iCache: An Intelligent Cache Allocation Strategy for Multitenant in High-Performance Solid-State Disks

Donghua Li, Hui Sun, *Member, IEEE*, and Xiao Qin, *Senior Member, IEEE*

***Abstract*—Thanks to high-density flash memory and high parallelism, multitenant solid-state drives (MSSDs) have become a popular high-performance storage device for enhancing cache resource utilization and reducing operational costs within these SSDs. The competition for limited cache resources inside the MSSD among multiple tenants, however, can lead to performance interference among the tenants, and prior studies focused on quality of service (QoS) in MSSDs. An efficient caching scheme is crucial for optimizing SSD performance and lifetime. Existing caching schemes aim to shorten response time by the virtue of improved cache hit rates, which offer limited performance improvement as well as low cache resource efficiency. In this article, we propose an intelligent cache allocation scheme named iCache, which employs a long short-term memory (LSTM) model to capture the I/Os access patterns of workloads and dynamically allocates cache resources inside an MSSD according to maximum benefit point (MBP) and optimal allocation point (OAP). The extensive experimental results demonstrate that iCache reduces response time by up to 87%, 24%, and 20% compared against the existing caching schemes—Shared, Justitia, and MLCache, respectively. The empirical study confirms that the new traits of iCache immensely improve system performance by enhancing the cache efficiency of MSSDs and guaranteeing fairness in performance across varying workloads.**

***Index Terms*—Cache management, fairness, long short-term memory (LSTM), multitenant, performance optimization, solid-state drives.**

I. Introduction

MULTITENANT solid-state drives (MSSDs) have become a crucial device in modern storage systems [1], [2], [3]. It is challenging to ensure consistent quality of service (QoS) across tenants while maximizing cache utilization in systems [4], [5], [6]. However, the approach to sharing cache resources inside an MSSD for multiple tenants inevitably leads to performance interference and unfair resource allocation among these tenants, and the primary reason is threefold. First, tenants exhibit highly diverse access patterns, which complicate the prediction of cache resources [7]. Second, uncoordinated resource competition among tenants can result in unfair allocation of cache resources [8], [9]. Third, suboptimal resource allocation deteriorates resource utilization and overall system performance [10], [11]. Therefore, an effective caching scheme is of paramount importance for optimizing performance and fairness of cache resources for multiple tenants.

To promote the development of flash storage in commercial applications and address the shortcomings of traditional flash-based SSD cache management algorithms, Co-Active [12] addresses the issue of insufficient clean data for replacement in algorithms like CFLRU [13] under intensive write workloads by proactively writing back dirty data in the cache during idle chip periods. This ensures that almost 100% of the data replaced in the cache is clean, even under intensive write workloads, significantly reducing cache replacement overhead. LAC [14] enhances response speed and extends SSD lifespan through I/O intensity awareness, die-level load monitoring, and a hot data protection algorithm. However, these methods face challenges in addressing data cache contention issues in multiapplication environments.

With the proliferation of multiple tenants, conventional caching schemes struggle to balance performance optimization with fair resource allocation. Existing caching schemes, such as Shared and CostPI [15], enhance the performance of high-priority workloads. The Equal technique achieves fair cache allocation rather than responding to dynamically changing workloads. Justitia [1] effectively addressed performance disparities in scenarios with various access intensities in workloads, but its performance improvements are limited when the request intensities of workloads are similar. In addition, MLCache [2] employs neural networks to learn the reuse distance distribution of workloads, thereby improving cache hit rates. Nonetheless, a high cache hit rate does not always translate into reduced latency, especially when *performance cliff* is overlooked [16], [17]: in this case, a minor change in cache space tend to result in performance degradation.

In this article, we address the limitations of the existing caching schemes in multitenant SSD by proposing an intelligent cache partitioning method named *iCache*. Employing long short-term memory (LSTM) networks to analyze historical access patterns, iCache predicts future requirements of tenants in terms of maximum benefit point (MBP) and optimal

Received 25 June 2024; revised 11 September 2024; accepted 28 October 2024. Date of publication 6 November 2024; date of current version 23 April 2025. This work was supported in part by National Natural Science Foundation of China under Grant 62472002 and Grant 62072001. The work of Xiao Qin was supported in part by the U.S. National Science Foundation under Grant IIS-1618669 and Grant OAC-1642133; in part by the National Aeronautics and Space Administration under Grant 80NSSC20M0044; in part by the National Highway Traffic Safety Administration under Grant 451861-19158, and in part by the Wright Media, LLC under Grant 240250 and Grant 240311. This article was recommended by Associate Editor A. Kumar. *(Corresponding author: Hui Sun.)*

Donghua Li and Hui Sun are with the School of Computer Science and Technology, Anhui University, Hefei 230201, China (e-mail: sunhui@ahu.edu.cn).

Xiao Qin is with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849 USA (e-mail: xqin@auburn.edu).

Digital Object Identifier 10.1109/TCAD.2024.3492993

allocation point (OAP). To achieve fair allocation of cache resources, iCache is designed to meet the evolving demands of workloads by leveraging the time-series analysis capabilities of the LSTM model to accurately forecast future requirements of cache resources. The prediction model in iCache not only considers I/O intensity and the ratio of write requests but also integrates multidimensional information, such as request size and the uniqueness rate of access addresses, tailoring the resource allocation scheme to actual needs. Through accurate prediction of the MBP measures, iCache prevents performance degradation due to resource shortages, while identification of the OAP averts marginal performance gains that may occur from resource overallocation. Moreover, iCache incorporates a fairness decision-making mechanism anchored on MBP and OAP. When the total cache capacity falls short of the optimal requirements of all tenants, iCache adopts a proportional allocation mechanism to ensure that each tenant receives a fair cache resource based on its actual demands. This intelligent resource adjustment strategy optimizes resource utilization efficiency and ensures fairness in system performance, thereby addressing the cache management challenges in multitenant SSDs. The contributions of this article are tabulated as follows.

1) iCache leverages the robust time-series analysis capability of the LSTM model, integrating traditional I/O intensity, write request ratios, and multidimensional data, including request size and unique address rates. iCache, therefore, can tailor a more precise cache resource allocation scheme for each tenant.
2) To optimize resource allocation, iCache monitors and predicts MBP and OAP, implementing a dynamic mechanism that adjusts cache resources according to the immediate demands of workloads. This approach mitigates performance degradation resulting from resource mismatches and avoids unnecessary costs due to resource surplus.
3) The experimental results demonstrate that iCache reduces response times by up to 87%, outperforming the traditional I/O intensity-based Shared strategies. Under resource-constrained conditions, iCache's fairness decision mechanism equitably distributes cache to balance system performance across various workloads, thereby enhancing system fairness of cache resource allocation.

The remainder of this article is organized as follows. Section II presents the background of MSSDs and the machine learning techniques. Section III describes the motivation for our work and elaborates on the limitations of the existing approaches. Section IV sheds bright light on the iCache design. Section V provides the experimental setup and performance evaluation. Finally, Section VII summarizes the manuscript.

II. Background

Multitenant SSDs: Multitenant SSDs have been popularly used in storage systems in data centers [5]. Diverse I/O requests from multiple tenants lead to unpredictable performance and management complexity [18], [19]. Multitenant SSDs must handle these diverse I/O requests, which have various access patterns, request sizes, and I/O intensities. SSDKeeper [11] and Justitia [1] reveal that data layout and caching schemes in storage systems can be optimized by analyzing the features of tenants' I/O requests.

Cache Allocation in Multitenant SSDs: In multitenant SSDs, the cache partitioning problem is important, as it determines the performance and fairness of the system. An effective cache allocation scheme ensures equitable resource distribution among various tenants while simultaneously enhancing overall system performance and reducing response times. However, traditional cache allocation methods fail to satisfy both performance and fairness requirements. For example, a fixed-ratio cache allocation policy [15] may lead to resource wastage or performance bottlenecks for tenants, whereas a hit rate-based cache allocation policy [20] may cause an unfair allocation of resources. Therefore, it is challenging to balance the resource allocation among different tenants in order to guarantee performance. Liu et al. [2] proposed an ML-based cache allocation policy that dynamically adjusts cache resources according to tenants' performance requirements to improve resource allocation in multitenant environments.

Notably, machine-learning-based caching schemes have been popularly employed in multitenant SSDs [21], [22], [23]. Machine learning models can predict incoming request access patterns by learning the ones in tenants, which can achieve accurate cache resource allocation. In particular, LSTM can perform well in handling time-series prediction problems [24], [25]. These models can accurately predict the I/O access patterns in multiple tenants and dynamically adjust cache allocation to match changes in access patterns under workload [26], [27]. For example, SLAP [28] predicts an object's reuse time range using the LSTM model and admits objects that will be reused in the context of the current cache size.

Overall, cache partitioning in multitenant SSDs is an important issue. Practical machine learning techniques can be used to design a caching scheme for multitenant SSDs, considering performance and fairness of resource allocation.

III. Motivation

An effective caching scheme for multitenant SSDs is crucial for balancing performance and fairness. Traditional caching schemes often overlook *performance cliff* (performance sharply declines as resources are reduced) and *performance saturation* (performance gains diminish as resources increase.), leading to latency and fairness issues under certain workloads. In this article, we found that reducing the cache space for workloads with high locality did not significantly increase latency. Similarly, a high cache hit rate does not always correspond to a significant reduction in latency. These insights motivated us to develop an intelligent caching scheme, iCache, to optimize the performance and fairness of multitenant SSDs.

