RC-RNN: Reconfigurable Cache Architecture for Storage Systems Using Recurrent Neural Networks

Shahriar Ebrahimi, Reza Salkhordeh, Seyed Ali Osia, Ali Taheri, Hamid R. Rabiee, and Hossen Asadi

Abstract—*Solid-State Drives* (SSDs) have significant performance advantages over traditional *Hard Disk Drives* (HDDs) such as lower latency and higher throughput. Significantly higher price per capacity and limited lifetime, however, prevents designers to completely substitute HDDs by SSDs in enterprise storage systems. SSD-based caching has recently been suggested for storage systems to benefit from higher performance of SSDs while minimizing the overall cost. While conventional caching algorithms such as *Least Recently Used* (LRU) provide high hit ratio in processors, due to the highly random behavior of *Input/Output* (I/O) workloads, they hardly provide the required performance level for storage systems. In addition to poor performance, inefficient algorithms also shorten SSD lifetime with unnecessary cache replacements. Such shortcomings motivate us to benefit from more complex non-linear algorithms to achieve higher cache performance and extend SSD lifetime.

In this paper, we propose *RC-RNN*, the first reconfigurable SSD-based cache architecture for storage systems that utilizes machine learning to identify performance-critical data pages for I/O caching. The proposed architecture uses *Recurrent Neural Networks* (RNN) to characterize ongoing workloads and optimize itself towards higher cache performance while improving SSD lifetime. *RC-RNN* attempts to learn characteristics of the running workload to predict its behavior and then uses the collected information to identify performance-critical data pages to fetch into the cache. We implement the proposed architecture on a physical server equipped with a *Core-i7* CPU, 256GB SSD, and a 2TB HDD running Linux *kernel 4.4.0.* Experimental results show that *RC-RNN* characterizes workloads with an accuracy up to 94.6% for SNIA I/O workloads. *RC-RNN* can perform similarly to the optimal cache algorithm by an accuracy of 95% on average, and outperforms previous SSD caching architectures by providing up to 7x higher hit ratio and decreasing cache replacements by up to 2x.

Index Terms—Data Storage Systems, I/O Caching, Solid-State Drives, Recurrent Neural Networks, Applied Machine Learning, I/O Workload Characterization

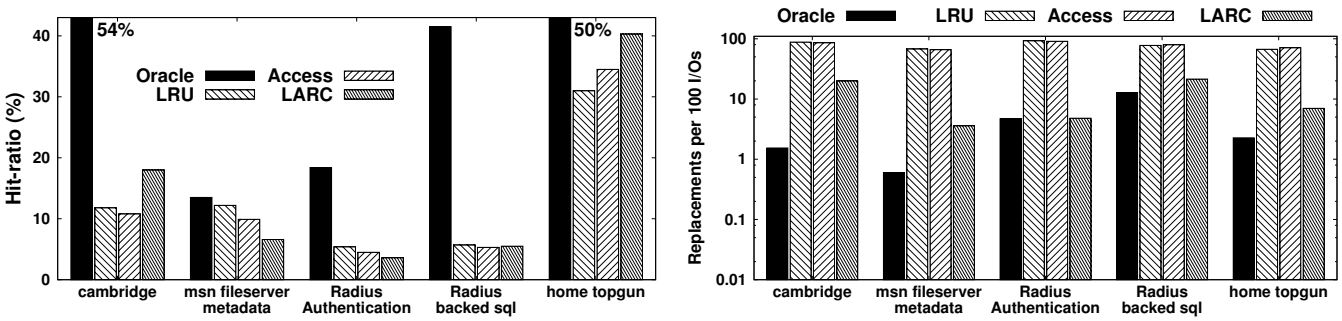
1 Introduction

Storage subsystems have significant impact on the overall performance of enterprise *Input/Output* (I/O) intensive applications. The major performance bottleneck of storage subsystem is mechanical storage devices such as *Hard Disk Drives* (HDDs), which suffer from limited response time. With emergence of *Flash-based Solid-State Drives* (SSDs) that have no mechanical components, the performance of storage subsystem can be significantly improved. SSDs, however, have several drawbacks such as reliability concerns [1], [2] and one order of magnitude higher price per capacity compared to HDDs [3].

To alleviate the shortcomings of SSDs while exploiting their benefits, employing SSD as an I/O cache for HDD-based storage subsystems has been studied in the previous studies [3], [4], [5], [6], [7], [8], [9], [10]. In SSD-based I/O caching architectures, frequently accessed requests are buffered in the SSD to provide fast response time while other requests are supplied by HDDs. Several application domains such as *Database Management Systems* (DBMS), mail servers, *OnLine Transaction Processing* (OLTP), and *High Performance Computing* (HPC) can benefit from SSD-based I/O caching [3], [5], [7]. Each aforementioned domain, however, has a distinct workload characteristic and therefore, requires a specific caching policy to fully exploit the benefits of SSD.

Flash cells in SSDs need to be erased before writing. Such cells can *only* endure a limited number of erases and hence, have a limited lifetime. Each cache replacement requires writing a new data page to the SSD and therefore, the number of cache replacements directly affects the SSD lifetime. Moreover, Due to the limited and expensive capacity of SSDs compared to HDDs, the cost of buffering I/O requests is very high. Therefore, accurately identifying the performance-critical data pages is crucial for a cost-efficient SSD-based caching architecture. Hence, for efficiently managing the I/O cache to improve performance while extending SSD lifetime, caching architectures should: (1) have accurate knowledge about the behavior of the current workload, and (2) be able to reconfigure themselves in case of workload changes. Such reconfiguration can be conducted by providing feedback, based on changes in hit the ratio [4], [5], [6]. Employing static flags for requests to distinguish metadata from data requests has also been suggested in the previous studies [7], [8]. Moreover, in architectures such as [3], [11], cache priorities are updated periodically based on the workload type. In addition, several previous studies

• *All of the authors are associated with the Department of Computer Engineering at Sharif University of Technology, Tehran, Iran. E-mails: shebrahimi@ce.sharif.edu, salkhordeh@ce.sharif.edu, osia@ce.sharif.edu, taheri@ce.sharif.edu, rabiee@sharif.edu, asadi@sharif.edu.*



(a) Hit Ratio (b) Cache Replacements

Fig. 1: Oracle Compared to Conventional Algorithms

aim to reduce the number of writes in SSD to increase SSD lifetime even at the cost of performance degradation [5], [10].

A major limitation of existing SSD-based caching architectures such as [4], [5], [6], [7] is that they are adapted from conventional caching algorithms employed in either main memory or processors cache levels. Such algorithms depend on high temporal and spatial locality while I/O workloads mostly exhibit unpredictable behavior with no linear locality [12], which degrades cache performance and SSD lifetime by issuing inefficient cache replacements. To show the inefficiency of previous studies in I/O workloads, we run experiments to compare hit ratio of *Least Recently Used* (LRU), *Access Frequency* [4], and *LARC* [5] with the *Optimal* Cache replacement policy [13] (Oracle/Belady) under four different I/O traces from *Storage Networking Industry Association* (SNIA) [14]. Figure 1a and Figure 1b show the hit ratio and the number of cache replacements conducted for conventional algorithms as opposed to the *Oracle* (i.e., optimal) algorithm, respectively. Although conventional algorithms have high hit ratio in workloads with high locality such as *mail_index*, they still impose significant number of unnecessary cache replacements, which decreases SSD lifetime. Moreover, under workloads such as *Radius_Backed_SQL*, conventional algorithms fail to provide high hit ratio due to the low linear locality in I/O requests. [3] These experiments reveal that conventional algorithms are *not* efficient for SSD-based I/O caching and impose high number of unnecessary cache replacements in many scenarios.

In addition, several existing architectures: a) are optimized towards a selection of workloads [5], [6], [7], [8] *or* b) modify standard filesystems to provide information about request types [7], [8], which makes them heavily dependent on a particular filesystem. Unlike CPU level caches, the average response time of I/O requests is several milliseconds, which provides sufficient *thinking time* to execute complex computations to efficiently identify and predict workload behavior. Such predictions can help caching architectures to reduce the performance and endurance gap between optimal and conventional algorithms. To the best of our knowledge, *none* of the previous studies have utilized machine learning methods to achieve higher cache performance in storage systems while preserving the cache lifetime intact.

In this paper, we propose the first reconfigurable SSD-based cache architecture, called RC-RNN, which tries to improve both performance and endurance of SSD-based caches by employing *Machine Learning* to: **a)** identify workload type, **b)** decide which data pages should be buffered, and **c)** decide which data pages are no longer beneficial to be buffered. RC-RNN proposes several request characteristics, which are used by the *Machine Learning* method to accurately identify the running workload. For each workload type, a *Machine Learning* model is constructed, which performs very similar to the *Optimal* cache policy by deciding to ignore or buffer miss accesses and evicting the cold data pages from the cache. The proposed cost function employed to construct the *Machine Learning* model considers several workload and storage device characteristics in order to accurately estimate the cost and benefit of buffering data pages.

We utilize *Recurrent Neural Network* (RNN) in the proposed architecture as one of the most powerful machine learning methods, which is proven to be accurate in several application domains such as text analysis [15] and speech recognition [16]. Finding patterns in a trace of the I/O requests has similarities to both of the mentioned application domains, which encouraged us to select RNN as the machine learning method. RC-RNN consists of offline and online phases. Any time-, CPU-, and memory-consuming operations are placed in the *offline* phase and are done only *once*. In the online phase, RC-RNN monitors the running workload and decides which data pages should be buffered or evicted based on the constructed models. To improve SSD lifetime, RC-RNN does not copy all missed data pages to the cache. The data pages with low probability of access in the future are ignored and responded directly by HDD to reduce the number of writes in SSDs.

RC-RNN is designed to be reconfigurable with negligible reconfiguration cost. To this end, it monitors I/O requests online and evaluates the workload time periodically offline. Upon detection of any change in the workload type, *RC-RNN* switches the loaded RNN model to match the new workload type. Note that the reconfiguration process is accomplished only by a swap of small RNN models (around 40 MB) in memory, which also can be loaded simultaneously due to low memory consumption. The internal state of cache is also updated to reflect the workload change. The reconfiguration process enables *RC-RNN* to have high accuracy in various workload types unlike previous studies which are optimized toward a few workloads [4], [5], [7].

We evaluate *RC-RNN* by using real SSDs and HDDs in a server equipped with *Core-i7* CPU running *Linux 4.4.0*

on *Ubuntu 16.04* operating system. RNN models are implemented by using the *Keras* [17] library and the *online* phase experiments are evaluated *only* by CPU. We show that the proposed characterization method has up to 94.5% accuracy in detecting workloads from SNIA I/O traces [14]. Experimental results show that *RC-RNN* can perform similarly to Oracle algorithm with less than 5% error in both preventing seldom accessed data pages from entering the cache and timely deciding the data pages to evict. *RC-RNN* improves hit ratio by up to 7x (2x on average) compared to state-of-the-art caching architectures. By filtering seldom accessed data pages, *RC-RNN* reduces the number of writes in SSD by up to 22% (1.7% on average) compared to previous studies.

