Learning to Drive Software-Defined Storage

Jian Huang Daixuan Li Jinghan Sun
Systems Platform Research Group
University of Illinois at Urbana-Champaign

Abstract

Thanks to the development of manufacturing technology, storage devices such as solid-state drives (SSDs) are becoming highly customizable to meet the ever-increasing demands on storage performance and capacity for different applications (i.e., software-defined storage). However, it is challenging to develop optimized storage devices with current human-driven systems-building approaches, due to the complicated storage stack. In this paper, we present learning-based approaches to facilitating the development of software-defined storage. To accelerate the manufacturing of efficient storage devices, we enable the automated learning of optimized hardware specifications for developing customized storage devices for specific application types. Our preliminary study shows that utilizing learning-based techniques to drive the development of software-defined storage is promising.

1 Introduction

For decades of development, storage systems have been the backbone of modern computer systems to support diverse data-intensive applications [10, 17]. They have been developed into a complicated system that involves the development of storage devices like flash-based solid-state drives (SSDs), storage software, and application-level data stores. In order to rapidly meet the ever-increasing performance and efficiency requirements of applications, the storage system needs to recognize workload changes and adapt quickly across the full stack in a coordinated fashion. However, it is hard to achieve this with current human-driven system-building approaches, due to the complicated storage stack. For instance, the device efficiency of an SSD is determined by hundreds of parameters, making it hard for hardware/firmware engineers to manually tune them. And, storage devices are usually designed and produced for generic workloads, which inevitably causes resource inefficiency in real-world deployments, such as cloud storage services that normally host a specific category of applications like Database as a Service (DBaaS) [3, 6, 5, 15]. Although the mature manufacturing techniques and fabrication process enable fast device manufacturing, we are lacking efficient frameworks that can instantly transfer application demands and characteristics into storage hardware specifications. And storage devices today lack the capability of learning diverse workload patterns, and they cannot make runtime adaptations for achieving the maximum storage performance.

In this paper, we present an effective learning framework that can facilitate the development of customized storage devices like flash-based SSDs for specific application types with automated learning of optimized hardware specifications. This is driven by the observations that storage device manufacturing techniques have become mature, while device configuration is still the bottleneck of the storage device development. With our proposed framework, we can enable storage vendors to identify optimal device configurations with much less time and effort.

Specifically, we exploit both supervised and unsupervised machine learning (ML) techniques to drive the hardware configurations for new SSDs, with specified hardware constraints. Given a storage workload, we will recommend an optimized SSD configuration that delivers optimized storage performance. It leverages linear regression techniques to expose the device configurations that have the strongest correlation to the storage performance. To present reasonable device configurations, we formulate different types of hardware parameters in the SSD, translate them into the vectors in the

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

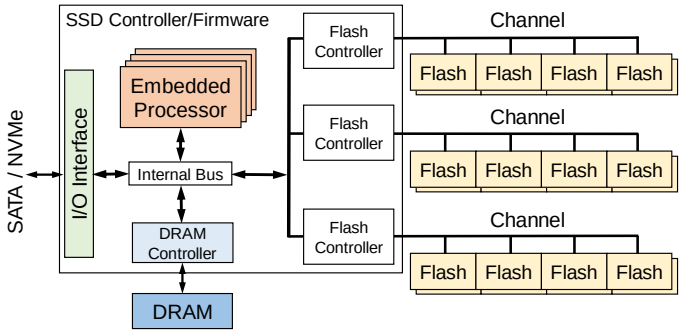

Figure 1: Internal architecture of flash-based SSDs.

ML model, and utilize learning techniques to explore the optimization space and identify optimized options, with specified constraints such as SSD capacity, device interfaces (NVMe or SATA), flash memory types, and power budget of the storage device. To reduce the execution time of learning an optimized SSD configuration while ensuring the learning accuracy, we develop pruning algorithms to identify the most important hardware parameters in SSDs for different workloads. Our preliminary study leads to interesting insights. We turn them into learning rules that can aid us to prioritize the optimization strategies when producing new SSDs. For instance, (1) not all SSD parameters are equal, the layout arrangement of flash chips is an important factor for storage performance; (2) with different targets and configuration constraints, the tuning procedure of device configurations are different, and for each parameter, its correlation with SSD performance is also different; (3) not all parameters are sensitive to storage performance, and some of them can be configured as the same as commodity SSDs today. By ordering these parameters based on their impact on storage performance, we can significantly reduce the learning time of an optimized hardware configuration.

