Towards Better Understanding of Black-box Auto-Tuning: A Comparative Analysis for Storage Systems

Zhen Cao, ***Stony Brook University;*** **Vasily Tarasov,** ***IBM Research–Almaden;***
Sachin Tiwari and Erez Zadok, ***Stony Brook University***

https://www.usenix.org/conference/atc18/presentation/cao

This paper is included in the Proceedings of the 2018 USENIX Annual Technical Conference (USENIX ATC '18).

July 11–13, 2018 • Boston, MA, USA

ISBN 978-1-939133-02-1

Open access to the Proceedings of the 2018 USENIX Annual Technical Conference is sponsored by USENIX.

Towards Better Understanding of Black-box Auto-Tuning: A Comparative Analysis for Storage Systems

Zhen Cao[1], Vasily Tarasov[2], Sachin Tiwari[1], and Erez Zadok[1]
[1]*Stony Brook University* and [2]*IBM Research—Almaden*

Abstract

Modern computer systems come with a large number of configurable parameters that control their behavior. Tuning system parameters can provide significant gains in performance but is challenging because of the immense number of configurations and complex, non-linear system behavior. In recent years, several studies attempted to automate the tuning of system configurations; but they all applied only one or few optimization methods. In this paper, for the first time, we apply and then perform comparative analysis of multiple black-box optimization techniques on storage systems, which are often the slowest components of computing systems. Our experiments were conducted on a parameter space consisting of nearly 25,000 unique configurations and over 450,000 data points. We compared these methods for their ability to find near-optimal configurations, convergence time, and instantaneous system throughput during auto-tuning. We found that optimal configurations differed by hardware, software, and workloads—and that no one technique was superior to all others. Based on the results and domain expertise, we begin to explain the efficacy of these important automated black-box optimization methods from a systems perspective.

1 Introduction

Storage is a critical element of computer systems and key to data-intensive applications. Storage systems come with a vast number of configurable parameters that control system's behavior. Ext4 alone has around 60 parameters with whopping 10^{37} unique combinations of values. Default parameter settings provided by vendors are often suboptimal for a specific user deployment; previous research showed that tuning even a small subset of parameters can improve power and performance efficiency of storage systems by as much as 9× [66].

Traditionally, system administrators pick parameter settings based on their expertise and experience. Due to the increased complexity of storage systems, however, manual tuning does not scale well [87]. Recently, several attempts were made to automate the tuning of computer systems in general and storage systems in particular [71, 78]. Black-box auto-tuning is an especially popular approach thanks to its obliviousness to a system's internals [86]. For example, Genetic Algorithms (GA) were applied to optimize the I/O performance of HDF5-based applications [5] and Bayesian Optimization (BO) was used to find a near-optimal configuration for Cloud VMs [3]. Other methods include Evolutionary Strategies [62], Smart Hill-Climbing [84], and Simulated Annealing [21]. The basic mechanism behind black-box auto-tuning is to iteratively try different configurations, measure an objective function's value—and based on the previously learned information—select the next configurations to try. For storage systems, objective functions can be throughput, energy consumption, purchase cost, or even a formula combining different metrics [50, 71]. Despite some appealing results, there is no deep understanding how exactly these methods work, their efficacy and efficiency, and which methods are more suitable for which problems. Moreover, previous works evaluated only one or few algorithms at a time. In this paper, for the first time (to the best of our knowledge), we apply and analytically compare *multiple* black-box optimization techniques on storage systems.

To demonstrate and compare these algorithms' ability to find (near-)optimal configurations, we started by exhaustively evaluating several storage systems under four workloads on two servers with different hardware and storage devices; the largest system consisted of 6,222 unique configurations. Over a period of 2+ years, we executed 450,000+ experimental runs. We stored all data points in a relational database for query convenience, including hardware and workload details, throughput, energy consumption, running time, etc. In this paper, we focused on optimizing for throughput, but our methodology and observations are applicable to other metrics as well. We will release our dataset publicly to facilitate more research into auto-tuning and better understanding of storage systems.