A. Imbalance Between Performance and Fairness

In multitenant SSDs, caching schemes critically influence overall system performance. As multiple tenants share the same cache resources, performance degradation and unfair resource allocation often occur. This competition not only

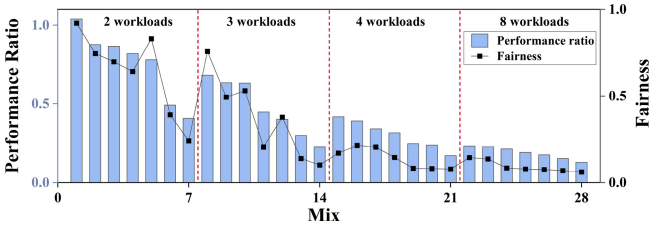


Fig. 1. We tested the PR and fairness of workload combinations of 2, 3, 4, and 8 under 28 different mix configurations.

affects the performance of individual tenants but also threatens system efficiency [29]. The Shared strategy overlooks the diversity of cache requirements under different workloads, leading to unfair resource distribution and increased uncertainty in workload performance. Allocating cache space dynamically according to a tenant's I/O access pattern is a challenging task. We employed the performance ratio (PR) in (1) to quantify the performance difference of a workload in an exclusive cache scenario versus a shared cache scenario, reflecting the impact of the Shared caching scheme on response time

$$\mathrm{PR}_i = \frac{\mathrm{RT}_i^{\mathrm{Alone}}}{\mathrm{RT}_i^{\mathrm{Shared}}} \times 100\% \tag{1}$$

where $\mathrm{RT}_i^{\mathrm{Alone}}$ represents the response time of the workload in the exclusive cache space, which RT_i^{Shared} denotes the response time of workload i when cache resources are shared among multiple workloads. If the shared caching scheme degrades performance, the PR is lower than 100%. Conversely, if the performance remains the same or improves, the PR is no lower than 100%

$$\mathrm{Fairness} = \frac{\min_i\{\mathrm{PR}_i\}}{\max_i\{\mathrm{PR}_i\}}. \tag{2}$$

Fairness is a metric used to assess the equity of PRs among different workloads, as shown in (2). A lower value of this index indicates lower fairness in the system. As shown in Fig. 1, when multiple workloads run concurrently, the average performance is significantly reduced compared to when executed individually. For example, under mixes 7, 14, and 21, performance decreases by 60%–80%. Furthermore, the Shared strategy struggles to ensure fairness across workloads, resulting in substantial performance disparities among workloads within the same mix.

To study the caching scheme in multitenant SSDs, it is critical to identify performance bottlenecks and quantify the sensitivity of workloads to cache configuration, with the aim of addressing issues of performance and fairness. Fig. 2 shows the nonlinear relationship between latency and cache space size. Notably, the *performance cliff* region is indicated by red circles. When a workload in a multitenant environment is allocated a cache space below the size indicated by the *performance cliff*, latency increases significantly, leading to performance unfairness among the tenants. The *performance saturation* phase, outlined by blue rectangles in the figure, reveals diminishing returns of performance gains during the cache allocation phase, where additional cache space does

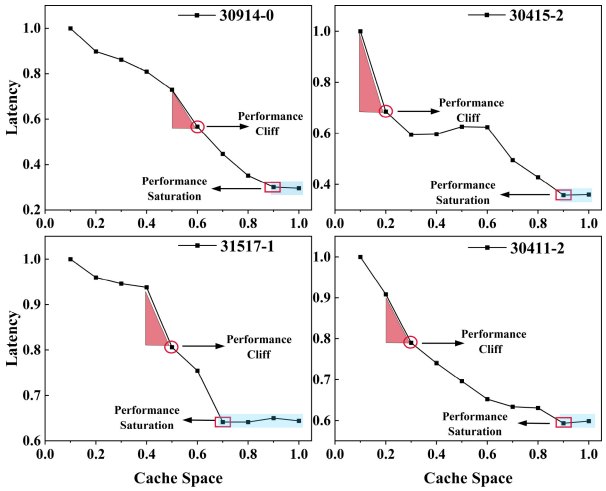
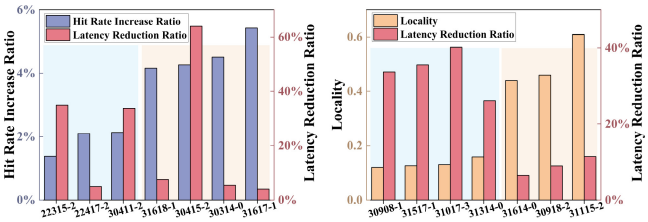


Fig. 2. Due to the presence of "performance cliff" and "performance saturation" points, there is nonlinear relationship between cache size and workload latency.



(a) (b)

Fig. 3. Increasing the cache space of the same size, workloads with high cache hit ratio gain, and workloads with high locality may not significantly reduce latency. (a) Correlation between hit ratio gain and latency. (b) Correlation between locality and latency.

not proportionately reduce latency. In multitenant SSDs, accurately predicting and managing cache sizes corresponding to *performance cliff* and *performance saturation* can significantly enhance performance and fairness (see Section IV-C for details).

B. Imbalance Between On-Demand and Available Cache Space

A high cache hit rate can effectively reduce the frequency of flash memory access within an SSD, thereby reducing latency. However, due to variations in workload access patterns, the latency improvement resulting from increased cache hit rates varies across workloads.

As illustrated in Fig. 3(a), adding the same amount of cache space for different workloads may yield similar cache hit rate increases; however, the ratio of latency reduction between workloads differs significantly. A high gain in cache hit rate does not always correlate directly with a significant decrease in latency.

Some workloads with smaller improvements in cache hit rates can present substantial reductions in latency. These findings suggest that a cache allocation scheme driven solely by enhancing cache hit rates does not address the issue of performance fairness in multitenant SSDs. Machine-learning models can accurately model workload characteristics and cache requirements, effectively addressing the difficulties

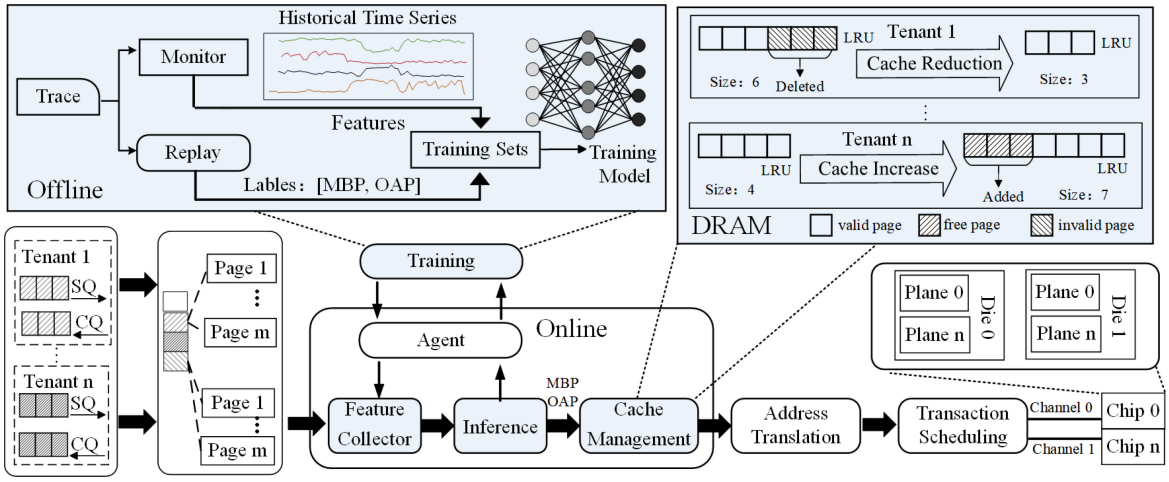


Fig. 4. Overview system of iCache. iCache primarily consists of three components: feature collection, model training and inference, and cache management. Notably, the model training component is conducted offline on the host side.

in latency perception inherent in traditional methods (see Section IV-D for details). In MSSDs, there is often a discrepancy between the cache space required by workloads and the allocated cache space.

Establishing a model that associates workload access patterns with cache requirements is crucial for improving workload performance and fairness in cache allocation.

Workload locality is considered a key criterion for cache allocation, with more cache resources typically allocated to high-locality workloads. However, this approach is not always optimal, as shown in Fig. 3(b). By adding equal cache space for different workloads, some low-locality workloads show more significant latency reductions than high-locality workloads. Hence, cache allocation schemes based solely on locality may not always lead to an optimal cache resource configuration. Allocating excess cache resources to high-locality workloads can exacerbate fairness issues in an MSSD.

For caching schemes in MSSDs, multiple workload characteristics should be considered, rather than relying on a single characteristic. This approach can significantly enhance the efficiency of cache allocation (see Section IV-B).

IV. iCache

A. iCache Architecture

iCache is an intelligent cache scheme designed for dynamic multitenant SSDs. It can intelligently allocate the cache resources within the MSSD to adapt to the varying I/O access patterns of each workload. iCache leverages three core components: 1) feature collection module; 2) an LSTM-based on-demand cache space prediction model; and 3) a dynamic cache allocation module. These components effectively address issues, such as insufficient latency awareness, performance degradation, and unfair resource allocation in traditional caching schemes.

In iCache, the feature collection module continuously captures I/O requests and workload characteristics that reflect the system's state in real time. Subsequently, the LSTM model predicts future cache demand, including the MBP and OAP, which determine the allocated cache resources. The model training phase is conducted offline on the host. As a result, the computational resources and time required during the training process have zero impact on the real-time performance of the SSD. In the prediction phase, the MBP- and OAP-based outputs from the model and the cache allocation module dynamically adjust their strategies to optimize cache resource allocation, reduce latency, and improve SSD performance. Through this approach, iCache enhances the performance of individual tenants and ensures fair cache space allocation among tenants. Fig. 4 presents iCache and its process, from workload monitoring and feature extraction to cache resource demand prediction and dynamic resource allocation. iCache maintains efficiency and fairness while adapting to complex applications.