2 Background and Motivation

SSD Architecture. We present the internal architecture of a typical SSD in Figure 1. An SSD consists of five major components: a set of flash memory packages, an SSD controller having embedded processors like ARM, off-chip DRAM (SSD DRAM), flash controllers, and the I/O interface that includes SATA and NVMe protocols [8, 13, 21]. The flash packages are organized in a hierarchical manner. Each SSD has multiple channels where each channel can process read/write commands independently. Each channel is shared by multiple flash packages. Each package has multiple flash chips. Within each chip, there are multiple planes. Each plane includes multiple flash blocks, and each block has multiple flash pages. The page size varies in different SSDs. When a free flash page is written once, that page is no longer available for future writes until that page is erased. However, erase operation is expensive and performed at block granularity. As each flash block has limited endurance, it is important for blocks to age uniformly (i.e., wear leveling). Modern SSD controllers employ out-of-place write, GC, and wear leveling to overcome these shortcomings and maintain indirections for the address translation in their flash translation layer (FTL).

Software-Defined Storage. With the increasing demands on storage performance from applications, we have seen a trend that modern storage systems are embracing software-defined hardware techniques [9]. This allows upper-level applications to achieve maximum performance benefits and resource efficiency with customized storage devices. For instance, the recent development of software-defined flash [29, 15] enables platform operators to customize the number of flash channels and chips in an SSD, with the cooperation with SSD vendors.

For applications, such as Database-as-a-Service [34], web services [2], web search [15], and batch data analytics [12], their workloads can be highly classified. For instance, we use our proposed learning-based workload characterization approach to study the storage traces from a set of popular application workloads. Our experiments show that each workload type has its unique characteristics, and I/O traces from the same workload type have similarities in their data access pattern, as we used Principal Component Analysis to map them into two dimensions in Figure 2. This shows that it is feasible to customize storage devices for a specific category of applications.

SSD Manufacturing Procedure. According to the interviews with SSD product managers and our discussions with SSD vendors, finalizing SSD configurations is on the critical path in the SSD design.

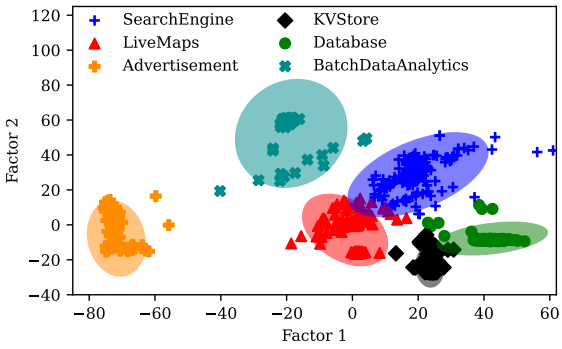

Figure 2: A clustering of popular storage workloads.

These configurations are usually determined by the requirements from applications and customers. Without them, the SSD development cannot proceed to the manufacturing stage.

To finalize the SSD specifications, a simple approach is to test and profile application workloads with different hardware configurations. However, this is not scalable as we target different application workloads. With the confirmation from SSD vendors, there are more than a hundred tunable parameters in an SSD. Given an application workload, it is challenging for developers to test all the combinations of the device parameters. And likewise, given a new SSD specification, it requires significant manual effort to quantify the effectiveness of the selected specifications. In this work, we use the learning-based approach to automate the SSD hardware configurations.

Our goal is to enable the automated learning of SSD configurations for a specific workload with learning techniques: (1) it should generate an optimized SSD configuration for a target workload, while this hardware configuration has minimal negative impact on other non-target workloads; (2) it should identify an optimized SSD configuration in a short time, without introducing much computation overhead; (3) it should scale to support diverse target workloads as well as different device constraints.

3 Automated Learning of Storage Device Configurations

We leverage linear regression techniques to expose device specifications that have the strongest correlation with the storage efficiency. To present reasonable device specifications, we formulate different types of hardware parameters in the SSD, translate them into the vectors of the learning models, and learn the optimized configurations with specified constraints, such as storage capacity, SSD interfaces (NVMe or SATA), and power budget.

To minimize the learning time of an optimized SSD configuration, we develop pruning algorithms to pinpoint the most correlated device parameters for the storage efficiency, and extract a set of learning rules to guide our learning procedure. We will also maintain a configuration database to store the learned workloads and the corresponding SSD specifications. Therefore, we can utilize the learned experiences for new workloads.