In summary, the **main contributions** of this work are as follows:

- We propose the first comprehensive I/O workload characterization method, which considers long dependencies between requests. Unlike previous workload characterization methods, which *only* consider a limited number of past requests, the proposed method can identify several streams of requests originating from a single or multiple applications.
- We introduce RC-RNN, the first I/O caching architecture capable of employing machine learning methods to decide which data pages should be placed in or evicted from the cache in a dynamic priority-based caching policy. RC-RNN constructs a model from an optimal cache policy and employs it in the runtime to decide suitable cache operations for the running workload.
- To improve the accuracy of the machine learning model employed in RC-RNN, we propose a cost function that considers the characteristics of SSDs as compared to HDDs to estimate the actual cost of caching, by-passing, or evicting a data page.
- RC-RNN monitors the running workload using the constructed RNN model for workload characterization, where once a change in the workload is detected, the employed RNN model for cache policy is dynamically changed. The performance impact of the reconfiguration process is kept as minimum as possible.
- We implement RC-RNN in a real system and show that the proposed architecture improves cache hit ratio up to 7x (2x on average) with negligable overhead during *online* phase compared to previous work.

The rest of the paper is organized as follows. In Section 2, the background and previous studies are presented. Section 3 discusses motivations of this paper. In Section 4, the proposed architecture is detailed. Experimental results are presented in Section 5. Finally, Section 6 concludes the paper.

2 Background and Related Works

In this section, we first discuss emergence of new technologies for storage devices and the motivations to employ heterogeneous hybrid architectures. Next, the suggested hybrid architectures in previous studies are detailed. Current I/O workload characterization methods are also discussed in this section. Finally, a summary of machine learning methods, applicable to I/O caching is provided.

2.1 Hybrid Architectures

With emergence of SSDs having performance advantages over mechanical HDDs, the average response time of storage devices has been decreased significantly. However, the higher price of SSDs compared to HDDs and limited lifetime prevents data centers to completely replace HDD-based storage subsystems with SSDs.

Multi-tiered storage subsystems have been suggested by previous studies to exploit the advantages of both mechanical and non-mechanical storage devices [4], [5], [6], [7], [8], [9], [10], [18], [19]. There are two main approaches in hybrid storage systems: (1) *Tiering* and (2) *Caching*. In tiering, SSD is placed at the same level as HDD and the overall capacity of the system is equal to sum of the space of both devices [18], [19]. Contrary to tiering, in caching a copy of data page is moved to SSD to speed up I/O requests. Tiering is more efficient in terms of overall cost and capacity compared to caching, but due to costly data migration between tiers, it has lower performance during abrupt workload changes. Therefore, tiering is more suitable for systems with steady workloads [20], [21]. On the other hand, a caching approach achieves higher performance in workloads with sudden changes, which makes them efficient on systems running simultaneous workloads.

In this paper, we focus on caching since it is capable of responding faster to workload changes. Due to the unpredictable and non-linear behavior of I/O workloads, the performance of an SSD-based caching approach highly depends on the caching algorithm. Using an inefficient caching algorithm not only degrades performance, it also shortens SSD lifetime because of the unnecessary cache replacements. To achieve efficient caching management, it is necessary to predict behavior of workload and be able to configure caching architecture based on the access pattern. In addition, in case of workload change, previously configured caching algorithm may not be efficient any more. Hence, it is important to have a reconfigurable architecture, which can adapt to rapid changes in the workload.

2.2 Previous Hybrid Architecture

There are numerous studies in SSD-based caching for storage systems [4], [5], [6], [7], [8], [9], [10], [22]. Most of these studies suggest simple and linear caching algorithms based on conventional algorithms such as *Least Recently Used* (LRU) or *Access Frequency* that have shown to be practical and efficient in other levels of memory hierarchy such as CPU caches. Although LRU has low overhead, certain workloads can cause cache thrashing [3], [23] and significantly decrease its efficiency.

Previous studies that use LRU as their baseline caching algorithm tried to overcome this problem with various approaches. ARC [4] employs two lists to keep track of recency and frequency of accessed I/O blocks. It buffers a percentage of each list based on the hit ratio in the recent accesses and keeps balance between blocks that have been accessed recently and blocks, which have been mostly accessed during the workload. *LARC* [5] employs a virtual LRU queue as a filter for recently accessed blocks and prevents randomly accessed blocks from entering cache in the first access. If a block is accessed for the second

time while in filter, it gets promoted to the main LRU and will be buffered. With this approach, *LARC* prevents cache thrashing and significantly decreases cache replacements compared to LRU. *mARC* [6] benefits from both previous architectures in different phases of the workload. It uses hit ratio, recency, and frequency as feedback to switch between two methods. Although the mentioned studies have improved SSD caching performance in few workloads, they still have low performance in other workload types. In addition, the number of unnecessary cache replacements is still high in such architectures.

In addition to modifying LRU, using access frequency has also been suggested in the previous studies [7], [8]. In [7], a caching architecture called Azor is suggested, which is an access frequency cache that assigns metadata blocks a higher priority over data blocks. This architecture will be effective only for workloads with high percentage of metadata accesses. In case of workload change, older metadata blocks, which are not performance critical anymore, will be kept in cache and prevent performance critical pages from entering the cache. *Hystor* [8] proposes a hybrid storage architecture consists of both SSD and HDD to improve the overall performance. One of the drawbacks of *hystor* is allocating a fixed section of SSD as a writeback buffer for HDD subsystem, which can be either too large or too small for certain workloads. The size of the write-back cache significantly affects the performance of running workloads.

2.3 Workload Characterization

The performance of caching architectures in storage systems is highly affected by the characteristics of the running workload. Therefore, workload analysis plays a key role in studies aimed at improving performance of I/O subsystems. The behavior of enterprise application workloads is very complex and difficult to characterize because the performance of each request depends on the previous ones. The non-linear behavior of I/O workloads and burstiness of requests [3], [24] add to the complexity of characterization.

To understand the I/O workload characteristics, several previous studies have proposed different analytical models for analyzing enterprise I/O traces [25], [26], [27], [28], [29]. *Read/write ratio*, *I/O size*, and *inter-arrival times* are a few of the common parameters employed in previous studies for workload characterization. In [25], spatial locality and outstanding I/Os are considered in characterization of workloads for HDD-based storage systems. In [28], workloads are divided into three major domains: enterprise, desktop, and consumer electronics. This work shows that burstiness as a temporal locality measure is highly application dependent and significantly affects the overall workload performance. In [29], six different web servers are characterized in detail and two proposed caching strategies for web caches are obtained through data analysis. Although several parameters such as *read/write ratio, I/O size, temporal* and *spatial* localities, and *burstiness* have been examined in previous studies, they still cannot predict the I/O workload behavior with high accuracy. As we show in Section 3.2, this is due to the long dependency of requests to previous requests, which cannot be captured by state-of-the-art characterization methods.

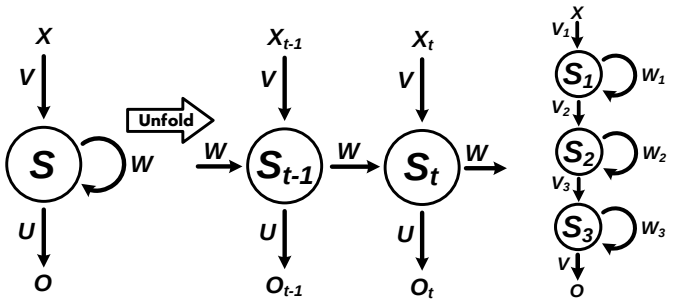
2.4 Machine Learning - RNN

In recent years, artificial intelligence has been extensively used in a broad range of applications such as multimedia signal processing (e.g., image, video and speech processing), intelligent systems (e.g., autonomous cars and smart homes) and bio-informatics (e.g., DNA analysis) [15], [16], [30], [31]. Such applications need complicated analysis of large volume of data, which is usually achieved by machine learning methods that recognize the complex implicit patterns of the data and use it for future prediction. In general, building a machine learning model has two phases: *training* and *testing*. In the training phase, which is done offline, a large volume of training data is analyzed to solve an optimization problem for learning a model. In the test phase, the trained model is used to predict features of unseen new data. Among machine learning methods, *Deep Learning* has made great progress in recent years and has achieved state-of-the-art results in many problems [15], [16], [31]. *Deep Convolutional Neural Networks* (CNNs) and *deep Recurrent Neural Networks* (RNNs) are the most useful architectures in this field. The former is used to analyze image data and the latter is employed for sequential data analysis such as text and time series data [32]. We are going to use RNNs in this paper.

RNN attempts to model a nonlinear dynamical system with input $x(t)$, inner state $s(t)$, and output $y(t)$. As an example in data storage systems, the input data $x(t)$ can be considered as a request at time t from an I/O workload trace and $y(t)$ indicates the caching mechanism suitable for individual requests. In this scenario, the state $s(t)$ plays the role of the memory and tries to summarize all of the past events $(x(1), ..., x(t-1))$. RNN implements the following scenario in a recursive manner: 1) take the new input data, 2) check the previous state, 3) mix them together to build the current state, and finally 4) make the new output by manipulating the current state:

$$s(t) = f(s(t-1), x(t))$$
$$y(t) = g(s(t)) \quad (1)$$

In order to model the non-linearity of the system, f and g are usually obtained by combining a linear transformation and a simple nonlinear function, e.g., *sigmoid*. Since RNNs, unlike feed-forward neural networks, can use their internal memory to process arbitrary sequences of inputs, they are more suitable for analyzing I/O traces. In RNNs, connections between the nodes form a directed cycle to imitate a dynamic temporal behavior. RNNs can use their internal memory to process arbitrary sequences with input, inner *state*, and output layers. A simple RNN with one hidden layer as the inner state takes the sequential input and updates its current state based on the current input and previous state. The inner state plays the role of the memory and tries to summarize the input data. The output which could be the result of classification or regression, is produced by manipulating the state of the system. Since natural dynamical systems are non-linear, RNN also uses simple nonlinear functions such as *sigmoid*, *tanh*, or *max*, and builds a complex non-linear function by combining them [30]. A schematic of a simple RNN is shown in Figure 2a. Equation 2 shows the relation between input (x), state (s),



(a) Simple RNN (b) Deep RNN

Fig. 2: Different RNN Architectures [15]

and output (u) of a typical RNN. Where g and f are the non-linear functions (in classification, common choices for these functions are the *rectifier linear unit* and *softmax* function), and A, B, and C are the weights, which should be learned during the training process.

$$s(t) = f(A.s(t-1) + B.x(t)), \quad u(t) = g(C.s(t)) \qquad (2)$$

For training RNNs, we should first prepare suitable training datasets, which consist of sufficient input sequences with their correct ground truth label. The training is achieved by using an efficient implementation of *Stochastic Gradient Descent* (SGD) algorithm, called *back propagation through time*. We obtain the best estimates for the weights through the training process, but the model should be evaluated by using test datasets. Different challenges, e.g. vanishing and exploding gradients, might be raised during the training of a simple RNN. These challenges can be resolved by using a special kind of memory unit called *Long Short Term Memory* (LSTM) [33] in the RNN architecture. LSTM defines the inner state with a more complex process, with input, output, and forget gate. The details of this memory unit can be found in [33]. Increasing the number of hidden layers and building a deep RNN is an important generalization of RNN, which can handle more complex data patterns at the cost of more time and memory consuming training process. Figure 2 shows the schematic of a simple RNN (Figure 2a) and a 3-layered deep RNN (Figure 2b), respectively.

