2 flash chips per channel. Table I summarizes the SSD configuration used in our experiments. Since SSDKeeper is a channel allocation mechanism based on machine learning, a large amount of data needs to be used as the training data set. To achieve this goal, we follow the same method as [17]. The mixed workloads for training are synthetic. We mainly change the read/write characteristics and read/write proportion to synthesize the new mixed workloads. We fixed the total number of I/O requests to 2 million. Note that changing the read or write proportion of each workload means different relative intensities because in this paper we explore the mixed workload that each workload is read-dominated or write-dominated. At last, we mix real workloads to simulate multiple tenants to evaluate SSDKeeper, and the real workloads as shown in Table II.

In our evaluation, we mix four workloads to simulate four tenants sharing the same SSD device. The features collector

Table I: SSD Configuration

Parameter	Value	Parameter	Value
Page size	16KB	Read latency	20μs
#of pages/block	128	Write latency	200μs
#of blocks/plane	4096	Erase latency	1.5ms
#of planes/chip	4	Physical capacity	512GB
#of chips/channel	2	#of channels	8

Table II: Characteristics of the Evaluated I/O Workloads

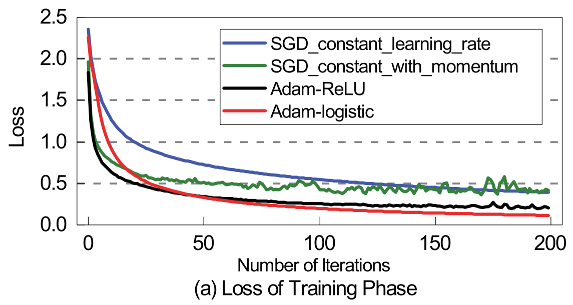
Workload	Write Ratio	Read Ratio	Request Count
mds_0	88%	12%	1211034
mds_1	7%	93%	1637711
rsrch_0	91%	9%	1433654
prxy_0	97%	3%	12518968
src_1	5%	95%	45746222
web_2	1%	99%	5175367

outputs a nine-dimensional vector representing the features of the mixed workload based on Section IV. The nine-dimensional vector consists of the overall intensity level of mixed workload (1-D), the read/write characteristic of each workload (4-D) and the read/write proportion of each workload (4-D), such as *([5] [1, 0, 1, 0] [0.1, 0.2, 0.3, 0.4])*. The overall intensity level of mixed workload represents the total number of I/O requests in a period of time and we divide it into twenty levels. The read/write characteristic of each workload are expressed as 0 (write) and 1 (read). The sum of read/write proportion of each workload is equal to 1.

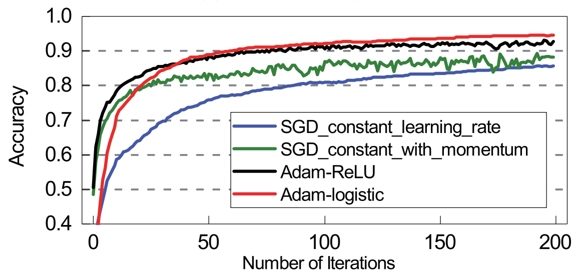
B. Model Training

In order to train a model to cope with the varying mixed workloads, we collect a large amount of data. We test read and write response latency under different channel allocation strategies by randomly changing the features of the mixed workloads. We generate 5,000 mixed workloads with different access patterns. Each mixed workload corresponds to 42 allocation strategies as described in Section IV, generating a total of 210,000 (42*5,000) records. We choose the best channel allocation strategy for the 5,000 mixed workloads from the 210,000 records based on the lowest response latency and label them as the data set. We shuffle the 5,000 pieces of data, and the proportion between the training data set and test data set is *7:3*.

We implement four optimization methods for ANN to balance the model accuracy and the training overhead. Through experiments, we finally choose the initial learning rate of SGD to be 0.2, momentum to be 0.9, and Adam's initial learning rate to be 0.02. For Adam, ReLu and logistic represent different activation functions. Loss and test accuracy in the training phase are shown in Figure 4. With the increase of the number of iterations, the four optimization methods can reduce loss and finally make the neural network converge, as shown in Figure 4(a). Compared with SGD, Adam has obvious advantages in model training in this paper. Figure 4 (b) shows the test accuracy changes during the training phase



(a) Loss of Training Phase



(b) Accuracy on Test Data Set

Figure 4: The loss curve of training phase and accuracy curve on test data set with different optimization.

Table III: The final loss, accuracy and training time of the trained model with different optimization.