B. Features for ML Model in iCache

1) Feature Analysis: The key to optimizing cache performance is understanding the interaction between workload characteristics and system performance. This section introduces the four critical workload characteristics used in the iCache machine-learning model: 1) write request ratio; 2) I/O intensity; 3) request size; and 4) unique address access rate. Extensive experiments were conducted using data traces generated by simulators and custom scripts (see Fig. 5) to investigate the specific impact of these workload features on response time. By employing a multifeature representation, we can accurately predict cache demands under various workloads and optimize the allocation of cache resources.

Write Request Proportion: Write request ratio is defined as the proportion of write operations among all requests in a given period. In Fig. 5(a), when other workload characteristics remain constant, as the write request ratio increases from 40% to 80%, the response time shows a significant increase. Write operations involve a complex data persistence process, including data updates and erasures, which incur higher latency than pure reads. Second, a larger cache tends to reduce response time because it can store more write I/Os, thereby decreasing the frequency of data writes to flash memory.

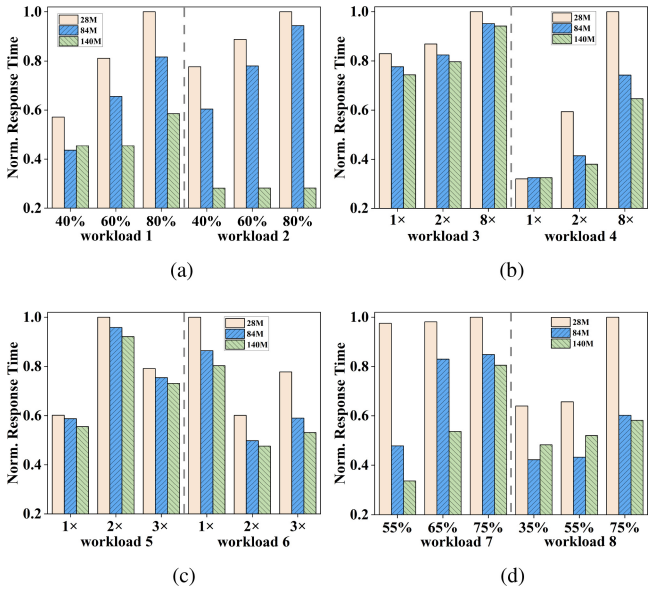


Fig. 5. Impact of different workload characteristics and varying cache sizes on workload response time. (a) Write request proportion. (b) Request size. (c) Request intensity. (d) Unique address access rate.

Request Size: Request size refers to the average access size of read and write requests in a period. Under different workloads, response time presents an upward trend as request size expands (request size increased by 2–8 times), see Fig. 5(b). A larger request size implies that each I/O operation involves a greater volume of data, which can increase data transmission time and total processing duration, thereby increasing response time. Second, increasing the cache space allows for the accommodation of more requests in workloads, thereby reducing the response time.

Request Intensity: Request intensity is defined as the total number of I/O requests per unit of time. As shown in Fig. 5(c), as request intensity increases by 1–2 times, there are two cases: in the first case, the response time increases with higher request intensity, and in another case, the response time decreases. When I/O request intensity surges significantly, the consequent rise in workload can potentially elongate the I/O queue, thereby increasing response time. However, modern storage systems can process data in parallel, enabling them to handle multiple I/O requests concurrently. An increase in request intensity can leverage this parallel processing mechanism, enhancing overall throughput and mitigating response time to a certain extent. Moreover, the results indicate that a larger cache capacity reduces response time, a particularly pronounced trend.

Unique Address Access Rate: The unique address access rate is defined as the ratio of the number of distinct requests to the total number of requests under a given workload. In Fig. 5(d), if the other workload characteristics remain constant, the large unique address access rate (by 10% and 20%) can curb response time. This is because a high rate of unique address accesses leads to frequent updates of cache data, thereby reducing the cache hit rate and increasing access latency. Additionally, a larger cache capacity can mitigate the problem of low cache hit rates caused by a high rate of unique address access.

2) Interaction of Multiple Features: The response times were differently sensitive to the proportion of write requests under different workloads. For instance, with a 28 MB cache, reducing the proportion of write requests decreased the response time by an average of 20% under workload 1, while the response time only decreased by an average of 11% under workload 2 [Fig. 5(a)]. Additionally, even with the same proportion of write requests, workloads exhibited differences in the reduction of response time with increasing cache capacity of the same size. This reveals that other workload features also significantly impact cache space allocation in addition to the proportion of write requests.

Under identical cache capacities, the impact of varying request sizes differs across different workloads. With a 28-MB cache, increasing the request size for workload 3 results in an average response time increase of 10%, whereas for workload 4, the increase exceeds 30% [Fig. 5(b)]. Furthermore, keeping the request size constant and augmenting the cache capacity by the same amount results in an average response time reduction of 4% for workload 3, while workload 4 experiences nearly no change. This indicates that, in addition to request size, other workload characteristics also influence cache allocation efficiency.

Under a fixed cache capacity, the impact of adjusting request intensity varies across different workloads. This variation stems primarily from distinct access patterns and data localities of the workload. Request intensity can affect the cache hit rate and response time. In addition, for workloads with the same request intensity, the response time changes induced by increasing the cache capacity of the same size show an inconsistent trend [Fig. 5(c)]. This observation suggests that other workload features significantly affect cache performance besides cache capacity and request intensity.

For the same cache capacity, changing the unique address rate affects different workloads to varying degrees. For workloads with similar unique address rates, the reduction in response time differed when the same size cache capacity was increased [Fig. 5(d)]. Thus, response time improvement is jointly affected by multiple features instead of a single feature, such as the unique address ratio. Thus, the response time should be comprehensively optimized based on multiple features in cache resource management and optimization decisions.

Given that the efficiency of cache allocation is influenced by a composite of workload features, the impact of increasing or decreasing the cache space varies across different workloads. The access patterns of the workloads exhibit complex temporal dependencies and nonlinear traits, indicating that cache allocation must be dynamic rather than static in response to workload fluctuations. iCache employs neural networks to comprehensively consider the impact of multiple workload characteristics on cache allocation, thereby addressing the challenges of cache allocation under multifeature scenarios.

3) Feature Collection: To implement iCache in MSSDs, we propose a feature acquisition framework to accurately obtain the features of the workloads. This framework provides decision-making capabilities for the dynamic allocation of

cache space to optimize access latency and improve resource utilization.

The host interface logic (HIL) module divides host-side requests into page-sized subrequests. iCache collects statistics on the total number of requests, the number of write requests, the request sizes, and the number of unique addresses within each time interval. The feature collection module then extracts features from these subrequests. The collected information is used to construct the training dataset in the host and to serve as input features in the inference phase. These features affect the prediction of future cache space requirements, which is critical for improving MSSD performance. iCache collects four features—write ratio, request size, request intensity, and unique address rate—as inputs to the machine-learning model.

4) Feature Calculation: The access patterns in the workload exhibit fluctuations over long periods but display relative stability on a short-term scale. For example, the Systor'17 trace is characterized by high I/O intensity and variations in request intensity. When selecting the feature sampling interval, it is crucial to balance capturing correlation information and minimizing computational overhead. A small sampling interval causes the model to process a large amount of irrelevant data, introducing excessive noise that prevents iCache from extracting useful information. This not only increases computational complexity but also hinders prediction accuracy. Conversely, if the sampling interval is too large, important short-term dynamics and correlations among requests may be missed, making it challenging for the LSTM—relying on time-series information—to accurately capture patterns in the sequences. iCache performs a feature sampling operation per second, which effectively captures dynamic changes in cache demand while preserving correlation information among requests for accurate model predictions.

I/O intensity is defined as the total number of requests during a period, revealing the active level of the workload. We have Intensity $= \sum$ RequestNum. The write ratio, which indicates the proportion of write requests to total I/O requests in a period, is represented as WriteRatio $=$ ($\sum$ WriteNum/Intensity). This metric reveals the distribution of read and write requests. Request size refers to the average data amount per request in a period, reflecting the data intensity of the requests, calculated as RequestSize $=$ ($\sum$ RequestSize/Intensity). The unique address ratio is the ratio of addresses that appear only once to the total number of addresses in a period, calculated as UniqueRatio $=$ ($\sum$ UniqueNum/Intensity).

C. Label

1) Definitions of MBP and OAP: In iCache, we studied the impact of cache capacity on response time and identified two critical operational points: 1) the MBP and 2) the OAP. In Fig. 6, the response time decreases as the cache size increases within the range [S_1, S_2]. When the cache size is increased from S_2 to S_3, the response time t is significantly reduced, marking the point of maximum performance enhancement, S_3 defined as the MBP. As the cache space continues to increase beyond the S_4, the reduction in response time becomes negligible, and thus S_4 is defined as the OAP.

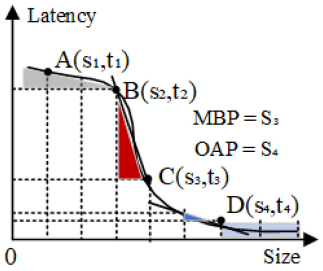


Fig. 6. Modeling of MBP and OAP. The cache size corresponding to MBP is equal to S_3, and OAP is equal to S_4.