RNN models are mostly used on one dimensional datasets. However, RNN can also be employed on multi-dimensional datasets by feeding d-dimensional vector (e.g. [addr, size, ...]) at each time step as shown in several previous studies such as [34]. RC-RNN uses the same technique to feed the multi-dimensional dataset to the RNN model. For instance, we use a 100*4 matrix for 100 consecutive time steps to classify the characteristics of the workload based on *four* different dimentions.

3 Motivation

The motivation for this work is three-fold. First, we show the gap between state-of-the-art and optimal (*Oracle*) caching architectures in terms of performance and SSD endurance. Second, we demonstrate the inaccuracy of previous workload characterization methods in several scenarios. Finally, we present several examples of the potential benefits of employing machine learning in both I/O workload characterization and SSD-based I/O cache management.

3.1 Cache Management

To evaluate the efficiency of caching architectures, hit ratio should be compared to the hit ratio of *Oracle* caching architecture, which is proven to have the maximum possible hit ratio due to having the knowledge about all future accesses [13]. Figure 1a shows the hit ratio of three different algorithms: 1) *LRU*, 2) *Access* [7], and 3) *LARC* [5], compared to *Oracle*. The hit ratio gap between *Oracle* and state-of-the-art algorithms is up to 7x. This gap highly depends on the workload reuse distance and access pattern. Previous studies rely on such characteristics of workloads to identify hot data pages. In workloads with low localities, previous studies fail to predict which data pages will be re-referenced.

In addition to the hit ratio, the number of cache replacements is also an important factor in SSD-based caching architectures. Cache replacements require a write operation on the SSD to copy the data page to the cache. Such write operations significantly decrease SSD lifetime and therefore, should be minimized. Figure 1b depicts the number of cache replacements for the same set of workloads as the previous experiment. As shown in this figure, *LARC* has the lowest number of cache replacements among state-of-the-art caching architectures. This is due to its two-level LRU, which prevents seldom accessed data pages from entering the cache. *Oracle* has up to 13x less cache replacements compared to *LARC*. Under *cambridge* workload [35], where accesses have long reuse distance, previous methods such as *LARC* fail to capture the relation between requests while most of the performance-critical blocks are evicted from the cache before they are accessed again. This results in significant hit ratio degradation compared to *Oracle*. On the other hand, frequent swaps between blocks in the cache results in more cache replacements compared to the *Oracle* caching architecture. The significant gap between the *Oracle* and state-of-the-art cache architectures in terms of hit ratio and cache replacements reveal that caching architectures still can be significantly improved by employing more complex policies.

3.2 Workload Characterization

In order to evaluate the accuracy of previously proposed characterization methods, three widely used characterization methods including a) *Temporal Working Set Distribution* (TWSD) [12], b) Frequency [7], [8], and c) IOSize [5] have been implemented and examined. *IOSize* characterizes requests based on their size. In *Frequency*, requests are characterized based on the access frequency, type (read/write), and size. *TWSD* method, which is the most detailed analysis on these three parameters (access frequency, type, and size), divides I/O requests into four types *strided*, *sequential*, *random*, and *overlapped* and takes the dependency of requests into account. A request is considered *sequential*, if its size exceeds a threshold or it starts (ends) at the end (start) of one of previous requests. *Strided* denotes requests with a small gap from previously accessed blocks. In addition, if a request overlaps with previous requests, it is flagged as

TABLE 1: Storage System Workload Scenarios

Scenario	Workloads
Single Purpose Server	Radius_Auth mail_index
Virtualization Server	home_ikki Radius_Auth mail_index
Storage System	enterprise_tpc_1 home_ikki Radius_Auth mail_index

overlapped. Finally, if a request does not meet any of the mentioned conditions, it will be flagged as *random*.

To evaluate the accuracy of the workload characterization methods, they are fed into an RNN model constructor as the main cost function. To be able to compare different workload characterization methods, we divide workload traces into two separate phases: *learning* and *evaluation*. During *learning* phase, each method collects the required information from more than 16 trace files, each having at least 60,000 requests. Extracted information from the trace and the workload type are given to machine learning classifier model constructor to build a model for each workload characterization method. In the evaluation phase, each workload characterization method is tested 100 times, where each test consists of 100 requests. The accuracy is calculated by the number of correct workload identifications.

To simulate various storage servers with different workload complexities, three scenarios are designed using *I/O traces* from *SNIA* [14]. Table 1 shows the three scenarios for the workloads. In order to see the results of introducing one additional workload to a system, we decide to keep previous workloads exactly in later scenarios. Therefore, the reduced accuracy of characterization methods would not depend on the modified workload types. This experiment indicates the impact of additional workloads on the accuracy of characterization. *Single Purpose Server* data set emulates an email server by running an authentication application and an email server. *Virtualization Server* runs three independent applications to emulate a server running several virtual machines. Finally, the last test data set (*Storage System*) emulates a storage system by running several applications each having a separate filesystem.

The goal of a characterization method is to find the type of running workload that the given set of I/Os belong to. To this end, the accuracy of characterization methods is calculated by the ratio of correct decisions of workload type over time, while workload types change periodically. For instance, in *single purpose* scenario, characterization methods must only differentiate between two workload types (Radius_auth or mail_index). Given one hundred requests from a workload, the characterizer must decide which of the workload types are more likely to have generated the given hundred requests.

Figure 3 shows the accuracy of the workload characterization methods on test data sets. *TWSD* and *Frequency* have high accuracy when only one application is running. By increasing the number of running applications, the accuracy is reduced significantly. The highest accuracy in the third test data set is 60%. This experiment reveals that the workload characterization methods fail to accurately predict workload

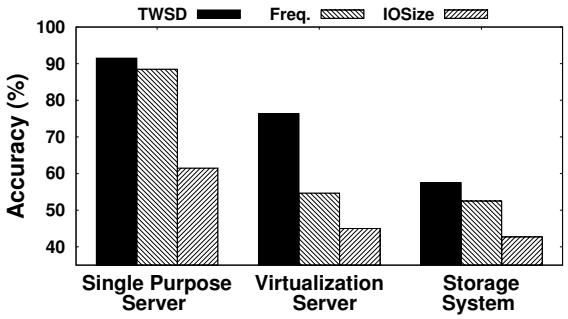


Fig. 3: Accuracy of Previous Workload Characterization Methods

when multiple applications are running. Therefore, more complex methods need be employed to reach high accuracy in workload characterization.

3.3 Exploiting Machine Learning Methods

Machine learning algorithms have shown to be successful in identifying complex data patterns in many domains by providing methods such as classification, regression, and clustering [15], [16], [30], [31], [32]. In addition, deep learning approaches such as RNNs are in general more accurate and more effective than the heuristics approaches that are currently employed in the caching management systems throughout the memory and I/O stack. However, such machine learning approaches require few hundreds of microseconds for processing. The average latency of I/O requests is few milliseconds, which can tolerate such additional processing time. However, main memory and CPU cache levels have less than one hundred nanoseconds latency, and therefore, cannot use machine learning approaches with such a *relatively* high latency.

SSD caching, on the other hand, can benefit from machine learning and specially RNNs. This is due to three main reasons: 1) the significant gap between state-of-the-art architectures and Oracle, 2) RNNs as one of the most accurate discriminative classifiers for sequences with time dependency, and 3) the average response time of several milliseconds in I/O requests. Thus, RNNs can be employed to detect complex *temporal* and *spatial* localities of I/O workloads. We note that the average response time of I/O requests in this layer (several milliseconds) is sufficient for processing different RNN modules using only CPU. Moreover, other parameters such as request *size* and *type* (*read*/*write*) can be employed to further improve the accuracy of such methods by defining *n*-dimensional neural vectors [34]. Therefore, RNNs can be used for identifying patterns in long sequences with similar characteristics to I/O requests. Employing RNNs with LSTM [33] units can enable us to efficiently analyze long I/O workload traces. We can conclude here that I/O traces are suitable input for several machine learning methods such as RNN and the runtime computation overheads (as our experiments indicate) do *not* have a significant effect on the overall performance.

Several studies have employed machine learning techniques in different levels of system hierarchy. C-Miner [36] is a prefetching method for sequential prefetching in storage systems. This method aims to provide faster I/O

transactions between operating system and the backend storage system by considering the frequent subsequences in the blocks and *pre-fetching* them to the memory. This is orthogonal to RC-RNN that decides which data pages should be moved to the SSD *after* they are accessed. Additionally, the patterns that RNN can identify in the workload are not discoverable by simple mining methods such as *frequent subsequences* that can only identify linear correlations, while I/O accesses have non-linear correlations. Therefore, more complex machine learning approaches such as RNN should be employed to identify patterns in the I/O workloads.

4 Proposed Architecture

To mitigate the shortcomings of previous studies in terms of providing high hit ratio and SSD lifetime, we propose RC-RNN, the first reconfigurable SSD-based I/O caching architecture employing RNN models. RC-RNN employs two RNN models: a) a single-layer RNN model for *classification* of the running workload and matching its characteristics to one of the predefined workload categories, and b) a deep three-layer RNN model, called *caching* model, to decide which data pages should be copied to/evicted from the cache. For each workload category, a separate *caching* RNN model is constructed, which is optimized toward the requirements of that specific workload type. Since constructing RNN models is rather time-consuming, RC-RNN is divided into *offline* and *online* phases. The offline phase is executed once for the entire lifetime of the system while the online phase monitors the I/O requests in the runtime. In the offline phase, RC-RNN constructs the required RNN models by analyzing a wide range of the workload traces from several enterprise applications. The fully built RNN models are later used in online phase to monitor workload and control the cache in real-time. During online phase, the ongoing workload is periodically characterized to identify the workload type. Based on the identified workload type, the corresponding RNN model is employed for cache management. Fig. 4 shows the detailed architecture and data flow of offline and online phases. In the remainder of this section, offline and online phases are detailed.

4.1 Offline Phase

The offline phase is responsible for a) providing an RNN model for identifying workload type, and b) providing a cache management RNN model for each workload type. To identify workload types, a collection of I/O traces from SNIA [14] and *filebench* [37] are examined and classified into four main workload categories: a) *Mail Server*, b) *Web Server*, c) *Database*, and d) *File Server*. The classification is based on the general trace categories in SNIA. As shown in Figure 4, all workload traces and their corresponding type are fed into the RNN constructor in a *supervised* manner (1) to build a model for identifying workload type based on the I/O access pattern.