Optimizer	Loss	Accuracy	Training Time(ms)
SGD	0.39	85.6%	14389
SGD-momentum	0.41	88.1%	13672
Adam-ReLU	0.21	92.7%	15196
Adam-logistic	0.11	94.5%	19646

of the four optimization methods. With the decrease of loss in the training data set, the accuracy of the model is gradually improved. The accuracy of Adam-logistic is 94.5%, the highest among the four methods when iterations are 200. However, comparing with other optimization methods, the training time of Adam-logistic is 36.5%, 43.6%, 29.3% more than that of the other methods, as shown in Table III. The reason is that the computational complexity of logistic as the activation function is greater than ReLU. Note that our training phase is off-line, so it will not affect the performance of the SSD device. The accuracy of the model reflects the probability of SSDKeeper adopting the optimal strategy when coping with mixed workload with different patterns. In addition, SGD has a lower loss in the training phase and a lower accuracy on test data set comparing with SGD-momentum due to overfitting.

C. Performance Analysis

In order to evaluate the performance of SSDKeeper, we mix real workloads to simulate multiple tenants sharing the same SSD as shown in Table IV. For each mixed workload, we first mix the four workloads in chronological order and then take one million traces. Table V shows features of the four mixed workloads and channel allocation strategies by SSDKeeper selected. From Table V, we can observe that SSDKeeper has

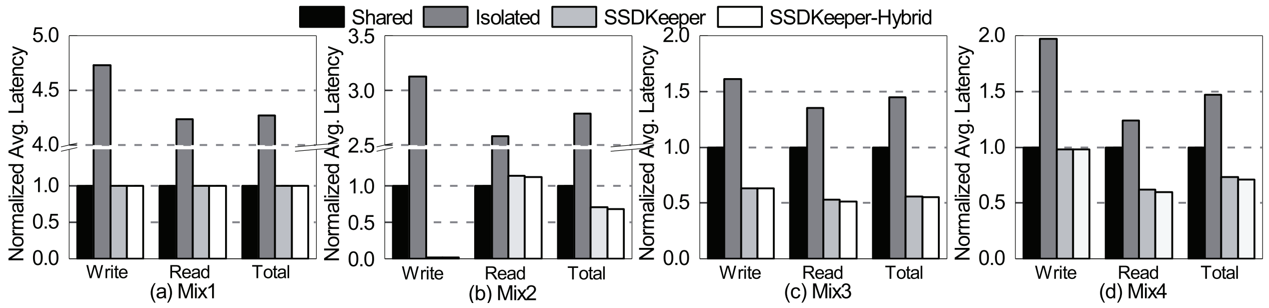


Figure 5: The write, read and total responded latency of four mixed workloads with different characteristics.

Table IV: Mixed Workloads

Mixed Workload	Workloads
Mix1	mds_0, mds_1, rsrch_0, prxy_0
Mix2	prxy_0, src_1, rsrch_0, mds_1
Mix3	web_2, rsrch_0, prxy_0, mds_0
Mix4	rsrch_0, web_2, mds_1, prxy_0

Table V: Characteristics of the four mixed workloads and channel allocation strategies of SSDKeeper.

Mixed Workload	Characteristics of Mixed Workload	SSDKeeper Channel Allocation
Mix1	[3] [0,1,0,0] [0.08,0.09,0.08,0.75]	Shared
Mix2	[18] [0,1,0,1] [0.21,0.72,0.02,0.05]	1:7
Mix3	[16] [1,0,0,0] [0.67,0.26,0.03,0.04]	5:1:1:1
Mix4	[17] [0,1,1,0] [0.65,0.03,0.27,0.05]	4:2:1:1

the ability to adaptively select the best channel allocation strategy based on the different mixed workload access patterns. SSDKeeper not only includes the channel allocation strategy of traditional *Shared*, but also contains other new strategies such as *1:7* or *5:1:1:1*.

Figure 5 shows read, write, and total response latency of four mixed workloads, respectively. We use *Shared* as the baseline because sharing all channels for multiple tenants is the most widely used. Note that *Isolated* refers to the situation where SSD channels are evenly assigned by all workloads in this paper. For Figure 5(a), the best selected channel allocation strategy by SSDKeeper is *Shared*, so it has the same performance as *Shared*. If the channels are allocated blindly in the isolated way, the overall performance will be reduced by 327%. However, *Shared* is unable to fit all scenarios. As shown in Mix2, Mix3, and Mix4, the overall performance of SSDKeeper is improved by 29.6%, 43.2% and 27.1% comparing with traditional *Shared* allocation strategy, respectively. Furthermore, SSDKeeper with hybrid page allocation mode improves the average overall performance by 2.1%.