The MBP significantly increases the complexity of cache allocation in MSSDs. First, if the cache space allocated to a workload is less than the MBP, the performance of the MSSD tends to be suboptimal. Second, minor fluctuations in cache capacity near the MBP can cause drastic performance variability, making it challenging to guarantee consistent performance under workloads. Therefore, balancing the cache capacity for each tenant to approach or reach the MBP is crucial for achieving optimal cache space utilization. The determination of the OAP indicates that the marginal benefits of increasing cache space become minimal. Once the OAP is reached, further expansion of the cache space cannot enhance performance; therefore, timely cessation of cache expansion is important to avoid underutilization of cache resources.

2) Determination of MBP and OAP: We propose advanced neural network techniques for cache resource allocation to achieve the potential benefits derived from the MBP and OAP in MSSDs. This approach enables the precise prediction and matching of cache demands for various workloads, thereby optimizing overall system performance

$$K = \frac{\Delta\text{Latency}}{\Delta\text{Size}}. \tag{3}$$

To precisely identify the MBP and OAP for workloads within each time segment, we designed a comprehensive set of experiments to study performance variations. We measured the impact of different cache sizes on response time within the same period. The cache size increments were set at 10% of the total cache capacity. Data were collected using the MQSim simulator. As shown in (3), we defined the performance gain ratio K to quantify the impact of increased cache capacity on response time, where $\Delta\text{Latency} = \text{Latency}_1 - \text{Latency}_2$ represents the reduction in response latency due to increased cache space, $\Delta\text{Size} = \text{Size}_2 - \text{Size}_1$ is the increment in cache size ($\text{Size}_2 > \text{Size}_1$). By analyzing all computed K values, we determined the MBP to be the maximum value among all K values

$$\text{MBP} = \text{Size}_i \text{ where } K_i = \max\{K_1, K_2, \ldots, K_n\}. \tag{4}$$

For the determination of OAP, we set a performance gain threshold L. If the performance gain K falls below L, the corresponding cache size is considered to be the OAP

$$\text{OAP} = \min\{\text{Size}_i \mid K_i < L \text{ and } \text{Size}_i > \text{MBP}\} \tag{5}$$

where L is set at 5% of performance improvement. This precise measurement provides key training labels for using LSTM models to allocate cache resources dynamically. This enables the model to more accurately predict future cache demands and

optimize cache configuration strategies, thereby significantly enhancing system performance and cache resource utilization.

D. Model Selection

1) Properties of LSTM Model: In MSSDs, the cache must serve requests from diverse data streams that exhibit various access patterns, such as I/O intensity and write request ratios. Static cache allocation schemes and priority-based approaches often fail to adapt to dynamic changes in the I/Os access patterns of workloads, leading to wasted cache resources and degraded performance. Machine learning and deep learning models have demonstrated outstanding capabilities in handling time-series predictions. LSTM networks are particularly effective due to their sequence modeling capabilities, gating mechanisms, and adaptive learning abilities.

2) Application of LSTM Model in Cache Allocation: In an MSSD, the exceptional ability of LSTM to model sequential data enables it to accurately identify the I/Os access patterns of different workloads. By analyzing historical I/Os access patterns, LSTM can predict the MBP and OAP over time, which is critical for dynamically adjusting the distribution of cache space for different workloads. Furthermore, the gating mechanism of LSTM provides adaptability to changes in workload by adjusting the network weights to accommodate new workloads, thereby maintaining low latency and high performance. Additionally, LSTM's adaptive learning capability can integrate multidimensional features, enhancing the accuracy of predicting cache demands for each workload. This comprehensive analytical capability is important for optimizing cache allocation and enhancing overall system performance in multitenant environments. Therefore, LSTM is used to predict cache demands, improving the efficiency of cache management and significantly enhancing the overall performance of MSSDs.

E. Model Training and Inference

1) Training: The LSTM model training for allocating cache resources in MSSDs includes several steps. Initially, a feature collection model periodically extracts features from the workload to form feature data in the training set. Subsequently, we measured response times for different cache sizes to determine the MBP and OAP for each period. These features and labels constitute the training dataset. Before training the LSTM model, we preprocessed the raw data collected from the SSD, including feature selection and data serialization, to meet the requirements of the LSTM model. For the LSTM network architecture, we selected a single-layer network comprising 128 neurons. The mean-squared error (MSE) was used as the loss function, and the Adam optimizer was employed to train the model. This configuration effectively trains the network to handle dynamically changing data streams and cache demands. During the model training phase, network performance is optimized through the iterative execution of forward propagation, loss computation, backward propagation, and weight updates. Upon completion of training, the model's prediction accuracy was validated on an independent test set. Maintaining the accuracy and adaptability of the model is crucial, requiring regular optimizations and parameter adjustments based on performance feedback and new data input. This dynamic adjustment strategy ensures that the model provides accurate cache allocation recommendations for multiple applications.

iCache configures a model for each individual workload, aiming to learn its unique access patterns to predict the MBP and OAP for future cache demands. By utilizing multidimensional feature analysis to accurately identify and learn the access patterns of different workloads, iCache can detect the dynamic changes in workloads. iCache applies the predicted MBP and OAP to dynamically adjust cache resources to accommodate changing workload demands. Additionally, iCache trains the model during an initial phase and periodically updates the model to reflect fluctuations in the workload. This periodic update mechanism enables the model effectively to adapt to new access patterns and demands, maintaining its effectiveness in dynamically changing workloads.

2) Inference: The trained LSTM model was integrated into the SSD cache management module during the inference stage. This model processes sequences of requests, including write ratio, request size, request intensity, and unique address rate in real time.

Using the I/Os patterns learned during the training stage, iCache predicts the future MBP and OAP. Based on these predictions, the cache allocation module dynamically adjusts the cache allocation to meet real-time workload demands. Leveraging the advantages of the LSTM model in time-series prediction and adaptation to new data, iCache manages cache space using a proactive adjustment approach to enhance performance and cache efficiency.

The LSTM model analyzes workloads and updates its predictions at each predefined time step, ensuring that the cache allocation scheme aligns with the future I/Os access patterns of the workload. The length of the historical period detected by the LSTM can be adjusted by modifying the unique time-step length of the LSTM model. In the training and inference stages, the time step length of the model is a critical parameter, as shown in Fig. 7, determining the window size of the historical data. The selection of the time-step length depends on the features of the workloads and the prediction time range. In cache allocation, a shorter time step can quickly respond to workload changes; however, frequent adjustments of the cache space may disrupt the locality of the workload. A longer time step can help the model capture longer-term dependencies but also increase the computational load of model inference and lengthen the prediction time. We tested the effects of different time-step lengths on cache efficiency, model computational efficiency, and prediction error, and found that a time-step length of 180 offers the best balance among these three performances. This setting allows the model to perceive changes in workload in a timely manner, thereby enhancing the efficiency of cache allocation while maintaining low prediction error and inference time.

In addition, we demonstrate the model update procedure as follows. First, the model training is conducted offline on the host, ensuring that the computational resources and time required during training have no impact on the real-time performance of the SSD and the model inference. Once the

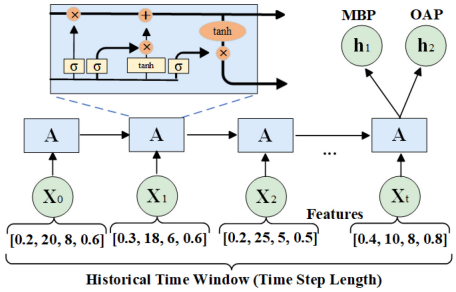


Fig. 7. Inference process of LSTM. The LSTM model predicts future MBP and OAP by observing workload characteristics over a past period, such as write ratio, IO intensity, request size, and uniqueness rate.

model training is completed, the host offloads the model to the SSD firmware via an agent module. After the firmware invokes the model, the trained model will work in full swing in the MSSD. Second, to adapt to the dynamic changes in workloads, the LSTM model should be updated daily because VDI workloads exhibit a daily cycle with the lowest request volume occurring between 00:00 A.M. and 5:00 A.M., the period of which is chosen for offline model updates to avoid interfering with normal operations. During the update process, the modified model—being offloaded to the SSD firmware via an agent module—can promptly guide the prediction of cache resources for the next day, thereby enhancing overall system performance. Regarding the cost of model updates, the model size is less than 100 kB, and the transmission time on the high-bandwidth PCIe 3.0 (985 MB/s) and SATA 3.0 (600 MB/s) interfaces is almost negligible with little impact on MSSD performance. This strategy improves the efficiency of model updates and maintains system performance.

F. Dynamic Cache Allocation Strategy

In MSSD, it is crucial to dynamically adjust cache resources to accommodate workload variability. iCache leverages an LSTM model to predict the future cache demands of workloads precisely. By analyzing historical data, including metrics, such as write request ratio, request size, request intensity, and unique address rate, iCache can forecast the MBP and OAP for each workload over incoming periods.

The dynamic cache allocation module monitors and compares the current cache size ($\text{Size}_{\text{current}}$) of a workload with its OAP, adjusting the cache resources accordingly. When a workload's $\text{Size}_{\text{current}}$ exceeds its OAP, the cache resource usage has surpassed the optimal configuration. In this case, iCache updates the workload status to a *Supply* state, where it initiates a smart write-back strategy. This strategy starts from the least recently used (LRU) end of the list, eliminating the cache space exceeding the OAP. Read data are deleted, while write data are written back to flash memory. Each data eviction increases the free cache space. Conversely, if a workload's $\text{Size}_{\text{current}}$ is below the OAP, iCache updates its status to a *Demand* state, indicating insufficient cache resources. If free space is available, iCache increases the target cache resources for that workload. When a workload's $\text{Size}_{\text{current}}$ matches the OAP, the cache resource allocation reaches an ideal state, and the workload lies in a *Stable* state, ensuring optimal resource configuration.