Learning-based Workload Clustering. To develop the learning-based workload clustering, we first partition each I/O trace into small windows. According to our study of diverse workloads, we use 3,000 trace entries in each window by default. This is because fewer entries may lose the unique data access patterns of the trace. The trace information used to conduct the workload characterization include the I/O timestamp, I/O size, block address, and operation types. We focus on their relative values by normalizing them with the values of the starting entry of each window. Since each I/O entry has multiple fields, such as I/O type, address, accessed data size, and timestamp, we use Principal Component Analysis (PCA) [18] to transfer each window of the I/O traces into two dimensions. This will simplify the learning of I/O workload patterns by transferring the complex characteristics of I/O traces into fewer dimensions. After that, we use k-means to cluster these data points. We calculate the distance between the center of the examined data points and the center of an existing cluster. If the distance is below a configurable threshold (10 by default), we claim that the new workload belongs to this cluster. If we cannot identify a similar cluster, it will create a new cluster. As shown in Figure 2,

our learning-based workload clustering can successfully identify a cluster of storage workloads that belong to the same or similar workload type.

Transfer SSD Configuration Learning into ML Models. To model the SSD configurations via ML techniques, we formulate them into three major parts in the ML model: (1) the efficiency metrics used as the optimization targets for SSDs; (2) SSD hardware configurations that can be vectorized as parameters in an ML model; and (3) the constraints (e.g., the SSD capacity and power budget) that bound the optimization space of the ML model. As for storage efficiency, we focus on the storage latency and throughput, under the constraints of the SSD capacity and power budget. To quantify whether a SSD configuration delivers optimized performance or not, we use reference performance as the baseline (e.g., the latency and throughput obtained from a commercial SSD's configurations), and the relative performance improvements as the evaluation metrics. Given a workload *W*, we have a target configuration *target*, and a reference configuration *refer*, we have the following performance optimization goal:

$$\textbf{\textit{Performance}}_{\textbf{\textit{W}}}(\textbf{\textit{target}}) = \quad (1-\alpha) \times log(\frac{\textbf{\textit{Latency}}_{\textbf{\textit{refer}}}}{\textbf{\textit{Latency}}_{\textbf{\textit{target}}}}) + \alpha \times log(\frac{\textbf{\textit{Throughput}}_{\textbf{\textit{target}}}}{\textbf{\textit{Throughput}}_{\textbf{\textit{refer}}}}) \tag{1}$$

where α is a tunable coefficient factor for balancing the latency and throughput at a proper scale. We set $\alpha = 0.5$ by default in , based on our study of different coefficients.

To represent SSD hardware specifications in the ML models, we transfer them into four types of ML parameters and use different ways to set their values. They include *continuous*, *discrete*, *boolean*, and *categorical* parameters. The continuous parameters include cached mapping table (CMT) size and over-provisioning ratio for garbage collection (GC); the discrete parameter include the SSD DRAM capacity, I/O queue depth, and page size; the boolean parameters in the ML model to indicate whether a function or feature (e.g., statistic wear leveling and greedy GC) will be enabled in the SSD or not; and the categorical parameter can be converted to the dummy variable, for example, there are 16 possible values for the plane allocation scheme, we create a list with the length of 16.

Learning-based Parameter Pruning. Modern SSDs usually have hundreds of hardware specifications or parameters. We first use a coarse-grained pruning method that adjusts the values of continuous and discrete numerical device parameters with large stride length. But we ensure the values of these device parameters are configured in a reasonable range, and still satisfy the configuration constraints. After eliminating the insensitive parameters with the coarse-grained pruning, we continue the parameter pruning with a fine-grained approach. We employ the linear regression technique LASSO to identify the linear correlations between the SSD parameters and performance. We use the regression coefficient scores generated by LASSO to guide the tuning order for different parameters for further improving the tuning efficiency.

We set a regression space by maintaining the constraints of the SSD capacity and power budget. We vary the values of SSD device parameters, and measure the regression coefficient for each device parameter. A higher regression coefficient score of a parameter means it has a stronger correlation with the SSD performance. We abandon the parameter whose score is below a threshold. Therefore, we can focus on the parameter tuning for the important ones.