The characterization method in RC-RNN is based on the deep learning model. The characterization model learns the difference of given workloads in a supervised manner during the offline phase. Then, during online phase (where evaluations are performed), the model decides which type of workload is more likely to have generated the requests in a given period of time.

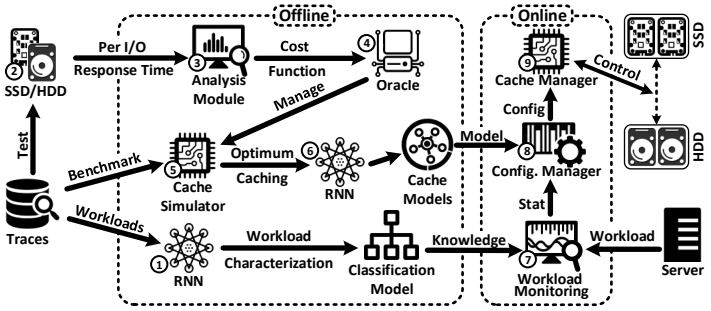


Fig. 4: Overall Two-phase Architecture Flow

To construct RNN models for cache management, we need to label all requests by the decision of the Oracle. The traditional Oracle algorithm *only* considers the accesses to data pages in its cost function. Due to the asymmetric performance of SSDs in read/write requests and also significant difference between sequential/random performance of HDDs, such algorithm might not result in the optimal cache decisions. Therefore, we propose a benefit function, which can accurately estimate the benefit of caching data pages based on the several request characteristics.

Moreover, (2) all of the traces are replied on real hardware (both SSD and HDD) to obtain the response time of the requests for each workload. The results are analyzed in *Analysis Module* (3) to calculate cost/benefit of caching each data page based on the access pattern. In addition to the response time of requests, read/write ratio and frequency of accesses to data pages are also considered in the benefit function. The employed benefit formula for building RNN models of cache management is presented in Equation 3.

As mentioned earlier in Section 1, the performance of SSDs is highly depended on I/O request type [3]. In order to use the most detailed SSD characterization analysis results, we employ *priority-based* caching mechanism with the benefit function (equation 3) to assign priorities to I/O requests. Therefore, requests with higher *benefit* will have higher chance of being cached by the system. T_{HDD} and T_{SSD} denote HDD and SSD response times for a request, respectively. Requests having wider gap between SSD and HDD response times will benefit more from entering the cache and therefore, will be assigned higher priority. Since requests with smaller sizes occupy less space in cache and larger requests have acceptable performance on HDDs, smaller requests assigned higher benefit by $\frac{1}{Req_{Size}}$ factor.

$$Benefit = \frac{T_{HDD}}{T_{SSD}} \times (N_{acc} - 1) \times \frac{1}{Req_{size}} \times (1 + \frac{N_{reads}}{N_{acc}}) \quad (3)$$

Improving read hit ratio reduces the number of data pages copied to the cache and therefore, reduces the number of writes in the SSD that increases SSD lifetime. Moreover, SSDs provide higher performance in read requests contrary to HDDs, which have almost identical performance on read and write requests [5]. Therefore, benefit of caching read-intensive data pages should be higher than write-intensive data pages. To this end, the read ratio of accesses to the data page $(1 + \frac{N_{reads}}{N_{acc}})$ is also considered in the benefit function. In addition to improving performance, prioritizing caching

of read-intensive data pages also improves SSD lifetime by reducing the number of writes in the SSD.

Oracle Cache (4) replays the trace file and for each access compares the benefit of caching a newly accessed data page with residing data pages in the cache. If the newly accessed data page has higher benefit, it replaces the data page with lowest benefit value in the cache. *Cache Simulator* (5) is a simple cache, which manages the data structures for the cache based on the decisions of the *Oracle Cache*. For each access, *Cache Simulator* adds two tags to the access in the trace file: a) *cached* and b) *duration*. The *cached* tag denotes that whether or not Oracle decided to place the data page in the cache. The *duration* tag denotes the number of accesses between entering a data page to the cache and replacing it with another data page. The value of *duration* depends on the cache size. Since our RNN models need to be cache-size independent, the actual value of *duration* is replaced with one of the three values reported in Table 2.

The boundaries are selected based on the empirical studies and in such a way, three groups are almost uniformly populated. A three-level *RNN* model (6) learns the behavior of the Oracle by analyzing the tagged trace files created by *Cache Simulator*. Four RNN models are constructed, one for each workload category. The output models are saved to be used later by the *online* phase of RC-RNN.

4.2 Online Phase

The RNN module tries to classify and distinguish four workload types by the five values provided for each I/O request. As shown in previous works such as [3], [12], [24], the history of I/O accesses can be employed to predict the I/O behavior of applications. Deep learning is able to identify the patterns in I/O workloads and its processing time is relatively small [38]. Hence, in this work we use deep learning to find the patterns in I/O workloads and decide the cache policy based on such information.

Workload Monitoring (7) employs the RNN model for workload characterization and identifies the running workload type. While the system is running, *Workload Monitoring* is invoked once a minute. It captures 1000 I/O requests and runs accesses through the RNN model for identifying workload type. If a change in the workload type is detected, *Workload Monitoring* replaces the current RNN model for cache management with RNN model of the new workload type. The process of identifying workload type is done asynchronously to prevent any delay in responding to I/O accesses.

Configuration Manager (8) is responsible for reconfiguring the cache when a change in the workload is identified by *Workload Monitoring*. The reconfiguration process upon workload change consists of two stages, a) loading new RNN model, and b) re-evaluating benefit values of data pages in the cache based on the new configuration. The reconfiguration process has very low overhead in terms of time and memory. All cache models are generated during *offline* phase and the *Configuration Manager* only switches between models during workload change and responds to the newly arrived I/O requests using the loaded model. All models have the same computation time, which is less than one millisecond for each access, on average. On multi-core CPUs, computation for I/O requests can be done simultaneously. The average size of RNN models is less than 40 megabytes. Note that the memory requirements of baseline SSD caching is about 0.2% of the entire SSD capacity. For a sample SSD size of 1 TB, the baseline SSD caching requires at least 2 GB of memory. Therefore, the memory overhead of RC-RNN model is negligible. Therefore, all models can be loaded into the main memory during the cache startup. However, if the memory demand becomes significantly high, parts of lesser used RNN models can be swapped out by the virtual memory management of the OS.

TABLE 2: Reclaiming Labels

Label	Duration Areas
soon	$duration < Cache\ Size$
mean	$Cache\ Size < duration < 5 \times Cache\ Size$
late	$5 \times Cache\ Size < duration$

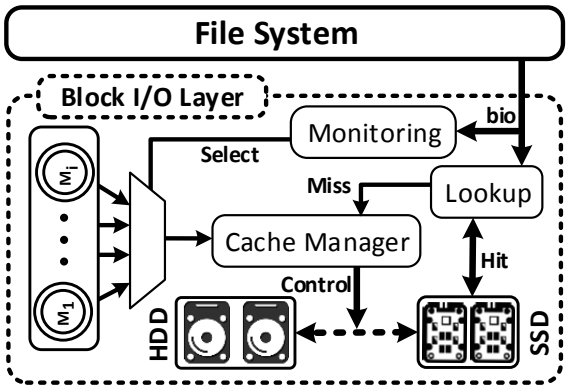


Fig. 5: Online Phase Architecture

Figure 5 shows the overall architecture of the implemented module for *online* phase of RC-RNN. *Lookup* module maintains a list of data pages in cache and redirects hit accesses to the SSD. Miss accesses are sent to the *Cache manager* to decide whether or not it should be cached. If the cache does not have any free space left, the cache manager decides which data pages should be evicted. The *Lookup* module handles all required I/O requests for moving data pages from/to the cache. The *Monitoring* module identifies current workload type and loads the corresponding RNN model into the *Cache manager*.

Cache manager (9) employs currently selected RNN model to decide a) whether or not to buffer the missed access, and b) which data page should be evicted when cache is full. Fig. 6 shows the overall flow of decisions made by *Cache manager* based on the outputs of Oracle. The output of RNN model for each access directly states that the request should be buffered or not. The eviction of data pages, however, requires additional process by the cache manager since RNN model only labels eviction time of requests as *soon*, *mean*, or *late*. To decide which pages should be evicted, RC-RNN maintains an LRU queue for each label and places requests in their corresponding queue. On hit accesses, the requested data page will be moved to the head of its current queue. Data pages are demoted to lower priority queues when $5 \times cachesize$ requests are processed after entrance of data pages to the cache. Data pages in the *mean* and *late* queues are demoted to *soon* and *mean* queues, respectively. To prevent demoting hot data

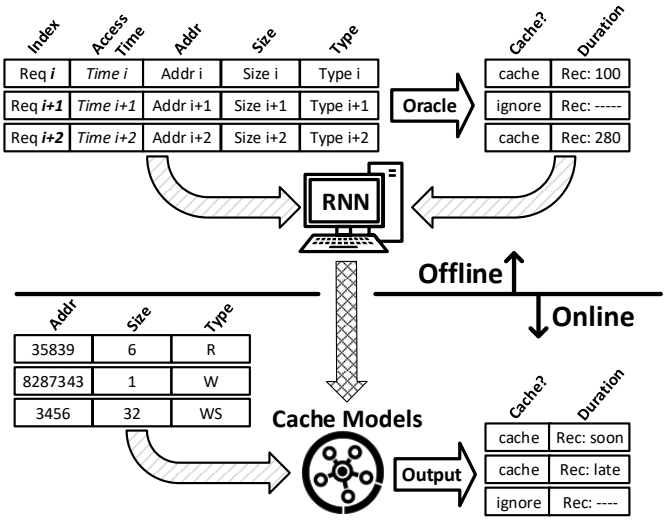


Fig. 6: Proposed Architecture and Dataflow for RC-RNN

pages, if a data page resides in top 20% of the queue, it will not be demoted. The victim for eviction will be the last data page in the *soon* queue. If the *soon* queue is empty, the *mean* and *late* queues will be searched, respectively. I/O caching architectures (unlike CPU caches or virtual memory) do not need to copy each missed access to the cache and therefore, can bypass the cache and directly supplied by HDD. This option enables RC-RNN to prevent seldom accessed data pages from entering the cache.

Regarding general limitations of RNN such as the number of distinguishable classes, we note that RC-RNN does not try to predict the address of the upcoming requests. During online workload characterization, RC-RNN predicts the workload category (e.g., Mail Server or File Server) based on the workload behavior. The number of such categories is limited and at most four categories have been identified in previous studies [3], [12], [24]. Moreover, based on the selected workload type, each I/O request is classified by a binary classification (to buffer or not to buffer) and if the model decides to buffer the data page, we use a further 3-class classification (*soon*, *mean*, and *late*). Therefore, RC-RNN does not need to have a separate category for *each* data page and the limitation of RNN in the number of categories does not affect RC-RNN.