Next, we applied several popular techniques to the collected dataset to find optimal configurations under various hardware and workload settings: Simulated Annealing (SA), Genetic Algorithms (GA), Bayesian Optimization (BO), and Deep Q-Networks (DQN). We also tried Random Search (RS) in our experiments, which showed surprisingly good results in previous research [8]. We compared these techniques from various aspects, such as the ability to find near-optimal configurations, convergence time, and instantaneous system throughput during auto-tuning. For example, we found that several techniques were able to converge to good configurations given enough time, but their efficacy differed a lot. GA and BO outperformed SA and

DQN on our parameter spaces, both in terms of convergence time and instantaneous throughputs. We also showed that hyper-parameter settings of these optimization algorithms, such as mutation rate in GA, could affect the tuning results. We further compared the techniques across three behavioral dimensions: (1) *Exploration*: how much the technique searches the space randomly. (2) *Exploitation*: how much the technique leverages the "neighborhood" of the current candidate or previous search history to find even better configurations. (3) *History*: how much data from previous evaluations is kept and utilized in the overall search process. We show that all techniques employ these three key concepts to varying degrees and the trade-off among them plays an important role in the effectiveness and efficiency of the algorithms. Based on our experimental results and domain expertise, we provide explanations of efficacy of such black-box optimization methods from a storage perspective. We observed that certain parameters would have a greater effect on system performance than others, and the set of dominant parameters depends on file systems and workloads. This allows us to provide more insights into the auto-tuning process.

Auto-tuning storage systems is fairly complex and challenging. We made several necessary assumptions and simplifications while collecting our exhaustive data, which we detail in §3. Therefore, some of our observations might differ when applied to production systems. However, the main purpose of this paper is *not* to provide a complete solution; rather, we focus on comparing and understanding the efficacy of several popular optimization techniques when applied to storage systems. We believe this paves the way for practical auto-tuning storage systems in real-time.

The rest of the paper is organized as follows. §2 explains the challenges of auto-tuning storage systems and provides necessary background knowledge. §3 describes our experimental methodology and environments. In §4 we applied multiple optimization methods and evaluated and explained them from various aspects. §5 covers limitations and future plans for our work. §6 lists related work. We conclude and discuss future directions in §7.

2 Background

Storage systems are often a critical component of computer systems, and are the foundation for many data-intensive applications. Usually they come with a large number of configurable options that could affect or even determine the systems' performance [12, 74], energy consumption [66], and other aspects [47, 71]. Here we define a *parameter* as one configurable option, and a *configuration* as a combination of parameter values. For example, the parameter *block_size* of Ext4 can take 3 values: 1K, 2K, and 4K. Based on this, *[journal_mode="data=writeback", block_size=4K, inode_size=4K]* is one *configuration* with 3 specific parameters: *journal mode*, *block size*, and *inode size*. All possible configurations form a *parameter space*.

When configuring storage systems, users often stick with the default configurations provided by vendors because 1) it is nearly impossible to know the impact of every parameter across multiple layers; and 2) vendors' default configurations are trusted to be "good enough". However, previous studies [66] showed that tuning even a tiny subset of parameters could improve the performance and energy efficiency for storage systems by as much as 9×. As technological progress slows down, it becomes even more important to squeeze every bit of performance out of deployed storage systems.

In the rest of this section we first discuss the challenges of system tuning (§2.1). Then, §2.2 briefly introduces several promising techniques that we explore in this paper. §2.3 explains certain methods that we deem less promising. §2.4 provides a unified view of these optimization methods.

2.1 Challenges

The tuning task for storage systems is difficult, due to the following four challenges.