Automated Learning of SSD Configurations. We now develop the ML model to learn optimized SSD configurations. Given a workload, we will first use the configurations stored in the database as the initial configuration set, and leverage both Gaussian Process Regression (GPR) and discrete Stochastic Gradient Descent (SGD) algorithms to learn different configurations. For each learned configuration, we will use a cycle-accurate SSD simulator to validate its performance and energy consumption until the model converges (i.e., an optimized SSD configuration is identified). We conduct the learning procedure of SSD configurations in four steps: (1) For a new workload, we will use the learning-based workload clustering to cluster the workload, and look up the learned configurations for the corresponding workload cluster in the database. We will use these configurations and their delivered performance to initialize the ML model. (2) In order to check the effectiveness of these configurations, we develop a grading mechanism with the goal of unifying different performance metrics. To achieve the maximal optimizations for both data access latency and throughput, we use the Formula 1 as the learning goal. (3) To identify optimized configurations, we first identify the top three best configurations (i.e., the configurations whose grades rank at the top) from the learned configurations, and randomly selects one as the search root. In the gradient descent process, we expand the search space from the root by checking all the adjacent configurations under configuration constraints (e.g., SSD capacity) in each searching iteration. Given the capacity constraint, we

Table 1: Performance of learned configurations for NVMe-based MLC SSDs (normalized to Intel 750 SSD). We list the target workloads in the first row. The first column shows the non-target workloads. The numbers represent the speedup of the storage latency/throughput. Our learning-based approach delivers the best performance for target workloads (bold numbers).

Target workload →	Recomm	KVStore	Database	WebSearch	BatchAnalytics	CloudStorage	LiveMaps
Recomm	**1.25/1.26**	1.02/1.12	1.20/1.21	0.99/1.04	1.19/1.21	1.10/1.13	1.18/1.17
KVStore	1.10/1.10	**1.22/1.23**	1.10/1.11	1.13/1.13	1.17/1.18	1.12/1.13	1.11/1.13
Database	1.15/1.15	1.07/1.07	**1.20/1.20**	0.99/0.99	1.15/1.14	1.02/1.02	1.13/1.13
WebSearch	0.87/0.88	1.24/1.26	0.72/0.73	**1.61/1.61**	0.75/0.76	1.01/1.00	1.00/1.00
BatchAnalytics	1.14/1.13	1.09/1.10	1.13/1.13	1.14/1.15	**1.26/1.27**	1.08/1.08	1.12/1.11
CloudStorage	1.34/1.36	1.15/1.16	1.26/1.25	1.04/1.05	1.21/1.22	**1.48/1.46**	1.15/1.14
LiveMaps	1.16/1.16	1.04/1.06	1.18/1.18	0.97/0.98	1.13/1.14	1.07/1.06	**1.24/1.23**

tune one relevant parameter at one time, and tune the parameters following the ranking of their absolute coefficient scores for improved learning efficiency. (4) Once we obtain the selected device configuration, we will conduct the efficiency validation using a cycle-accurate SSD simulator. The validation results will be updated to the model for improving the accuracy of next search.

Preliminary Study Results. We implemented our proposed learning framework using PyTorch [30], the *scikit-learn* tool [32], and a production-level SSD simulator MQSim [33]. We perform experiments with a variety of storage workloads. Beyond these workloads, we also employ a set of new storage workloads that have not been learned to examine its efficiency and generality. They cover various workloads that include key-value stores, databases, map services, advertisement recommendations, batch data analytics, and enterprise file servers [37, 20, 19, 24, 16]. Our experimental results show that we can deliver an SSD configuration that can achieve 1.20–1.61× performance improvement for a target workload, and 1.09× improvement on average for non-target workloads, compared to the configurations specified in the released commodity SSDs.

4 Related Works

Most recently, researchers have started to leverage machine learning techniques to solve system optimization problems, such as the task scheduling [31, 40, 36], cluster resource management [27, 39, 7, 11, 4], performance optimizations [14, 23, 26, 41], data management [35, 22, 1, 25], and others [28, 38]. However, few studies conduct a systematic investigation of applying the learning techniques to develop SSD devices. To the best of our knowledge, is the first work that utilizes the learning techniques to enable the automated tuning of SSD specifications. We believe it will not only benefit SSD vendors and manufacturers but also platform operators such as those for cloud services and data centers. A major limitation of applying ML to solve systems problems is its training and tuning procedure are time consuming. As the ML model becomes complicated (such as those developed based on deep neural networks), it would be challenging sometimes to obtain explainable learning results. Therefore, aimed to use simple yet effective learning algorithms to ensure its learning procedure is explainable and reasonable.