5 Experimental Results

In this section, first the detailed experimental setup for implementing and evaluating *RC-RNN* is presented. Next, experimental results regarding tuning RNN models for both workload characterization and cache management are provided. We offer two scenarios for evaluation of *RC-RNN*: 1) static workloads and 2) reconfiguration process. During each scenario, the accuracy of workload characterization model is evaluated. Afterwards, the proposed cache management model is compared to previous studies in terms of hit ratio and SSD lifetime. Finally, the impact of reconfiguration process is discussed while system is benchmarked under multiple workloads.

5.1 Experimental Setup

TABLE 3: Experimental Setup

Device	Model
CPU	Core-i7 7500HQ
Memory	16 GB DDR3
HDD	2 TB Western Digital Red Pro HDD
SSD	512 GB Samsung SSD 850 PRO
GPU	NVIDIA TITAN X **(Only used during *offline* phase)**

RC-RNN is implemented as a kernel module based on *EnhanceIO* [39] in *Linux Kernel 4.4.0*. A user-space application receives requests from kernel module, feeds them to the RNN model, and sends back the result to the kernel module. RNN models in the *offline* phase are constructed by *Keras* [17] library and *RMSProp* [40] optimizer with the default configurations. We consider two main hyperparameters to construct RNN models: a) number of hidden units, and b) I/O sequence window length. A system with 12 GB main memory with a *NVIDIA Titan X* GPU running *Ubuntu* 16.04 has been employed for constructing RNN models. *Oracle* algorithm is implemented as a C++ module in an in-house cache simulator[1]. The complete log of the decisions made by *Oracle* for every I/O request is stored to be used as an input for *three-level deep* RNN model, which is shown in Figure 6.

Table 3 presents the complete experimental setup for both offline and online phases of our evaluations. Note that the GPU is only utilized *once* and *only* in the offline phase. In the experiments, the entire online evaluation is evaluated using *only* CPU. The online phase of RC-RNN in the experimental results uses the same hardware as LRU and LARC under all the workloads. In all experiments, the size of the SSD is set to 20% of the working set size of the workload. Table 4 shows the number of requests, read/write ratio, and the average request size of traces.

We note that the offline phase is not dependent to the running workload. Therefore, as long as the overall hardware employed remains the same, we can reuse the results of the offline phase. In case of a hardware change, the offline phase needs to be executed only *once* before the start of the online phase. In our experiments, the offline phase is executed for three hours and all the experimental results are captured during the online phase. The results of online monitoring can be used as input traces for the offline phase. This is an asynchronous operation and does not affect the performance of the online phase. Although giving feedback from the online phase to the offline phase can result in more accuracy, it was not included in the results to have a fair comparison with previous architectures.

5.2 RNN Configuration

The linear relation among I/Os has been extensively studied by the previous work, such as [3], [12], [24]. In this paper, we aim to achieve higher accuracy in classifying the sequential data of I/Os by investigating the non-linear correlation in storage workloads. To this end, we used *Recurrent Neural Networks* (RNN) that are designed to model sequential data. A usual RNN has a short-term memory, but in combination with *Long Short Term Memory* (LSTM), it will support a long-term memory. Moreover, different logistic activation functions, such as *Softmax*, *tanh* and *ReLU* can be

1. The source code of simulator and RNN models will be publicly available upon acceptance of the paper.

TABLE 4: General Characteristics of Traces

Trace	# of Reqs.	R/W Ratio	Avg. Req. Size
radius_authentication	80k	0.07	12.44 KB
radius_backed_SQL	60k	0.21	9.37 KB
mail_index	60k	2.56	42 KB
home_ikki	60k	0.84	4 KB
home_madmax	60k	0.25	4 KB
home_topgun	60k	0.11	4 KB
enterprise_tpc_1	90k	2.05	8.24 KB
MS_enterprise_ex	70k	0.16	21.6 KB
Cambridge1	65k	1.3	9.46 KB
MS build server	60k	7.5	4 KB
MS live maps	70k	all read	4 KB
web proxy	60k	4.5	4 KB
web server	60k	12.7	4 KB

employed in implementation of RNN/LSTM. However, as we have multiple classes in our workload characterization, we choose *Softmax* over other logistic alternatives. Finally, we utilize a three-layer network to reduce the overhead without sacrificing the desired performance.

The employed hyperparameters for constructing RNN models have significant impact on the accuracy of RNN models. Therefore, the optimal configuration for our RNN models should be selected before fully constructing the models. In order to determine the most efficient configuration for implemented RNN, we test the models with multiple number of hidden units under different workload characterization scenarios described in Table 1. After tuning the configurations in the workload characterization of the RNN model, evaluations are done in comparison against *Oracle* algorithm.

5.2.1 Workload Characterization Model

The input of the one-level RNN model for characterization is sequences of 100 consecutive requests containing four data fields: (1) *arrival time*, (2) *address*, (3) *size* and (4) *request type*, which are detailed in Figure 6. The output is a *softmax* layer containing the estimated probabilities of each data field for workload types.

Learning phase consists of two repetitive actions: 1) *labeling* and 2) *evaluation*. RNN (re)labels the input data sequences and evaluates its labeling by predicting the test data. In the next iteration, RNN corrects itself to reach higher accuracy in classifying data. In order to evaluate the accuracy of the workload characterization model, a part of the workload sequences are used for constructing RNN models in the *learning* phase and entirely different sequences are employed for testing the accuracy of the proposed RNN model. Figure 7 shows the accuracy of *labeling* and *evaluation* of the proposed RNN-based characterization method during *learning* phase of *Storage System* workload (listed in Table 1). In this scenario, the accuracy converges into 95% after 8 iterations where it saturates. Once the model reaches its saturation, learning phase is completed. We note that all of samples from employed I/O traces are cross-validated over a large sequence of available I/O workloads to make sure practicality of trained RNN models in general.

Figure 8 shows the accuracy of various RNN configurations under the first scenario (*Single Purpose Server*) of Table 1. Experiments reveal that in most cases, increasing I/O sequence window length results in higher accuracy as RNN has longer sequences and can build deeper and more detailed insight to learn the data access pattern of workloads. Another observation is that increasing the number of hidden units in RNN structure will not always result in higher accuracy. For instance, 100 hidden units configuration has lower accuracy than 50 hidden units configuration using a sequence window size of 100. This is due to *saturation* of model with 50 hidden units as the accuracy reaches above 99% and the model learns the input pattern completely. In this situation, higher structure overheads such as longer window length and higher number of hidden units will not result in higher accuracy. Based on such observations, we configure RNN models with 50 hidden units and window size of 100 I/O requests. Note that the single layer RNN model already achieves more than 99% accuracy. Therefore, adding extra layers to the model would only result in a more complex model without accuracy gain.

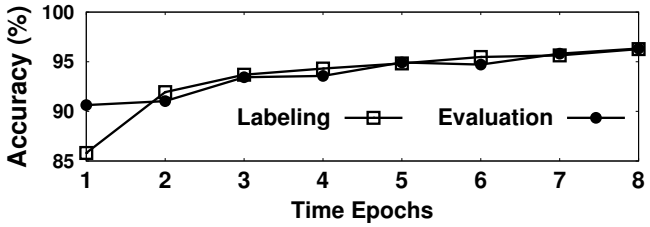


Fig. 7: Accuracy in Learning Phase of Proposed Characterization Method

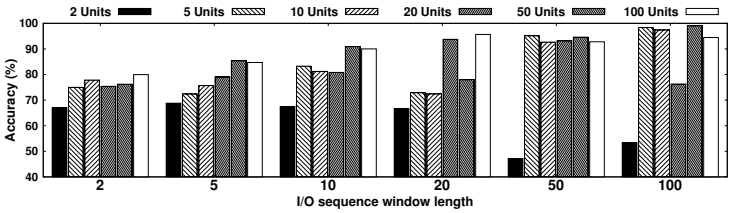


Fig. 8: Accuracy of Implemented RNN with Different Configurations for Proposed Characterization Method

5.2.2 Caching Management Model

In order to construct the RNN model for cache management, a three-level RNN is employed. Each level in the RNN model consists of 256 LSTM units. Workloads are divided into sequences of 100 consecutive requests. Each request is a five dimensional vector of *address*, *size*, *request type*, *cached/ignored*, and *duration* label, which have been presented in Section 4. The output of the RNN for each workload type is a model, which is able to mimic *Oracle* decisions under the running workload. By setting batch size to 32 and optimizer to *RMSProp* [40], we train the model and report the accuracy of decision for the workload, independently.

The contribution of this work is not limited to choosing RNN model. The effect of several novel optimizations employed in this work is shown in Table 5. *Baseline-RNN* denotes the accuracy of an RNN model without the proposed cost/benefit function. RC-RNN significantly improves the accuracy of the baseline RNN by employing a novel cost/benefit function, which considers the performance of both HDD and SSD. Additionally, RNN can *only* predict the behavior for limited number of categories.

TABLE 5: Optimizing Learning of the Oracle Using RNN

Workload Name	RC-RNN		Baseline-RNN		RCRNN imprv.
	cached	durat.	cached	durat.	
MS ent. ex.	91.23%	93.65%	62.69%	47.74%	185%
cambridge1	97.93%	95.17%	72.63%	60.34%	112%
home ikki	92.34%	100.00%	73.42%	54.76%	129%
home topgun	97.59%	94.90%	78.23%	65.52%	80%
home mdmx.	98.35%	92.43%	71.61%	58.13%	118%
Radius Auth.	89.31%	91.42%	58.40%	42.92%	225%
Radius b. sql	95.45%	72.54%	66.62%	43.49%	138%
ent. tpcE	88.31%	100.00%	83.31%	74.18%	43%
mail index	94.32%	99.53%	76.98%	90.71%	34%
MS bld. serv.	97.59%	99.90%	77.60%	72.83%	73%
MS live map	93.74%	100.00%	82.75%	98.63%	15%
msn data	97.17%	91.55%	78.14%	61.54%	85%
msn m.data	94.99%	97.07%	75.40%	68.99%	77%
varmail	96.74%	98.44%	63.47%	55.17%	172%
web proxy	93.51%	99.67%	71.20%	52.87%	147%
web server	95.23%	98.50%	78.36%	68.55%	75%

Therefore, predicting the exact data page for eviction is not possible by using RNN. However, RC-RNN classifies data pages into three categories (*soon*, *mean*, and *late*) in order to use RNN, while maintaining a high accuracy.