D. Channel Allocation Analysis

To illustrate a single channel allocation strategy could not adapt to the varying access patterns of mixed workloads, we test and draw the channel allocation strategies of SSDKeeper under different mixed workloads access patterns,

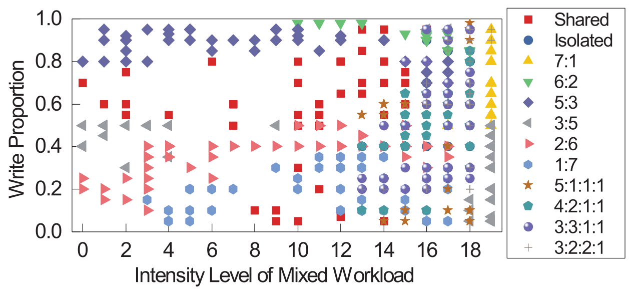


Figure 6: Channel allocation strategies of SSDKeeper for multiple tenants with different access characteristics.

as shown in Figure 6. Since the four workloads are mixed under an 8-channel SSD configuration, there are 42 channel allocation strategies that are difficult to represent in the figure. We simplify the representation of the strategies that divide the channel into four parts. For example, *5:1:1:1*, *1:5:1:1*, *1:1:5:1* and *1:1:1:5* are all represented by *5:1:1:1*, where 5 represents the number of channels allocated to the workload with the highest relative intensity. *4:2:1:1*, *3:3:1:1* and *3:2:2:1* use the same simplification. For the strategies that divide the channel into two parts, the first number is expressed allocating channels for write-dominated workloads. For example, *7:1* represents the mixed workload is divided into two parts according to write/read characteristic, allocating seven channels for all write-dominated workloads and one channel for all read-dominated workloads. We use the total write proportion of the four mixed workloads as the Y-axis to further simplify the analysis of SSDKeeper channel allocation strategies. Note that *[16][0,1,1,1][0.4,0.2,0.2,0.2]* and *[16][0,0,1,1][0.2,0.2,0.1,0.5]* have the same total write proportion and intensity level, but SSDKeeper has different channel allocation strategy for these two mixed workloads. So there are a lot of overlaps in Figure 6, which does not affect our analysis of the trend of SSDKeeper channel allocation strategies. The X-axis of Figure 6 represents the intensity level of mixed workload, ranging from 0 to 19.

As shown in Figure 6, we can observe that when the intensity level of mixed workloads is low (less than 3), the number of channels allocated by SSDKeeper for write-dominated workloads increases with the increase of write proportion, which are *2:6*, *3:5*, *Shared*, *5:3*, respectively. When the mixed workloads with smaller intensity levels (4 to 7) and write proportion(less than 0.2), SSDKeeper only allocates one channel to the write-dominated workloads and the other seven channels to the read-dominated workloads. Because allocating too many channels to the write-dominated workloads in this case would be a waste of resources. As the intensity level of mixed workloads increases, the channel allocation strategies of SSDKeeper become more complex. For intensity level is equal to 19, when write proportion is greater than 0.5, SSDKeeper allocates seven channels to the write-dominated workloads, only one channel to the read-dominated workloads, which makes the overall performance of the SSD reach the highest.

VI. Related Work

There are various of previous work related to SSDKeeper. Most of the related work lies in the following three areas: I/O isolation for multiple tenants, I/O scheduling to reduce access conflicts and intelligent SSDs. In this section, we discuss these related works separately.

I/O Isolation for Multiple Tenants. 3D stacking and multi-level cell technology increase the capacity of flash storage dramatically, which promotes high-density flash storage to be shared by multiple tenants. However, tenants with a high read-to-write ratio expect to monopolize the SSD. Traditional SSD management algorithm makes it difficult for multiple tenants to efficiently share an SSD. The main reason is that the current FTL can not open the communication barrier between applications and SSDs. Thus, Kang et al. [20] provides an interface for the host to tag data, and organize the data with similar lifetimes to improve the performance of garbage collection. In addition, Open-Channel SSDs [14] provide a way to divide the SSD into a series of I/O channels. Each channel has corresponding flash chips and allows multi-tenant applications sharing the same SSD device without a noisy neighbor effect. OPS Isolation [21] proposes a novel allocation scheme to avoid interfering by isolating virtual machines. FlashBlox [10] utilizes flash parallelism to improve isolation by partitioning resources on dedicated channels and dies for different tenants. Kim et al. [8] present utilitarian performance isolation to reduce the average response time by maximizing the utilization of each flash chip. These techniques provide methods for I/O isolation to reduce access conflicts. SSDKeeper is proposed to utilize access patterns of the mixed workload to allocate channels to multiple tenants for performance isolation.