Algorithm 1 LSTM-Guided Cache Allocation

Require: *WorkloadRequests*, *TotalCacheCapacity*
Ensure: *CacheAllocation*

```
1: while true do
2:     Data ← Get_data(Workloads)
3:     for all w in Workloads do
4:         w.MBP, w.OAP ← LSTM.Predict(Data[w])
5:     end for
6:     SumOAP ← Sum(w.OAP for w in Workloads)
7:     SumMBP ← Sum(w.MBP for w in Workloads)
8:     if SumOAP ≤ CacheCapacity then
9:         for all w in Workloads do
10:            Allo[w] ← w.OAP
11:        end for
12:    else if SumMBP ≤ CacheCapacity then
13:        for all w in Workloads do
14:            Allo[w] ← w.MBP
15:        end for
16:        Extra ← CacheCapacity - SumMBP
17:        for all w in Workloads do
18:            Allo[w] ← Allo[w] + Extra × (w.OAP / SumOAP)
19:        end for
20:    else
21:        for all w in Workloads do
22:            Allo[w] ← CacheCapacity × (w.MBP / SumMBP)
23:        end for
24:    end if
25:    Apply(Allocation)
26:    Wait_next()
27: end while
```

In MSSDs, iCache dynamically adjusts cache resources based on the needs of each tenant. This adjustment is predicated on the predictive capability of the LSTM model, which accurately anticipates changes in cache space requirements for each workload. When multiple tenants compete for cache resources, the cache resources are categorized into three states, namely, resource-abundant state, resource-strained state, and resource-constrained state. The following illustrates how the iCache system implements targeted dynamic cache resource allocation strategies based on the specific cache space requirements of each workload (see Algorithm 1).

1) Resource-Abundant State: When the sum of all workloads' OAPs does not exceed the total cache capacity, it indicates that cache resources are plentiful. iCache allocates cache resources to each workload according to its OAP, as predicted by the LSTM model. This precise matching strategy maximizes system performance, as each workload's resource requirement is met. Additionally, for workloads that overconsume resources (i.e., $\text{Size}_{\text{current}}$ is larger than OAP), iCache reduces their cache allocation and reallocates the resources to under-resourced workloads (i.e., $\text{Size}_{\text{current}}$ is smaller than OAP), thereby maintaining overall system efficiency and balance (lines 6–11).

2) Resource-Strained State: When the sum of the OAPs of all workloads exceeds the total cache capacity, but the sum of the MBPs remains within the total cache capacity, it indicates that achieving optimal resource allocation for each workload will be challenging due to limited cache resources. In this scenario, iCache first allocates the minimum necessary resources to each workload up to its MBP to maintain

basic operational efficiency. Subsequently, the remaining cache resources are distributed proportionally based on the OAP ratios of the workloads. This strategy not only prevents significant performance degradation due to insufficient resources but also enhances the performance of each workload as much as possible within the limits of the available resources (see lines 12–19).

3) Resource-Constrained State: When the total demand for MBP by all workloads exceeds the overall cache capacity, iCache adopts a balanced allocation strategy, distributing the limited resources according to the urgency level (i.e., proportion of MBP) of each workload. This strategy ensures that, in the case of severe resource limitations, each workload receives a fair share of resources based on its importance and demand, thereby maximizing the utility of resources (see lines 21–23).

In MSSDs, each workload must receive a minimum resource allocation. Even without active requests, iCache reserves a minimal cache space for each workload. This is crucial because a lack of cache resources can lead to sudden traffic surges, significantly increasing response time. By maintaining a baseline cache space for each workload, iCache effectively prevents severe degradation in service quality due to rapid increments in response time. Although this strategy can compromise optimization effectiveness, it plays a vital role in maintaining the QoS and ensuring fundamental fairness among different workloads. This balance is a necessary adjustment for resource allocation, aimed at system stability and adaptivity under various workloads.

Through this precise cache management method, iCache can adapt to different states of resource competition, effectively enhancing cache utilization. This is particularly evident in multitenant environments, where iCache demonstrates exceptional scheduling and optimization capabilities.

G. Overhead of iCache

Space Overhead: For each workload, iCache uses three counters to count the number of write requests, the total number of requests, and the request size, respectively, which result in negligible overhead. In addition, the access frequency of unique addresses is counted using key–value pairs. For a load of 2 million requests per hour, the statistics are updated every second. Assuming each address uses 2 bytes of storage and 1 byte is used to store the frequency of address accesses, this requires no more than 2 kB of storage space.

Time Overhead: Most flash memory controllers that utilize NVMe 1.3 or higher versions are based on the Cortex R8, which can achieve a 1.5-GHz clock frequency and 3.77 DMIPS/MHz [30]. For a neural network with an input size of n equal to 4, hidden layer neurons m equal 128, a sequence length of t equals 180, and two output units, the theoretical prediction operation involves $n \times (n \times m + m^2 + m) \times t$ floating-point operations. When we run iCache on the Cortex R8 platform, the estimated time required is approximately 3 ms. Given that the neural network is trained offline, predictions are performed regularly, even when operating online. If predictions occur every 3 min with several hundred thousand requests per minute, the computation time required for the neural network is negligible per request delay.

TABLE I
SIMULATION PARAMETERS IN MQSIM

SSD Organization	Host Interface: PCIe 3.0 (NVMe 1.2) Capacity: 512 GB, 8 channels, 4 chips/channel
Flash Interface	ONFI 3.1 (NVDDR2)
Flash Structural	256 pages/block, 2048 blocks/plane 2 planes/die, 2 dies/chip, 8 KB/page
Flash Latency	Read: 75 μs, Program: 750 μs, Erase: 3.8 ms
FTL	Mapping: DFTL [32], TSU: Sprinkler [33] Over provisioning Ratio: 0.07

TABLE II
INFORMATION OF THE MIXED TRACES

2 Workloads		3 Workloads		4 Workloads		8 Workloads	
Mix1	31517-1 31617-1	**Mix8**	30411-2 30918-2 22114-2	**Mix15**	21615-0 31314-0 22114-3 31614-0	**Mix22**	31017-3,31816-0 30918-2,31614-0 22417-2,22114-3 30314-0,31719-0
Mix2	30818-2 22114-3	**Mix9**	30415-2 30818-2 31719-0	**Mix16**	22215-2 30918-2 22417-2 31719-0	**Mix23**	31618-1,31816-0 30918-2,31614-0 22417-2,22114-3 30314-0,31719-0
Mix3	30818-2 31719-0	**Mix10**	30818-2 22114-3 22417-2	**Mix17**	30411-2 22417-2 31719-0 30908-1	**Mix24**	30415-2,31816-0 30918-2,31614-0 30914-0,22114-3 30314-0,31719-0
Mix4	31017-3 22114-3	**Mix11**	31517-1 22417-2 31719-0	**Mix18**	31017-3 22114-3 31614-0 31015-2	**Mix25**	30818-2,30918-2 31614-2,30415-2 31015-2,31719-0 30314-0,22114-3
Mix5	31017-3 30314-0	**Mix12**	30817-2 22417-2 31614-0	**Mix19**	30818-2 22114-3 22417-2 31015-2	**Mix26**	30908-1,31517-1 22215-2,30415-2 31015-2,31719-0 30314-0,22114-3
Mix6	30411-2 31015-2	**Mix13**	30415-2 31719-0 30314-0	**Mix20**	30415-2 22114-3 22417-2 31015-2	**Mix27**	22417-2,31618-1 31816-0,30415-2 31015-2,31719-0 30314-0,22114-3
Mix7	31517-1 31115-2	**Mix14**	22315-2 22114-3 31015-2	**Mix21**	30415-2 30411-2 22417-2 31015-2	**Mix28**	22215-2,30918-2 31614-0,30415-2 31015-2,31517-1 30314-0,22114-3

V. EXPERIMENT

To study the performance of iCache, we conducted a series of tests to explore its impact on critical performance under various mixed workloads. Specifically, we focused on max slowdown, response time, fairness, tail latency, and hit ratio, which are essential for the efficiency of caching schemes.

A. Setting

We implemented iCache in MQSim [33], an open-source multiqueue SSD simulator. The cache size is set to 256 MB. The configuration parameters for MQSim are listed in Table I. To validate iCache, We utilized real enterprise-scale Systor'17 [34] traces collected from Fujitsu Labs in Japan. We randomly selected 24 workloads to create 28 distinct mixed workloads (see Table II). To clearly describe the trace format, we retained the month, day, hour, and storage volume number information in each trace of the VDI workloads. For example, we renamed "2016031517-LUN1" to "31517-1." To simulate multiple tenants sharing the same solid-state drive device, we configured each workload to start competing for cache resources from the same initial moment. Alternative caching schemes in this article include Shared, Equal, CostPI, Justitia, MLCache, and OPT (ideal optimal cache partition scheme). The results demonstrate that our proposed iCache significantly enhances system performance compared to the alternatives.