Table 5 shows the accuracy of the proposed model in deciding *cached* and *duration* tags for requests. In addition to the proposed model, the accuracy of a baseline RNN model without the proposed cost function has also been evaluated and presented in Table 5. The *baseline* model selects the most frequent tag value and assigns it to all requests for each iteration. For instance, if *Oracle* tags more than 50% of requests as *cached*, the static model tags all requests in the evaluation phase as *cached* and vice versa. Under workloads such as *enterprise tpcE* and *MS live maps*, the *Oracle* algorithm tags more than 80% of the requests as *ignored*. The *basic* classifier tags all requests as *ignored* and therefore, has above 80% accuracy.

Baseline classifier shows the maximum accuracy achievable by static methods. The improvement of *RC-RNN* compared to *basic* classifier is calculated based on Equation 4 and is shown in the last column of Table 5. Overall, the cost function provided by RC-RNN improves the accuracy by more than 106% on average, which shows the significance of defining the cost/benefit function.

$$RCRNN\_Imprv. = \frac{RCRNN_{cached} \times RCRNN_{dur.}}{baseln_{cached.} \times baseln_{dur.}} \quad (4)$$

5.2.3 RNN models overhead

To show the overhead of processing RNN model in runtime, we measure the latency of the RNN decision-making process (during the online phase and *without* using GPU). Using 400,000 (1000×[100×4]) samples with a batch size of 32, RC-RNN adds only 0.3ms overhead to each processed I/O request. This adds only 15% overhead to the response time of the requests. Processing of parallel I/O requests can be done simultaneously, which further reduces the actual response time overhead of RC-RNN. In the experiments, the average response time overhead is less than 10%.

Although the overhead of RC-RNN might be relatively high for low-latency SSDs, it still reduces the average response time of IOs when low-latency SSDs are employed. This is because RC-RNN significantly increases the hit ratio of the cache. Therefore, it issues less requests to HDDs, which are typically much slower than SSDs. Since the response time of HDDs is in the range of several milliseconds, each access to HDDs that is serviced by the cache will overcompensate the CPU overhead imposed by running RC-RNN. This is shown in the experimental results of Section 5.3. The benefit of reducing HDD accesses not only compensates the overhead of RC-RNN, but also reduces the average latency. Additionally, as is shown in Section 5.3, RC-RNN requires much less cache replacements on average, which again reduces the number of accesses to HDDs.

Finally, we note that the overhead of RC-RNN is not dependent on the storage capacity or IOPS. Therefore, it can be employed on top of storage systems with tens of terabytes of storage. In addition, the memory overhead of RC-RNN is negligible compared to the overhead of conventional SSD caching. For instance, RC-RNN adds only 8% additional memory overhead on top of the memory requirements for basic SSD caching with the size of 1TB.

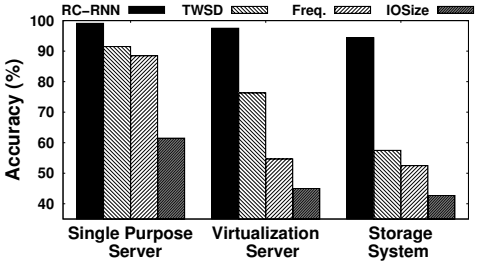


Fig. 9: Accuracy of Proposed Characterization Method Compared to Previous Studies

5.3 Static workloads Scenario

In this section, we evaluate both proposed characterization and cache management methods against previous studies in a scenario where workloads are static. The results present the performance of each method in a long-term single-purpose environment. At first, we evaluate proposed workload characterization method against state-of-the-art characterization methods. Finally, the proposed caching architecture is compared to previous studies in terms of hit ratio and SSD lifetime. Experiments in this section are done by employing a static scenario where there is no change in the running workload. To evaluate caching architectures, 16 traces from *SNIA* [14] and *Filebench* [37] have been selected. The proposed model is compared to four widely used caching architectures: (1) *LRU*, (2) *Access* [7], [8] (3) *LARC* [5], and (4) *RECA* [3]. In addition, *Oracle* has been implemented to show the maximum possible hit ratio and SSD endurance efficiency.

We evaluate the proposed characterization method in three scenarios presented in Table 1. Figure 9 shows the accuracy of the proposed characterization method compared to the previous studies. The proposed method has an accuracy of 99% in *Single Purpose Server* scenario. The results show that in *Virtualization Server* and *Storage System* scenarios, previous workload characterization methods fail to accurately predict the workload, while the proposed model is able to accurately identify workload type with more than 94% accuracy. The high accuracy of the proposed

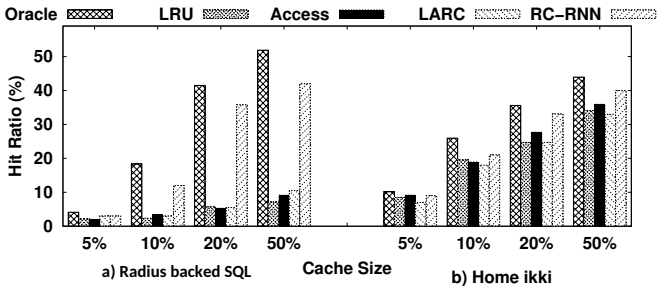


Fig. 10: Impact of Cache Size on the Overall Hit Ratio

model (up to 55% higher than previous studies) enables *RC-RNN* to identify workload type in almost all scenarios.

Larger caches can hold more data pages and therefore, have a higher hit probability. The cache hit ratio, however, does not increase linearly with the cache capacity. To show the effect of cache size on the hit ratio, we experiment several cache sizes for two workloads, shown in Fig. 10. We selected Radius and Home workloads which represent other workloads in this case. All omitted workloads behave similarly to one of the two presented ones. Employing SSD caches with a size of more than 20% of the size of working set size is not economically justified, since the cost of the SSD outweighs the performance improvement. Thus, we use 20% of the working set size as the cache size in our experiments. In our analysis, most of the cache misses are because of insufficient capacity. Cache misses due to the limited associativity are negligible in most workloads. Additionally, we measure the cache misses after the SSD is filled, and therefore, cold cache misses are not considered in our evaluations.

Figure 11 shows the hit ratio of several caching architectures under various workloads. *RC-RNN* performs 95% similar to *Oracle* on average and therefore, outperforms previous architectures with up to 7x higher hit ratio in workloads such as *Radius_backed_sql* and *Cambridge*. The hit ratio gap between *Oracle* and state-of-the-art caching architectures highly depends on the workload type. Previous studies manage cache based on *recency* and *frequency* of the I/O requests. Therefore, in workloads such as *main index* or *MSN fileserver metadata*, which have high linear temporal and spatial localities, the hit ratio of previous studies is close to *Oracle*. For most of the workloads, however, the hit ratio gap between *Oracle* and previous studies is considerably high. *RC-RNN* on the other hand, is able to achieve high hit ratio in most workloads due to the proposed RNN-based cache management model. The normalized hit ratio values of Figure 11 compared to *Oracle* are shown in Figure 12. Previous studies have less than 63% of the maximum possible hit ratio. *RC-RNN* has up to 90% of the maximum possible hit ratio, which is up to 30% higher.

In addition to hit ratio, the number of cache replacements is also analyzed. Reducing the number of cache replacements improves SSD lifetime and therefore reduces the cost of device replacements. Figure 13 presents normalized cache replacements per 100 I/Os for each caching architecture. *RC-RNN*, while having superior hit ratio compared to previous studies, lowers cache replacements in several workloads such as *cambridge* and *MS Build Server*. The number of cache replacements is reduced by up to 8x compared to *LARC*. In *Radius Authentication* workload, *RC-RNN* and *LARC* have almost the same number of cache replacements. The hit ratio of *RC-RNN*, however, is 7x higher than *LARC*. This is due to selecting more suitable data pages for caching in *RC-RNN*. In *cambridge* workload, *RC-RNN* has 8x less cache replacements while achieving more than 2.7x higher hit ratio compared to *LARC*. Our analysis shows that in this workload, working set size is larger than cache size. Therefore, *LARC* cannot benefit from data pages in cache while *RC-RNN* can detect the performance-critical data pages and keeps such data pages in cache till they are re-referenced.

Figure 14 shows the number of SSD writes per 100 I/Os. In almost all workloads, RC-RNN has a less or equal number of SSD writes, compared to LARC and RECA. This shows that RC-RNN not only improves the performance but also extends the SSD lifetime. Our analysis shows that most of the SSD lifetime improvement originates from the reduction in cache replacements.

Figure 15 shows the average response time of caching mechanisms under various workloads. RC-RNN reduces the response time of standard benchmarks by 15% on average in comparison with the best of the previous work (RECA [3]). The RC-RNN's computational overhead is only noticeable when the caching mechanism fails to deliver high hit ratios in workloads such as *"enterprise TPC-E"* and *"MS live maps backup"*, where *none* of the caching mechanisms provide any improvement. Other than that, in most of the scenarios, RC-RNN reduces the average response time of the workloads by up to 1.6 ms (43%). The computational overhead of RC-RNN (0.3 ms) has been considered in our performance measurements. These results suggest that employing RNN for SSD-based caches in storage systems can be beneficial in cases where the state-of-the-art statistical caching mechanisms fail to deliver acceptable cache performance in comparison with *Oracle*. Such scenarios are especially noticeable under workloads such as *cambridge*, *MS build server*, *Radius backed_sql*, and *Radius authentication*.

5.4 Reconfiguration Process

In this section, the proposed characterization method and cache management architecture are evaluated in case of multiple servers running several applications to simulate the behavior of enterprise storage systems. First, we compare the accuracy of the proposed RNN-based characterization method against previous work. Later, the impact of reconfigurability of the proposed architecture using multiple RNN models for cache management is analyzed. The most important parameter of workload characterization methods is the accuracy of identifying workload type when a new application starts. Workload characterization methods lose accuracy upon adding an application to the running applications. Methods having less accuracy loss are considered to be more efficient. Figure 16 shows the accuracy of various workload characterization methods upon adding a new application. While previous studies have up to 40% accuracy loss, *RC-RNN* maintains its accuracy with less than 5% accuracy loss. As shown in Figure 16, for workload characterization RC-RNN can achieve magnificently higher accuracy (up to 30%) than TWSD (the proposed method in [3]).