I/O Scheduling to Reduce Access Conflicts. I/O scheduling is the traditional method to reduce access conflicts. Gao et al. [6] proposes a novel scheduling on the host side to minimize access conflicts among I/O requests, which puts requests without access conflicts in the same batch to make full use of flash parallelism. Elyasi et al. [22] use slack-enable scheduling to reorder the sub-requests and issue to each flash chip to reduce the latency of response. Guo et al. [23] dispatch hot write data to the same physical block to alleviate the garbage collection overhead, and dispatch hot read data to different channels to exploit the channel level internal parallelism. ParaFS [24] schedules read/write/erase requests to multiple flash channels for more consistent system performance. Amphibian I/O scheduler [5] fully considers characteristics of the high-level I/O requests and the low-level parallelism of flash chips to improve the performance of SSD-based storage systems. Cui et al. [25] propose an I/O scheduler to give scheduling priority to fast writes and fast reads for access conflicts reduction. However, SSDKeepr reduces access conflicts between multiple tenants from a novel perspective.

Intelligent SSDs. At present, the concept of intelligence is getting more and more attention from academia to industry. AutoSSD [26] proposes an autonomous SSD architecture that self-manages FTL tasks by dynamically adjusting through feedback control to maintain a high-level of QoS performance. Kang et al. [27] [28] use reinforcement learning to assist garbage collection, and learn the access patterns of different workloads online to properly adjust operations of garbage collection, effectively reducing the long tail latency. In order to provide high I/O throughput, Ji et al. [9] utilize reinforcement learning to assist merging I/O requests from different workloads by learning the characteristics of different I/O patterns. Zhang et al.[29] develop a reinforcement learning driven page-level mapping and caching scheme to adapt and respond to ever-changing I/O streams in FTL. In this work, SSDKeeper uses a machine learning-assisted method to reduce access conflicts and make full use of storage resources for SSDs.

VII. Conclusion

In this paper, we have proposed SSDKeeper, a self-adapting channel allocation mechanism for multiple tenants to share the same SSD device based on the well-trained neural network. By collecting multi-tenant access patterns and access intensities, SSDKeeper selects an optimal channel allocation strategy for multiple tenants to minimize the overall response latency. In addition, SSDKeeper adopts a hybrid page allocation approach to further improving the performance of SSDs. Experimental results show that the proposed mechanism improves the overall performance by 24%. To demonstrate the effectiveness of SSDKeeper, we draw the loss curve of the training phase and accuracy curve on test data set with different optimizations. Moreover, we evaluate the performance and analyze channel allocation strategies of SSDKeeper.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable feedback and improvements to this paper. This work is partially supported by grants from the National Natural Science Foundation of China (61672116, 61601067, 61802038), Chongqing High-Tech Research Key Program (cstc2019jscx-mbdx0063), the Fundamental Research Funds for the Central Universities under Grant (0214005207005, 2019CD-JGFJSJ001), Chongqing Youth Talent Support Program, and China Postdoctoral Science Foundation (2017M620412).

REFERENCES

[1] D. Liu, L. Yao, L. Long, Z. Shao, and Y. Guan, "A workload-aware flash translation layer enhancing performance and lifespan of tlc/slc dual-mode flash memory in embedded systems," *Microprocessors and Microsystems*, vol. 52, pp. 343–354, 2017.

[2] B. Mao, S. Wu, H. Jiang, Y. Yang, and Z. Xi, "Edc: Improving the performance and space efficiency of flash-based storage systems with elastic data compression," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 29, no. 6, pp. 1261–1274, 2018.

[3] J. Guo, W. Wen, J. Hu, D. Wang, H. Li, and Y. Chen, "Flexlevel nand flash storage system design to reduce ldpc latency," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 7, pp. 1167–1180, 2016.

[4] Y. Lu, J. Shu, and W. Zheng, "Extending the lifetime of flash-based storage through reducing write amplification from file systems," in *Presented as part of the 11th {USENIX} Conference on File and Storage Technologies ({FAST} 13)*, 2013, pp. 257–270.

[5] B. Mao, S. Wu, and L. Duan, "Improving the ssd performance by exploiting request characteristics and internal parallelism," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 37, no. 2, pp. 472–484, 2018.