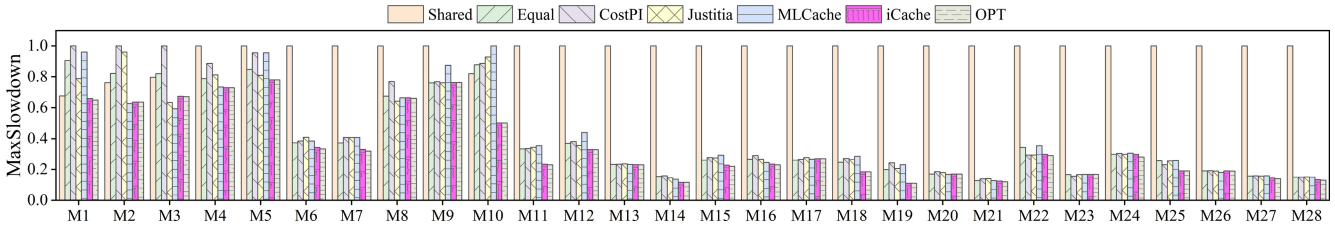


Fig. 8. Max slowdown of iCache versus other cache allocation policies for different workload combinations and different numbers of workloads. Lower max slowdown indicates better performance.

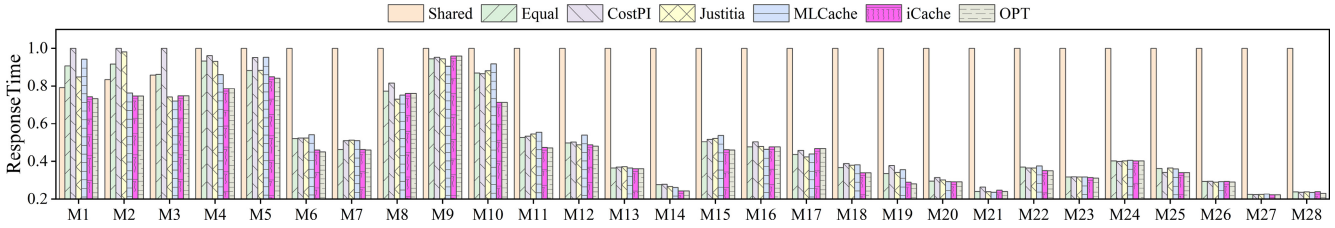


Fig. 9. Response time comparison of iCache with other cache allocation policies under different workload combinations and different numbers of workloads. Lower response times indicate better performance.

B. Evaluation Metrics

1) Max Slowdown: Max Slowdown quantifies the performance degradation (latency amplification) of the most affected workloads in the mix. In Fig. 8, the Shared scheme exhibits the highest max slowdown values, whereas iCache maintains the lowest levels. Existing cache allocation methods fail to circumvent the *performance cliff*. iCache predicts and allocates cache space no less than the size of MBP for each workload, avoiding a precipitous performance drop-off under workloads. In a mixed scenario for two workloads, compared with Shared, Equal, CostPI, Justitia, and MLCache, iCache achieved an average max slowdown reduction of 42%, 16%, 23%, 15%, and 14%. The highest reduction is 67% (Mix7), 25% (Mix1), 37% (Mix2), 33% (Mix2), and 31% (Mix1), respectively. iCache makes the performance of seven mixed workloads closely aligned with the standalone execution, with only around 10% performance loss. When the number of workloads increases, the cache space allocated to each workload inevitably diminishes, affecting the efficiency of cache distribution. For mixed workloads with 3, 4, and 8 tenants, iCache reduced the Max Slowdown by approximately 22%, 19%, and 10%, respectively, compared with MLCache, with the maximum reductions being 12% (Mix11), 12% (Mix19), and 6% (Mix25).

2) Response Time: Traditional Shared cache allocation and Equal cache distribution schemes do not consider the access patterns of workloads, thereby limiting their capability to reduce the response time of workloads. As shown in Fig. 9, iCache reduces response time by up to 75% (Mix27) compared with shared cache allocation schemes. Under Mix1, Mix2, and Mix4, iCache achieves an average reduction of about 15% in the response time compared with the Equal scheme. CostPI attempts to reduce response time by allocating dedicated cache space to latency-sensitive workloads; however, when the caching scheme cannot be aware of the latency sensitivity of workloads, severe performance degradation of latency-sensitive workloads under Mix3. Justitia addresses the issue of unfair cache distribution caused by different access intensities in workload. However, if the differences in access intensity among workloads are minimal, the response time is similar to that under the Equal scheme. MLCache allocates data cache space based on the reuse distance of write requests, but this approach has limitations in managing read–write cache. Optimizing workload response time by establishing a relationship between reuse distance and cache hit rate is sometimes ineffective. As shown in Fig. 3, a high hit rate gain does not always translate to significant response time optimization. Compared with MLCache, iCache reduces the response time of each workload by 4% on average. Under Mix10, the response time reduction under each workload reaches 20%.

3) Fairness: In a multiworkload environment, fairness refers to the ratio between the worst and best-performing workload performance. iCache allocates cache space based on each workload's MBP and OAP values. iCache avoids insufficient cache allocation and severe cache saturation, ensuring high fairness for each workload. As shown in Fig. 10, under Mix7, 10, 11, and 19, iCache improves in the fairness significantly. Under Mix4 and Mix3, the fairness performance of iCache is relatively average. Dynamic cache allocation sometimes results in mixed workloads performing better than when executed individually. For instance, iCache reduced the response time by 17% under workload 22114-3 in Mix4 and by 14% under 31719-0 in Mix3 compared to the standalone execution. A dynamic allocation method proactively evicts cache pages. When the evicted pages will not be accessed in the future, this benefits workload performance, albeit at the expense of fairness.

4) Tail Latency: Tail latency is a crucial metric for evaluating system performance under high load or extreme conditions, particularly in applications requiring high stability and rapid response. As shown in Fig. 11, iCache reduces the latency

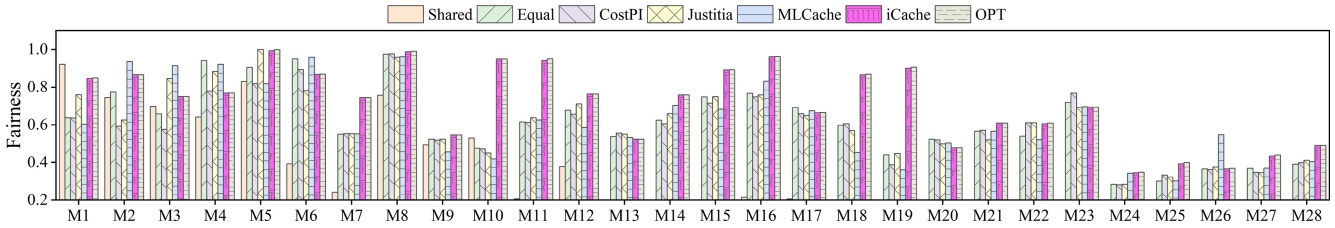


Fig. 10. Fairness of iCache versus other cache allocation policies under different workload combinations and different numbers of workloads. Greater fairness indicates better performance.

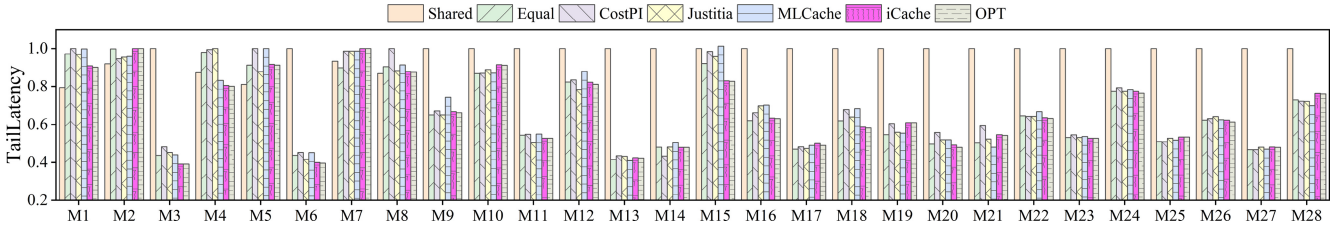


Fig. 11. Tail latency of iCache versus other cache allocation policies for different workload combinations and different numbers of workloads. Lower maximum response latency indicates better performance.

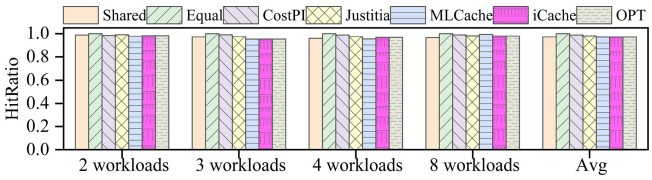


Fig. 12. Cache hit rates for iCache versus other cache allocation schemes under mixed workloads and different numbers of workloads.

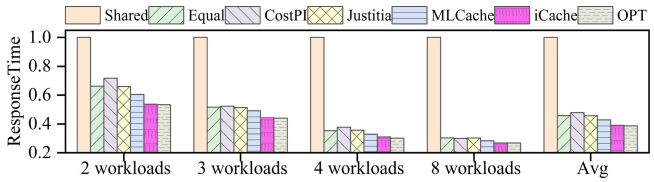


Fig. 13. Response time for different workload combinations with 60M cache capacity.

under each workload by an average of 7% compared with Shared schemes. Under two workloads, it achieves an average latency reduction of 46% per workload, with a peak reduction reaching 60% under Mix3. Compared with the MLCache, iCache brings about an average tail latency reduction of 3% for a tenant workload and a 6% decrease in latency for a tenant under two mix workloads. These results indicate that iCache significantly improves the system performance. iCache exhibits lower tail latency in most workload combinations. In some scenarios such as Mix2, however, the tail latency is not the lowest due to the dynamic changes in workload access patterns resulting in fluctuating cache demands. During the dynamic cache allocation process, iCache proactively evicts dirty data. This intentional eviction can temporarily enlarge queue wait time, thereby leading to an elevated tail latency in the short term.