5 Conclusion and Future Work

In this paper, we demonstrate a case study of using learning-based approaches to drive the design of software-defined storage devices. Our preliminary study shows that our learned SSD configurations can maximize the performance improvement for a target workload. As for the future work, we will enhance the proposed learning framework. Beyond this, the authors are developing systems and architecture techniques in combination with learning-based approaches to facilitate the development and deployment of software-defined storage across the entire system stack.

References

[1] Aken, D.V., Yang, D., Brillard, S., Fiorino, A., Zhang, B., Billian, C., Pavlo, A., 2021. An Inquiry into Machine Learning-based Automatic Configuration Tuning Services on Real-World Database Management Systems. Proceedings of the VLDB Endowment 14, 1241–1253.

[2] Amazon, 2021. Aws web services. `https://aws.amazon.com/`.

[3] Amazon Relational Database Service, 2022. https://aws.amazon.com/rds/.

[4] Ambati, P., Goiri, I., Frujeri, F., Gun, A., Wang, K., Dolan, B., Corell, B., Pasupuleti, S., Moscibroda, T., Elnikety, S., Fontoura, M., Bianchini, R., 2020. Providing slos for resource-harvesting vms in cloud platforms, in: Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20).

[5] Azure DocumentDB, 2022. https://azure.microsoft.com/en-us/services/documentdb/.

[6] Azure SQL Database, 2022. https://azure.microsoft.com/en-us/services/sql-database/.

[7] Bianchini, R., Fontoura, M., Cortez, E., Bonde, A., Muzio, A., Constantin, A.M., Moscibroda, T., Magalhaes, G., Bablani, G., Russinovich, M., 2020. Toward ml-centric cloud platforms. Communication of ACM 63.

[8] Chen, F., Lee, R., Zhang, X., 2011. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing, in: Proceedings of the IEEE 17th International Symposium on High Performance Computer Architecture (HPCA'11), IEEE.

[9] Compute, O., 2021. The open compute project. https://www.opencompute.org/.

[10] Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R., Woodford, D., 2012. Spanner: Google's globally-distributed database, in: Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI'12), Hollywood, CA.

[11] Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., Bianchini, R., 2017. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms, in: Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17), Shanghai, China.

[12] Dean, J., Ghemawat, S., 2008. Mapreduce: simplified data processing on large clusters. Communications of the ACM 51, 107–113.

[13] Gupta, A., Kim, Y., Urgaonkar, B., 2009. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings, in: Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09), Washington, DC, USA.

[14] Hao, M., Toksoz, L., Li, N., Halim, E.E., Hoffmann, H., Gunawi, H.S., 2020. LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network, in: Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20).

[15] Huang, J., Badam, A., Caulfield, L., Nath, S., Sengupta, S., Sharma, B., Qureshi, M.K., 2017a. FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs, in: Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17), Santa Clara, CA.

[16] Huang, J., Badam, A., Caulfield, L., Nath, S., Sengupta, S., Sharma, B., Qureshi, M.K., 2017b. FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs, in: Proceedings of the 15th Usenix Conference on File and Storage Technologies (FAST'17), Santa clara, CA.

[17] Huang, J., Badam, A., Chandra, R., Nightingale, E.B., 2015. WearDrive: Fast and Energy-Efficient Storage for Wearables, in: Proc. 2015 USENIX Annual Technical Conference (USENIX ATC'15), Santa Clara, CA.

[18] Jaadi, Z., 2022. A Step-by-Step Explanation of Principal Component Analysis (PCA). URL: https://builtin.com/data-science/step-step-explanation-principal-component-analysis.

[19] Kavalanekar, S., Worthington, B., 2007. Microsoft enterprise traces (SNIA IOTTA trace 148), in: Kuenning, G. (Ed.), SNIA IOTTA Trace Repository. Storage Networking Industry Association. URL: http://iotta.snia.org/traces/block-io/131?only=148.

[20] Kavalanekar, S., Worthington, B., 2008. Microsoft enterprise traces (SNIA IOTTA trace set 130), in: Kuenning, G. (Ed.), SNIA IOTTA Trace Repository. Storage Networking Industry Association. URL: http://iotta.snia.org/traces/block-io?only=130.

[21] Koo, G., Matam, K.K., I, T., Narra, H.V.K.G., Li, J., Tseng, H.W., Swanson, S., Annavaram, M., 2017. Summarizer: Trading communication with computing near storage, in: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'17), Cambridge, Massachusetts.