[6] C. Gao, L. Shi, M. Zhao, C. J. Xue, K. Wu, and E. H.-M. Sha, "Exploiting parallelism in i/o scheduling for access conflict minimization in flash-based solid state drives," in *Proceedings of the 30th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2014, pp. 1–11.

[7] Q. Li, L. Shi, C. Gao, K. Wu, C. J. Xue, Q. Zhuge, and E. H.-M. Sha, "Maximizing io performance via conflict reduction for flash memory storage systems," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, pp. 904–907.

[8] B. S. Kim, "Utilitarian performance isolation in shared ssds," in *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2018.

[9] C. Ji, Q. Li, C. Fu, C. J. Xue *et al.*, "Maximizing i/o throughput and minimizing performance variation via reinforcement learning based i/o merging for ssds: work-in-progress," in *International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. IEEE, 2018, pp. 1–2.

[10] J. Huang, A. Badam, L. Caulfield, S. Nath, S. Sengupta, B. Sharma, and M. 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)*, 2017, pp. 375–390.

[11] N. Elyasi, M. Arjomand, A. Sivasubramaniam, M. T. Kandemir, and C. R. Das, "Content popularity-based selective replication for read redirection in ssds," in *Proceedings of the 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2018, pp. 1–15.

[12] H. Kim, K. Han, and D. Shin, "Streamftl: Stream-level address translation scheme for memory constrained flash storage," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 617–620.

[13] B. S. Kim and S. L. Min, "Qos-aware flash memory controller," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2017, pp. 51–62.

[14] M. Bjørling, J. González, and P. Bonnet, "Lightnvm: The linux open-channel ssd subsystem." in *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST)*, 2017, pp. 359–374.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[16] J. Zhang, M. Kwon, D. Gouk, S. Koh, C. Lee, M. Alian, M. Chun, M. T. Kandemir, N. S. Kim, J. Kim *et al.*, "Flashshare: punching through server storage stack from kernel to firmware for ultra-low latency ssds," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 477–492.

[17] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu, "Mqsim: A framework for enabling realistic studies of modern multi-queue ssd devices," in *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST)*, 2018, pp. 49–66.

[18] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and C. Ren, "Exploring and exploiting the multilevel parallelism inside ssds for improved performance and endurance," *IEEE Transactions on Computers (TC)*, vol. 62, no. 6, pp. 1141–1155, 2012.

[19] J. Zhou, D. Han, J. Wang, X. Zhou, and C. Jiang, "A correlation-aware page-level ftl to exploit semantic links in workloads," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 30, no. 4, pp. 723–737, 2018.

[20] J.-U. Kang, J. Hyun, H. Maeng, and S. Cho, "The multi-streamed solid-state drive." in *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2014.

[21] J. Kim, D. Lee, and S. H. Noh, "Towards slo complying ssds through ops isolation," in *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST)*, 2015, pp. 183–189.

[22] N. Elyasi, M. Arjomand, A. Sivasubramaniam, M. T. Kandemir, C. R. Das, and M. Jung, "Exploiting intra-request slack to improve ssd performance," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 375–388, 2017.

[23] J. Guo, Y. Hu, B. Mao, and S. Wu, "Parallelism and garbage collection aware i/o scheduler with improved ssd performance," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 1184–1193.

[24] J. Zhang, J. Shu, and Y. Lu, "Parafs: A log-structured file system to exploit the internal parallelism of flash devices." in *USENIX Annual Technical Conference (USENIX ATC)*, 2016, pp. 87–100.

[25] J. Cui, W. Wu, X. Zhang, J. Huang, and Y. Wang, "Exploiting latency variation for access conflict reduction of nand flash memory," in *Proceedings of the 32nd Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2016, pp. 1–7.

[26] B. S. Kim, H. S. Yang, and S. L. Min, "Autossd: an autonomic ssd architecture," in *USENIX Annual Technical Conference (USENIX ATC)*, 2018, pp. 677–690.

[27] W. Kang, D. Shin, and S. Yoo, "Reinforcement learning-assisted garbage collection to mitigate long-tail latency in ssd," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 134, 2017.

[28] W. Kang and S. Yoo, "Dynamic management of key states for reinforcement learning-assisted garbage collection to reduce long tail latency in ssd," in *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[29] T. Zhang, Z. Zeng, and J. Li, "Reinforcement learning-driven address mapping and caching for flash-based remote sensing image processing," *Journal of Systems Architecture (JSA)*, 2019.