5) Cache Hit Rate: In MSSDs, the performance is not merely constrained by data access locality. Due to the asymmetry of read and write operations and the diversity of request sizes, a high cache hit rate does not directly introduce a low response time. As shown in Fig. 12, the average cache hit rate of iCache is slightly lower than that of the Equal strategy by less than 3%. Although the Equal strategy ensures fairness in cache allocation and reduces page replacement operations, leading to a higher cache hit rate, previous experiments have shown that the strategy struggles to guarantee response times due to its failure to adequately consider the dynamic changes in workloads. On the other hand, MLCache enhances the cache hit rate by utilizing histograms of reuse distances for write requests, but merely adjusting cache space based on locality characteristics in a multitenant environment is not always effective, and furthermore, MLCache is not suitable for read–write caching. In contrast, iCache employs an LSTM model to describe the relationship between workload and response time and uses OAP and MBP to guide resource allocation, thereby effectively enhancing both the response speed and fairness of the workload.

C. Extended Experiment

1) Small Cache Performance: We validated the portability of the iCache strategy. Fig. 13 presents the response time of caching schemes under 60M small cache capacity. We find that iCache achieves a significant reduction in response time under all workload mixes. Specifically, under scenarios with two and four workloads, iCache reduced the average response time for each workload by approximately 10% and 8%, respectively, compared to MLCache. These results indicate that iCache can effectively enhance system response time under small-sized cache capacity constraints through its optimization mechanisms, demonstrating its adaptability and superiority in multiple mixed workloads.

2) Cache Granularity and Frequency Effects: In the process of dynamically adjusting cache allocation, the selection of adjustment frequency critically affects the system's ability to respond to workload changes in a timely manner. The Systor'17 trace is characterized by high I/O intensity and

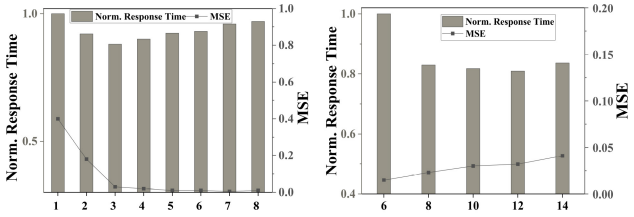


Fig. 14. Performance of iCache for different cache partitioning frequencies and granularities. (a) Cache adjustment frequency. (b) Cache allocation granularity.

frequent burst requests. Fig. 14(a) illustrates the performance of iCache as the cache partitioning frequency f varies from 1 to 10 min. The results indicate that neither high-frequent adjustments (i.e., lower f values) nor low-frequent adjustments (i.e., higher f values) for cache space can fully exploit the optimal performance of the system. Frequent adjustments to the cache space allocated for workloads can increase the computational overhead of model inference and the cost of cache replacement. Conversely, infrequent dynamic adjustments may fail to promptly adapt to workload changes, thereby diminishing the performance of dynamic resource allocation. A comprehensive analysis identified a 3-min interval as the optimal frequency for cache partitioning adjustments. This frequency ensures the system's sensitivity to workload variations while avoiding the additional costs associated with high-frequent adjustments. As the cache partitioning frequency decreases, the number of predictions diminishes, allowing the LSTM model to accumulate more historical data. This trend leads to smoother average predicted values and a reduction in MSE. However, this smoothing effect may overlook the dynamic fluctuations in cache demand caused by workload variability, making it challenging to capture short-term changes and affecting the accuracy of resource allocation.

iCache guides cache space allocation by predicting the future cache size requirements for each workload. To enhance prediction accuracy, iCache allocates cache space based on the predicted future cache space ratio. The actual required cache capacity is equal to the cache allocation ratio multiplied by the total cache size. The setting of cache granularity, or the number of divisions within the cache, affects the accuracy of the model's predictions and subsequently the performance of cache allocation. While finer ratio divisions can improve the management granularity of cache allocation, they also increase the complexity of model predictions and may yield limited performance improvements. Fig. 14(b) shows the performance of iCache under different cache partitioning granularities. The overall performance of iCache increases with the increment of the granularity parameter g. A larger cache granularity requires the model to process more partition data, increasing the complexity of prediction task and reducing prediction accuracy in iCache. In this article, we set the cache partitioning frequency to 3 and configure the cache granularity to 10, striking a balance between cache management flexibility and resource allocation efficiency.

3) Model Comparison: The selection of the machine learning model is a critical component of iCache, as the model determines the accuracy of the predictions. We compared various machine learning algorithms based on three metrics: 1) prediction error; 2) model size; and 3) inference time, where smaller values indicate better performance. We used the common metric MSE to evaluate prediction error. For the models, we selected representative traditional machine learning methods, such as Random Forest, deep learning methods like MLP, and the temporal model LSTM. As shown in Table III, the LSTM model outperforms the other two models in terms of prediction accuracy and storage size, albeit with slightly higher inference time. The superior prediction accuracy of LSTM is attributed to its ability to remember past information and capture long-term dependencies in time series, which is crucial for predicting dynamic changes in workloads. iCache performs model predictions every three minutes, and the time overhead distributed across each request is negligible. The LSTM model strikes a better balance between performance and overhead, thereby optimizing cache allocation.

TABLE III
PREDICTION ERROR AND OVERHEAD FOR DIFFERENT MODELS

Model	MSE	Inference time	Storage size
Random Forest	1.18	4.2 ns	100.8 KB
MLP	0.08	0.13 us	79.6 KB
LSTM	0.03	2.2 ms	70.7 KB

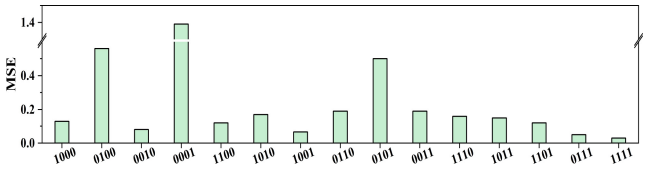


Fig. 15. Model prediction error for different number of features.

4) Feature Count Impact on MSE: To evaluate the model's performance, we tested the prediction error under varying numbers of features (ranging from 1 to 4). For example, the symbol "1111" means that the four features (I/O intensity, write ratio, request size, and unique address rate) are considered simultaneously. Symbol "1000" means that only the feature of I/O intensity is considered in the model. Fig. 15 indicates that the prediction error is minimized considering all four features. This is attributed to the dynamic characteristics of workloads. The impact of different features on system performance varies across stages. For instance, in a time frame, the proportion of write requests might be the most critical factor influencing cache performance, while in another period, the frequency of unique address accesses or the average size of requests could become the decisive factor. Therefore, iCache, by employing four features for prediction, considers multidimensional features in workloads, achieving sensitive detection of workload dynamics and optimizing resource allocation.

VI. RELATED WORK

A. Fairness Scheduling

For fairness scheduling, FLIN [8] provides fairness between requests from different applications. FIOS [9] achieves fair resource allocation under high I/O loads and reduces read and

write blocking. Jun and Shin [35] proposed a workload-aware scheduling algorithm that considers performance isolation and fairness for request scheduling in multivirtual machine environments. Fair-GC [36] achieves scheduling policies to achieve fairness under GC interference. Researchers use fuzzy [37] logic to control fairness within SSDs. Scheduling policy for concurrent requests of multiple tenants will also generate scheduling interference, all of which will generate intertenant I/O interference. This I/O interference stems from the different characteristics of tenants, including intensity, size, and access type. For example, high-intensity loads issue more I/O requests per unit of time, forcing I/O requests from low-intensity loads to cause longer queuing wait time.

B. Cache Scheme

For caching schemes inside SSDs, the shared cache policy may cause active data streams to occupy too much cache and reduce the hit rate of other streams. In a multitenant environment, both equal distribution and shared caching strategies struggle to respond to dynamic changes in workloads. The stream-based cache allocation approach allocates cache space based on stream characteristics, such as access frequency or data size. For example, more caches can be allocated for streams with high access frequency to improve the cache hit rate and reduce I/O latency. Conversely, less cache can be allocated to conserve resources for streams with low access frequency. Jun and Shin [35] proposed the ghost caching mechanism to adaptively adjust cache allocations. NCache [38] employs a decision tree model to enhance cache hit rate by identifying valuable data but ignores interworkload fairness. CostPI [15] improves performance by dedicating cache slots, but may reduce overall utilization. As research progressed, scholars began to explore dynamic cache allocation policies based on workload characteristics. Justitia [1] reserves privileges for low-access loads to improve fairness. These strategies dynamically adjust cache resource allocation by monitoring I/Os patterns in real time with a view to achieving better performance and lifetime. MLCache [2] builds a load-aware cache allocation scheme using MLP models by learning the effect of reuse distance on cache allocation. This approach can more accurately match workload characteristics. However, the complexity of multivariate and dynamic environments is such that a single load characterization may not be able to predict and optimize cache performance in a multitenant environment fully. Increasing cache hit rates may sacrifice fairness across multiple applications, and high hit rates do not directly reduce latency or improve overall performance.

Previous work has made significant contributions to SSD cache management and fairness scheduling, but few studies have combined these aspects. For shared solid-state drives, static management strategies and single-metric-based dynamic strategies struggle to capture the dynamism and diversity of workloads. Additionally, ignoring the *performance cliff* and *performance saturation* phenomena poses risks of increased response times and decreased cache utilization. Our work aims to enhance workload response speed and fairness by proposing an intelligent cache management strategy tailored for shared solid-state drives. This strategy not only considers the impact of multiple features on cache allocation but also predicts and leverages two key metrics MBP and OAP for fair cache distribution, thereby improving cache utilization and workload response speed.