[22] Kraska, T., Minhas, U.F., Neumann, T., Papaemmanouil, O., Patel, J.M., Re, C., Stonebraker, M., 2021. ML-In-Databases: Assessment and Prognosis. IEEE Computer Society Technical Committee on Data Engineering .

[23] Kurniawan, D.H., Toksoz, L., Badam, A., Emami, T., Madireddy, S., Ross, R.B., Hoffmann, H., Gunawi, H.S., 2021. Ionet: Towards an open machine learning training ground for i/o performance prediction. Technical Report .

[24] LASS, . Search engine i/o trace, in: UMass Trace Repository. Laboratory for Advanced System Software. URL: http://traces.cs.umass.edu/index.php/Storage/Storage.

[25] Li, G., Zhou, X., Li, S., Gao, B., 2019. Qtune: A query-aware database tuning system with deep reinforcement learning. Proceedings of the VLDB Endow. 12.

[26] Maas, M., Andersen, D.G., Isard, M., Javanmard, M.M., McKinley, K.S., Raffel, C., 2020. Learning-based memory allocation for c++ server workloads, in: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20), Lausanne, Switzerland.

[27] Mao, H., Schwarzkopf, M., Venkatakrishnan, S.B., Meng, Z., Alizadeh, M., 2019. Learning scheduling algorithms for data processing clusters, in: Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM'19), Beijing, China.

[28] Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J., Songhori, E.M., Wang, S., Lee, Y., Johnson, E., Pathak, O., Bae, S., Nazi, A., Pak, J., Tong, A., Srinivasa, K., Hang, W., Tuncer, E., Babu, A., Le, Q.V., Laudon, J., Ho, R.C., Carpenter, R., Dean, J., 2020. Chip placement with deep reinforcement learning. CoRR abs/2004.10746.

[29] Ouyang, J., Lin, S., Jiang, S., Hou, Z., Wang, Y., Wang, Y., 2014. SDF: Software-defined Flash for Web-scale Internet Storage Systems, in: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14), Salt Lake City, UT.

[30] PyTorch Team, 2021. Pytorch: From research to production. https://pytorch.org/.

[31] Qiu, H., Banerjee, S.S., Jha, S., Kalbarczyk, Z.T., Iyer, R.K., 2020. FIRM: An intelligent fine-grained resource management framework for slo-oriented microservices, in: Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20).

[32] Scikit-learn, 2021. scikit-learn: Machine learning in python. https://scikit-learn.org/stable.

[33] Tavakkol, A., Gómez-Luna, J., Sadrosadati, M., Ghose, S., Mutlu, O., 2018. Mqsim: A framework for enabling realistic studies of modern multi-queue SSD devices, in: Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18), Oakland, CA.

[34] TrustRadius, 2021. Database-as-a-service. https://www.trustradius.com/database-as-a-service-dbaas.

[35] Van Aken, D., Pavlo, A., Gordon, G.J., Zhang, B., 2017. Automatic database management system tuning through large-scale machine learning, in: Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD'17).

[36] Wang, Y., Arya, K., Kogias, M., Vanga, M., Bhandari, A., Yadwadkar, N.J., Sen, S., Elnikety, S., Kozyrakis, C., Bianchini, R., 2021. Smartharvest: Harvesting idle cpus safely and efficiently in the cloud, in: Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys'21).

[37] Yadgar, G., Gabel, M., Jaffer, S., Schroeder, B., 2018. YCSB RocksDB SSD traces (SNIA IOTTA trace set 28568), in: Kuenning, G. (Ed.), SNIA IOTTA Trace Repository. Storage Networking Industry Association. URL: http://iotta.snia.org/traces/block-io?only=28568.

[38] Yazdanbakhsh, A., Angermüller, C., Akin, B., Zhou, Y., Jones, A., Hashemi, M., Swersky, K., Chatterjee, S., Narayanaswami, R., Laudon, J., 2021. Apollo: Transferable architecture exploration. CoRR abs/2102.01723.

[39] Zhang, D., Dai, D., He, Y., Bao, F.S., Xie, B., 2020. RLScheduler: An Automated HPC Batch Job Scheduler Using Reinforcement Learning, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'20), Virtual Event.

[40] Zhong, Z., Xu, M., Rodriguez, M.A., Xu, C., Buyya, R., 2021. Machine learning-based orchestration of containers: A taxonomy and future directions. CoRR abs/2106.12739.

[41] Zhou, G., Maas, M., 2021. Learning on distributed traces for data center storage systems, in: Proceedings of the Machine Learning and Systems (MLSys'21), Austin, TX.