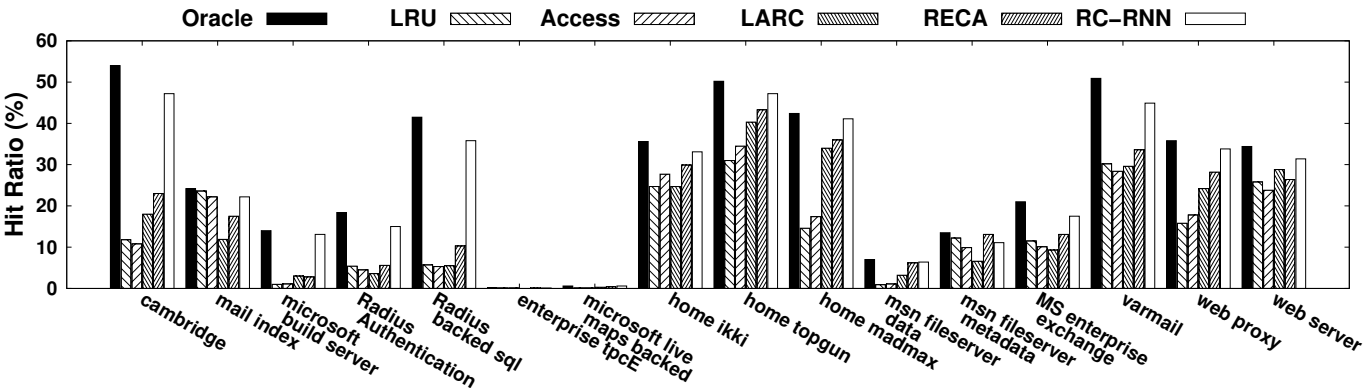


Fig. 11: Cache Performance (Hit Ratio)

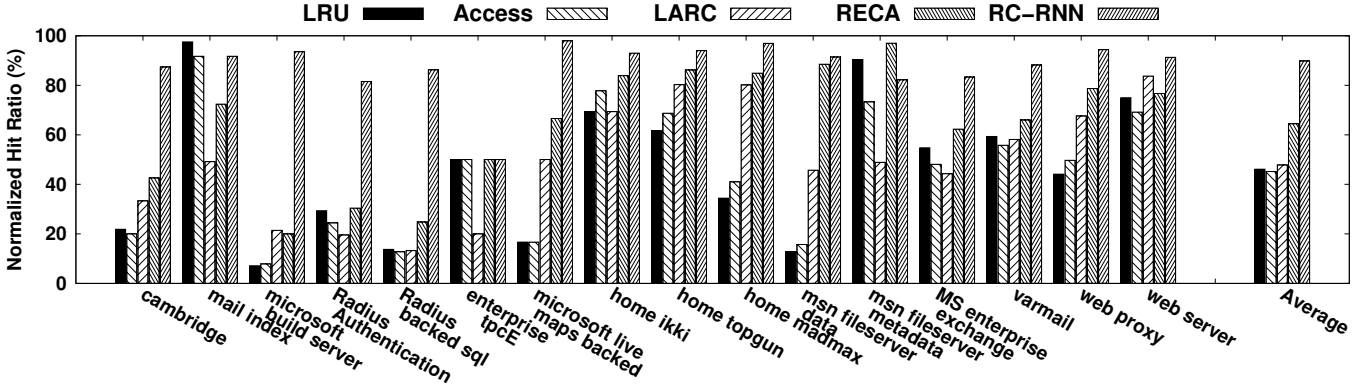


Fig. 12: Normalized Hit Ratio of Different Cache Architectures Over Oracle

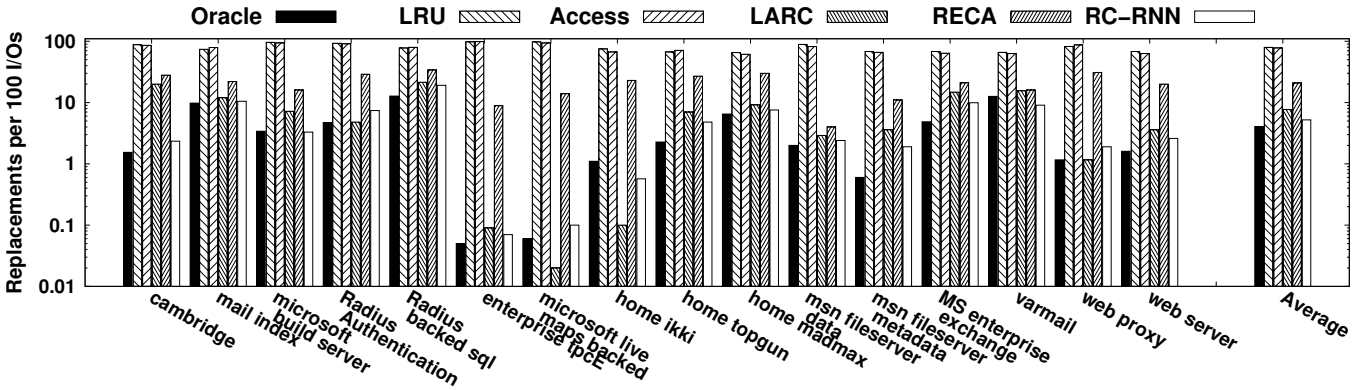


Fig. 13: Number of Cache Replacements

To evaluate the impact of reconfiguration process on the overall performance of *RC-RNN*, hit ratio is observed in case of a workload change. The workload consists of 50,000 requests from *radius_authentication* followed by 50,000 requests from *home_ikki* workload. Figure 17 shows the performance of *RC-RNN* with and without considering the reconfiguration. *Fixed_model* has *only* one model for all workload types and *RC-RNN* employs a model for each workload type. In addition, we show the reconfiguration overhead of RC-RNN in two scenarios: 1) *multiple model*, where the system has already loaded multiple caching models, and 2) *single model*, where the system requires to load the new RNN model into the main memory. In the first scenario, when the system detects a change in the workload type (using characterization model), the caching models are already loaded and *RC-RNN* maintains its efficiency after workload change. On the other hand, the number of missed blocks in the reconfiguration process of the *single model* scenario results in a small percentage of lower hit ratio in the later time periods. However, after a few periods of I/O requests, the single model RC-RNN eventually reaches the performance of the multiple model RC-RNN as shown in Figure 17. Note that *Fixed_model* fails to accurately predict the workload access pattern. Therefore, employing a model for each workload type has significant impact on the overall performance of caching architectures.

One of the drawbacks of the reconfiguration process is the memory overhead required for operating the monitoring module that analyzes the current workload every one thousand requests. In addition, in order to change the cache manager module, one needs to unload the previous and load the new RNN module into the main memory. The over-

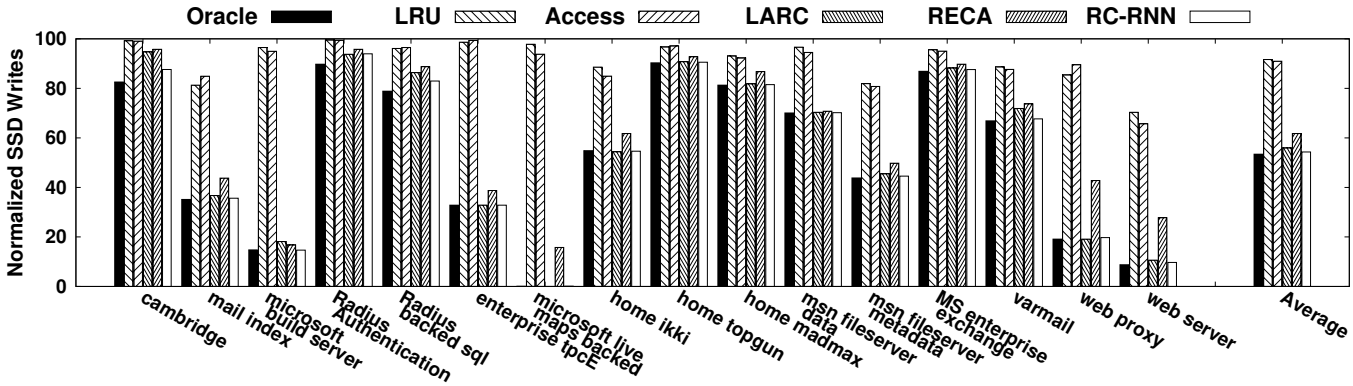


Fig. 14: Average Number of SSD Writes per 100 I/Os

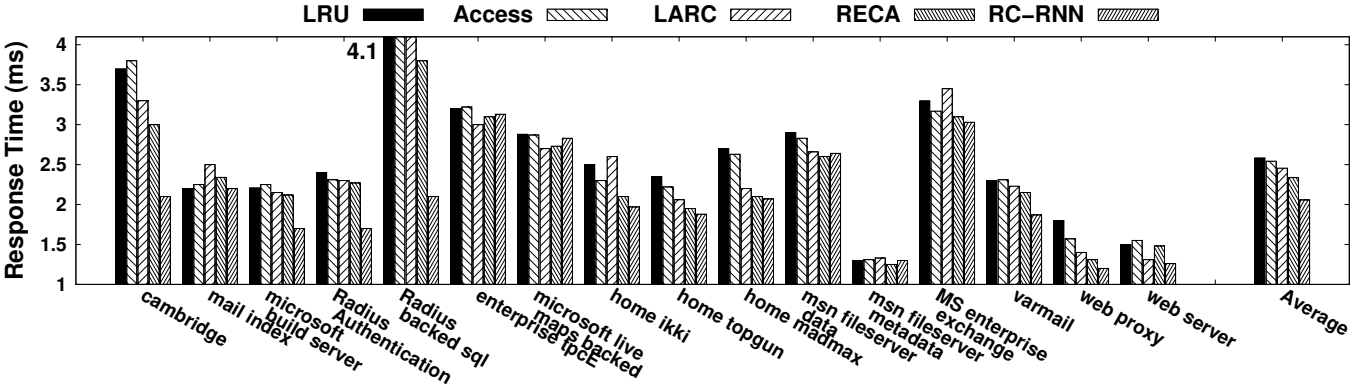


Fig. 15: Overall System Performance (Average I/O Response Time)

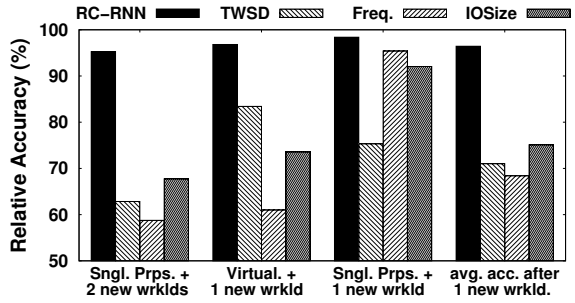


Fig. 16: Relative Accuracy of Characterization Methods upon Addition of New Workloads

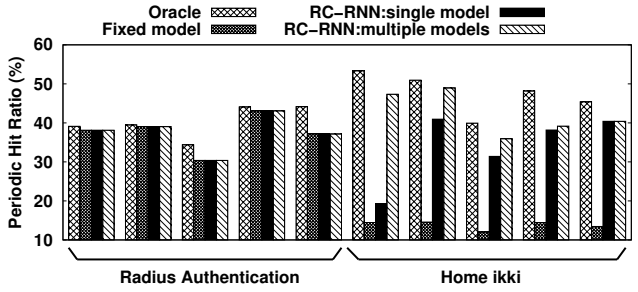


Fig. 17: Periodic hit ratio in case of workload change

head of re-evaluating benefit values is already included in the experimental results (Figure 15 and Figure 17). Since the reconfiguration is not done frequently, its prorated overhead on the total runtime is not significant.