VII. Conclusion

In this article, we brought forth a multitenant SSD caching allocation scheme named iCache, which is capable of optimizing cache resource distribution in MSSDs. iCache utilizes LSTM to predict the MBP and OAP measures from historical access patterns of workloads. By leveraging predicted resource needs alongside priority scheduling and intelligent resource allocation strategies, iCache significantly enhances the efficiency and precision of resource distribution: iCache effectively prevents performance degradation caused by improper allocation. The experimental results unfold that from perspectives of overall performance and response time of MSSDs, iCache substantially outperforms the conventional shared schemes and the latest machine-learning-based method—MLCache.

References

[1] R. Liu, Z. Tan, L. Long, Y. Wu, Y. Tan, and D. Liu, "Improving fairness for SSD devices through DRAM over-provisioning cache management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2444–2454, Oct. 2022.

[2] W. Liu, J. Cui, T. Li, J. Liu, and L. T. Yang, "A space-efficient fair cache scheme based on machine learning for NVMe SSDs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 1, pp. 383–399, Jan. 2023.

[3] H. Fan et al., "QoS-pro: A QoS-enhanced transaction processing framework for shared SSDs," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 1, pp. 1–25, 2024.

[4] J. Kim, D. Lee, and S. H. Noh, "Towards SLO complying SSDs through OPS isolation," in *Proc. 13th USENIX Conf. File Storage Technol. (FAST)*, 2015, pp. 183–189.

[5] J. Huang et al., "FlashBlox: Achieving both performance isolation and uniform lifetime for virtualized SSDs," in *Proc. 15th USENIX Conf. File Storage Technol. (FAST)*, 2017, pp. 375–390.

[6] M. Kwon, D. Gouk, C. Lee, B. Kim, J. Hwang, and M. Jung, "DC-Store: Eliminating noisy neighbor containers using deterministic I/O performance and resource isolation," in *Proc. 18th USENIX Conf. File Storage Technol. (FAST)*, 2020, pp. 183–191.

[7] J. Kim, E. Lee, and S. H. Noh, "I/O scheduling schemes for better I/O proportionality on flash-based SSDs," in *Proc. IEEE 24th Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst. (MASCOTS)*, 2016, pp. 221–230.

[8] A. Tavakkol et al., "FLIN: Enabling fairness and enhancing performance in modern NVME solid state drives," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, vol. 10, 2018, pp. 397–410.

[9] S. Park and K. Shen, "FIOS: A fair, efficient flash I/O scheduler," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, vol. 12, 2012, pp. 1–13.

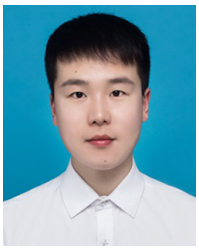
[10] B. S. Kim, "Utilitarian performance isolation in shared SSDs," in *Proc. 10th USENIX Workshop Hot Topics Storage File Syst. (HotStorage)*, 2018, pp. 1–14.

[11] R. Liu, D. Liu, X. Chen, Y. Tan, R. Zhang, and L. Liang, "Self-adapting channel allocation for multiple tenants sharing SSD devices," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 2, pp. 294–305, Feb. 2022.

[12] H. Sun, S. Dai, J. Huang, and X. Qin, "Co-active: A workload-aware collaborative cache management scheme for NVMe SSDs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 6, pp. 1437–1451, Jun. 2021.

[13] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, and J. Lee, "CFLRU: A replacement algorithm for flash memory," in *Proc. Int. Conf. Compilers, Archit. Synth. Embedded Syst.*, 2006, pp. 234–241.

[14] H. Sun, H. Tong, Y. Yue, and X. Qin, "LAC: A workload intensity-aware caching scheme for high-performance SSDs," *IEEE Trans. Comput.*, vol. 73, no. 7, pp. 1738–1752, Jul. 2024.
[15] J. Liu, F. Wang, and D. Feng, "CostPI: Cost-effective performance isolation for shared NVMe SSDs," in *Proc. 48th Int. Conf. Parallel Process.*, 2019, pp. 1–10.
[16] N. Beckmann and D. Sanchez, "Talus: A simple way to remove cliffs in cache performance," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2015, pp. 64–75.
[17] L. Liu, X. Dou, and Y. Chen, "Intelligent resource scheduling for co-located latency-critical services: A multi-model collaborative learning approach," in *Proc. 21st USENIX Conf. File Storage Technol. (FAST)*, 2023, pp. 153–166.
[18] G. Yadgar, M. Gabel, S. Jaffer, and B. Schroeder, "SSD-based workload characteristics and their performance implications," *ACM Trans. Storage*, vol. 17, no. 1, pp. 1–26, 2021.
[19] Q. Zou, Y. Zhu, J. Chen, Y. Deng, and X. Qin, "Characterization of I/O Behaviors in cloud storage workloads," *IEEE Trans. Comput.*, vol. 72, no. 10, pp. 2726–2739, Oct. 2023.
[20] B. Zhou et al., "SAC: A stream aware write cache scheme for multi-streamed solid state drives," in *Proc. 26th Asia–South Pacific Design Autom. Conf.*, 2021, pp. 645–650.
[21] L. V. Rodriguez et al., "Learning cache replacement with CACHEUS," in *Proc. 19th USENIX Conf. File Storage Technol. (FAST)*, 2021, pp. 341–354.
[22] H. Wang, X. Yi, P. Huang, B. Cheng, and K. Zhou, "Efficient SSD caching by avoiding unnecessary writes using machine learning," in *Proc. 47th Int. Conf. Parallel Process.*, 2018, pp. 1–10.
[23] Y. Liu, H. Wang, K. Zhou, C. Li, and R. Wu, "A survey on AI for storage," *CCF Trans. High Perform. Comput.*, vol. 4, no. 3, pp. 233–264, 2022.
[24] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, "DeepCache: A deep learning based framework for content caching," in *Proc. Workshop Netw. Meets AI ML*, 2018, pp. 48–53.
[25] S. Ebrahimi, R. Salkhordeh, S. A. Osia, A. Taheri, H. R. Rabiee, and H. Asadi, "RC-RNN: Reconfigurable cache architecture for storage systems using recurrent neural networks," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 3, pp. 1492–1506, Jul.–Sep. 2022.
[26] Y. Zhang et al., "OSCA: An online-model based cache allocation scheme in cloud block storage systems," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2020, pp. 785–798.
[27] K. Zhou et al., "Efficient SSD cache for cloud block storage via leveraging block reuse distances," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 11, pp. 2496–2509, Nov. 2020.
[28] K. Liu, K. Wu, H. Wang, K. Zhou, J. Zhang, and C. Li, "SLAP: An adaptive, learned admission policy for content delivery network caching," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2023, pp. 457–467.
[29] Y. Hu, H. Jiang, D. Feng, H. Luo, and L. Tian, "PASS: A proactive and adaptive SSD buffer scheme for data-intensive workloads," in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*, 2015, pp. 54–63.
[30] "Cortex-r8." Arm. 2024. [Online]. Available: https://developer.arm.com/Processors/Cortex-R8
[31] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," *ACM SIGPLAN Notices*, vol. 44, no. 3, pp. 229–240, 2009.
[32] M. Jung and M. T. Kandemir, "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2014, pp. 524–535.
[33] 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 *Proc. 16th USENIX Conf. File Storage Technol. (FAST)*, 2018, pp. 49–66.
[34] C. Lee, T. Kumano, T. Matsuki, H. Endo, N. Fukumoto, and M. Sugawara, "Understanding storage traffic characteristics on enterprise virtual desktop infrastructure," in *Proc. 10th ACM Int. Syst. Storage Conf.*, 2017, pp. 1–11.
[35] B. Jun and D. Shin, "Workload-aware budget compensation scheduling for NVMe solid state drives," in *Proc. IEEE Non-Volatile Memory Syst. Appl. Symp. (NVMSA)*, 2015, pp. 1–6.
[36] C. Ji et al., "Fair down to the device: A GC-aware fair scheduler for SSD," in *Proc. IEEE Non-Volatile Memory Syst. Appl. Symp. (NVMSA)*, 2019, pp. 1–6.
[37] S. Tripathy, D. Sahoo, M. Satpathy, and M. Mutyam, "Fuzzy fairness controller for NVMe SSDs," in *Proc. 34th ACM Int. Conf. Supercomput.*, 2020, pp. 1–12.
[38] H. Sun, Q. Cui, J. Huang, and X. Qin, "NCache: A machine-learning cache management scheme for computational SSDs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 6, pp. 1810–1823, Jun. 2023.



Donghua Li is currently pursuing the Ph.D. degree with Anhui University, Hefei, China.

His main research interests include SSD storage and intelligent cache management.

Hui Sun (Member, IEEE) received the Ph.D. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2014.

He is an Associate Professor of Computer Science with Anhui University, Hefei, China. His research interests include computer systems, edge computing, performance evaluation, nonvolatile memory-based storage systems, file systems, and I/O architectures.

Xiao Qin (Senior Member, IEEE) received the B.S. and M.S. degrees in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 1992 and 1999, respectively, and the Ph.D. degree in computer science from the University of Nebraska–Lincoln, Lincoln, NE, USA, in 2004.

He is currently an Alumni Professor and the Director of Graduate Programs with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL, USA. His research interests include parallel and distributed systems, storage systems, and performance evaluation.

Dr. Qin won an NSF CAREER Award in 2009.