6 Conclusion

Current state-of-the-art SSD caching architectures characterize running workload to identify performance-critical data pages. Our experiments reveal that such characterization methods fail to accurately identify workload type when several applications are running simultaneously such as in virtualization environments. Additionally, there is a significant gap between hit ratio of previous studies and maximum possible hit ratio (Oracle model). To reduce this gap and improve the accuracy of workload characterization, more complex methods such as RNNs need to be employed.

In this paper, we proposed *RC-RNN*, the first reconfigurable caching architecture using RNN to characterize workloads and manage SSD cache. *RC-RNN* employs an RNN model to identify workload type. For each workload type, a specific RNN is constructed by analyzing the decisions of *Oracle* caching architecture on various workloads. Experimental results show that *RC-RNN* can identify workload type by 40% higher accuracy compared to previous studies. *RC-RNN* also improves performance and SSD lifetime by an average of 15% and 1.7% (up to 50% and 22%), respectively. As one of the main future work, it is possible to test the effectiveness of different alternatives in hyperparameters selection of RC-RNN models.

References

[1] S. Ahmadian, F. Taheri, M. Lotfi, M. Karimi, and H. Asadi, "Investigating power outage effects on reliability of solid-state drives," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 207–212.

[2] M. Kishani, R. Eftekhari, and H. Asadi, "Evaluating impact of human errors on the availability of data storage systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 314–317.

[3] R. Salkhordeh, S. Ebrahimi, and H. Asadi, "Reca: An efficient reconfigurable cache architecture for storage systems with online workload characterization," *IEEE Transactions on Parallel & Distributed Systems (TPDS)*, no. 7, pp. 1605–1620, 2018.

[4] N. Megiddo and D. Modha, "Outperforming lru with an adaptive replacement cache algorithm," *Journal of Computer*, vol. 37, no. 4, p. 58–65, 2004.

[5] S. Huang, Q. Wei, J. Chen, C. Chen, and D. Feng, "Improving flash-based disk cache with lazy adaptive replacement," in *Proceedings of the 29th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, May 2013, p. 1–10.

[6] R. Santana, S. Lyons, R. Koller, R. Rangaswami, and J. Liu, "To arc or not to arc," in *Proceedings of the 7th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage)*, 2015, p. 4–14.

[7] Y. Klonatos, T. Makatos, M. Marazakis, M. D. Flouris, and A. Bilas, "Azor: Using two-level block selection to improve ssd-based i/o caches," in *IEEE Sixth International Conference on Networking, Architecture, and Storage (NAS)*, 2011, pp. 309–318.

[8] F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the best use of solid state drives in high performance storage systems," in *Proceedings of the International Conference on Supercomputing*. ACM, 2011, pp. 22–32.

[9] Y. Ni, J. Jiang, D. Jiang, X. Ma, J. Xiong, and Y. Wang, "S-rac: Ssd friendly caching for data center workloads," in *Proceedings of the 9th ACM International on Systems and Storage Conference*. ACM, 2016, pp. 8:1–8:12.

[10] W.-H. Kang, S.-W. Lee, B. Moon, Y.-S. Kee, and M. Oh, "Durable write cache in flash memory ssd for relational and nosql databases," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. ACM, 2014, pp. 529–540.

[11] R. Salkhordeh, M. Hadizadeh, and H. Asadi, "An efficient hybrid i/o caching architecture using heterogeneous ssds," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–12, 2018.

[12] M. Tarihi, H. Asadi, A. Haghdoost, M. Arjomand, and H. Sarbazi-Azad, "A hybrid non-volatile cache design for solid-state drives using comprehensive i/o characterization," *IEEE Transactions on Computers*, vol. 65, no. 6, pp. 1678–1691, June 2016.

[13] L. A. Belady, R. A. Nelson, and G. S. Shedler, "An anomaly in space-time characteristics of certain programs running in a paging machine," *Commun. ACM*, vol. 12, no. 6, pp. 349–353, Jun. 1969.

[14] S. N. I. A. (SNIA). (2017). [Online]. Available: http://www.snia.org

[15] S. Nishide, H. G. Okuno, T. Ogata, and J. Tani, "Handwriting prediction based character recognition using recurrent neural network," in *IEEE International Conference on Systems, Man, and Cybernetics*, Oct 2011, pp. 2549–2554.

[16] W. Ali, S. M. Shamsuddin, and A. S. Ismail, "Intelligent web proxy caching approaches based on machine learning techniques," *Decision Support Systems*, vol. 53, no. 3, pp. 565 – 579, 2012.

[17] F. Chollet, F. Rahman, G. de Marmiesse, and T. Lee, "Keras," Availabe [Online]: https://keras.io, 2015.

[18] R. Salkhordeh, H. Asadi, and S. Ebrahimi, "Operating system level data tiering using online workload characterization," *J. Supercomput.*, vol. 71, no. 4, pp. 1534–1562, Apr. 2015.

[19] R. Appuswamy, D. C. van Moolenbroek, and A. S. Tanenbaum, "Integrating flash-based ssds into the storage stack," in *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, April 2012, pp. 1–12.

[20] X. Wu and A. L. N. Reddy, "Exploiting concurrency to improve latency and throughput in a hybrid storage system," in *IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Aug 2010, pp. 14–23.

[21] J. Guerra, H. Pucha, J. Glider, W. Belluomini, and R. Rangaswami, "Cost effective storage using extent based dynamic tiering," in *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, ser. FAST'11. USENIX Association, 2011, pp. 20–20.

[22] S. Ahmadian, O. Mutlu, and H. Asadi, "Eci-cache: A high-endurance and cost-efficient i/o caching scheme for virtualized platforms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, p. 9, 2018.

[23] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, "The evicted-address filter: A unified mechanism to address both cache pollution and thrashing," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 2012, pp. 355–366.

[24] M. Tarihi, H. Asadi, and H. Sarbazi-Azad, "Diskaccel: Accelerating disk-based experiments by representative sampling," *SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 1, pp. 297–308, Jun. 2015.

[25] I. Ahmad, "Easy and efficient disk i/o workload characterization in vmware esx server," in *IEEE 10th International Symposium on Workload Characterization*, Sept 2007, pp. 149–158.

[26] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production windows servers," in *IEEE 11th International Symposium on Workload Characterization*, Sept 2008, pp. 119–128.

[27] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale i/o workloads," in *IEEE International Conference on Cluster Computing and Workshops*, Aug 2009, pp. 1–10.

[28] A. Riska and E. Riedel, "Disk drive level workload characterization," in *USENIX Annual Technical Conference (USENIX ATC)*, 2006, pp. 97–102.

[29] M. F. Arlitt and C. L. Williamson, "Web server workload characterization: The search for invariants," *SIGMETRICS Perform. Eval. Rev.*, vol. 24, no. 1, pp. 126–137, May 1996.

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[31] P. Li, J. Zhu, L. Peng, and Y. Guo, "Rnn based uyghur text line recognition and its training strategy," in *12th IAPR Workshop on Document Analysis Systems (DAS)*, April 2016, pp. 19–24.

[32] A. Graves, *Supervised sequence labelling with recurrent neural networks*. Springer, 2012, vol. 385.

[33] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015. [Online]. Available: http://arxiv.org/abs/1506.00019

[34] S. Andermatt, S. Pezold, and P. Cattin, "Multi-dimensional gated recurrent units for the segmentation of biomedical 3d-data," in *Deep Learning and Data Labeling for Medical Applications*. Springer, 2016, pp. 142–151.

[35] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *Trans. Storage*, vol. 4, no. 3, pp. 10:1–10:23, Nov. 2008.

[36] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou, "C-miner: Mining block correlations in storage systems." in *FAST*, vol. 4, 2004, pp. 173–186.

[37] A. Wilson, "The new and improved filebench," in *Proceedings of the 6th USENIX Conference on File and Stroage Technologies*, ser. FAST. USENIX Association, 2008.

[38] M. Gheisari, G. Wang, and M. Z. A. Bhuiyan, "A survey on deep learning in big data," in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, vol. 2. IEEE, 2017, pp. 173–180.

[39] STEC. (2017) Enhanceio ssd caching software. [Online]. Available: https://github.com/stecinc/EnhanceIO

[40] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

Shahriar Ebrahimi received his B.Sc. degree from Sharif University of Technology, Tehran, Iran, in 2014, and the M.Sc. degree from the same university in 2016, both in computer engineering. Currently, He is a PhD candidate of computer engineering in Sharif University of Technology. His research interests include data storage systems, operating systems, and cryptographic engineering.

Reza Salkhordeh received his B.Sc. from Ferdowsi University of Mashhad in 2011, M.Sc and Ph.D. from Sharif University of Technology (SUT) in 2013 and 2018, respectively. He was a member of PPoPP'18 artifact evaluation committee and a reviewer for Elsevier Microprocessors and Microsystems (2016) and Transactions on Computer-Aided Design of Integrated Circuits and Systems (2018). He was a member of Iran's National Elites Foundation. His research interests are storage systems design, solid-state drives, non-volatile memories, operating systems design, and caching architectures.

Seyed Ali Osia received his B.Sc. degree in Software Engineering from Sharif University of Technology in 2014. He is currently a Ph.D. candidate at the department of computer engineering, Sharif University of Technology. His research interests includes Statistical Machine Learning, Deep Learning, Privacy and Computer Vision.

Ali Taheri received his B.Sc. degree in Software Engineering from Shahid Beheshti University in 2015. He received his M.Sc. degree in Artificial Intelligence from Sharif University of Technology in 2017. His research interests includes Deep Learning and Privacy.

Hamid R. Rabiee (SM'07) received his Ph.D. in Electrical and Computer Engineering from Purdue University, West Lafayette, IN, in 1996. From 1993 to 1996 he was a Member of Technical Staff at AT&T Bell Laboratories. From 1996 to 1999 he worked as a Senior Software Engineer at Intel Corporation. He was also with PSU, OGI and OSU universities as an adjunct professor of Electrical and Computer Engineering from 1996-2000. Since September 2000, he has joined Sharif University of Technology, Tehran, Iran. He was also a visiting professor at the Imperial College of London for the 2017-2019 academic years. His research interests include statistical machine learning, Bayesian statistics, data analytics and complex networks with applications in social networks, multimedia systems, cloud and IoT privacy, bioinformatics, and brain networks.

Hossein Asadi (M'08, SM'14) received the PhD degree in electrical and computer engineering from Northeastern University, Boston, MA, USA, in 2007. He was with EMC Corporation, Hopkinton, MA, as a research scientist and senior hardware engineer, from 2006 to 2009. He has been with the Department of Computer Engineering, SUT, since 2009, where he is currently a full professor. His current research interests include data storage systems and networks, solid-state drives, operating system support for I/O and high-performance computing. More recently, he received the Best Paper Award at IEEE/ACM Design, Automation, and Test in Europe (DATE) in 2019. He has served as a guest editor of IEEE Transactions on Computers, an Associate Editor of Microelectronics Reliability, a Program Co-Chair of CADS2015, and the Program Chair of CSI National Computer Conference (CSICC2017). He is a senior member of the IEEE.