(1) Large parameter space. Modern storage systems are fairly complex and easily come with hundreds or even thousands of tunable parameters. One evaluation for storage systems can take multiple minutes or even hours, which makes exhaustive search impractical. Even human experts cannot know the exact impact of every parameter and thus have little insight into how to optimize. For example, Ext4+NFS would result in a parameter space consisting of more than 10^{22} unique configurations. IBM's General Parallel File System (GPFS) [64] contains more than 100 tunable parameters, and hence 10^{40} configurations. From the hardware perspective, SSDs [30, 53, 57, 65], shingled drives [1, 2, 32, 45], and non-volatile memory [40, 83] are gaining popularity, plus more layers (LVM, RAID) are added.

(2) Non-linearity. A system is *non-linear* when the output is not directly proportional to the input. Many computer systems are non-linear [16], including storage systems [74]. For example, Figure 1 shows the average operation latency of GPFS under a typical database server workload while changing only the value of the parameter *pagepool* from 32MB to 128MB, and setting all the others to their default. Clearly the average latency is not directly proportional to the *pagepool* size. In fact, through our experiments, we have seen many more parameters with similar behavior. Worse, the parameter space for storage systems is often sparse, irregular, and contains multiple peaks. This makes automatic optimization even more challenging, as it has to avoid

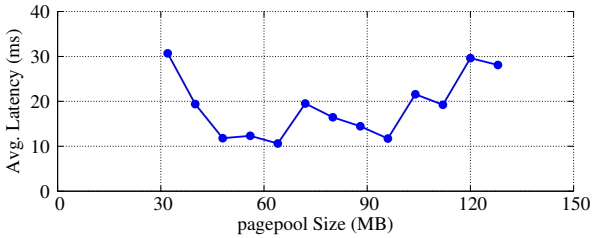

Figure 1: Storage systems are non-linear.

getting stuck in a local optima [36].

(3) Non-reusable results. Previous studies have shown that evaluation results of storage systems [12, 66] and databases [78] are dependent on the specific hardware and workloads. One good configuration might perform poorly when the environment changes. Our evaluation results in Section 4 show similar observations.

(4) Discrete and non-numeric parameters. Some storage system parameters can take continuous real values, while many others are discrete and take only a limited set of values. Some parameters are not numeric (e.g., I/O scheduler name or file system type). This adds difficulty in applying gradient-based approaches.

Given these challenges, manual tuning of storage systems becomes nearly impossible while automatic tuning merely difficult. In this paper we focus on automatic tuning and treat it as an optimization problem.

2.2 Applied Methods

Several classes of algorithms have been proposed for similar optimization tasks, including automated tuning for hyper-parameters of machine learning systems [7, 8, 59] and optimization of physical systems [3, 78]. Examples include Genetic Algorithms (GA) [18, 34], Simulated Annealing (SA) [15, 41], Bayesian Optimization (BO) [11,68], and Deep Q-Networks (DQN) [46,54,55]. Although these methods were proposed originally in different scholarly fields, they can all be characterized as black-box optimizations. In this section we introduce several of these techniques that we successfully applied in auto-tuning storage systems.

Simulated Annealing (SA) is inspired by the annealing process in metallurgy, which involves the heating and controlled cooling of a material to get to a state with minimum thermodynamic free energy. When applied to storage systems, a *state* corresponds to one *configuration*. *Neighbors* of a state refer to new configurations achieved by altering only one parameter value of the current state. The thermodynamic free energy is analogous to optimization objectives. SA works by maintaining the *temperature* of the system, which determines the probability of accepting a certain move. Instead of always moving towards better states as hill-climbing methods do, SA defines an *acceptance probability distribution*, which allows it to accept some bad moves in the short

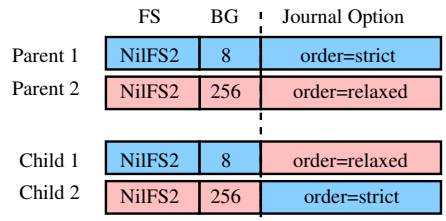

Figure 2: Crossover in Genetic Algorithm (GA).

run, that can lead to even-better moves later on. The system is initialized with a high temperature, and thus has high probability of accepting worse states in the beginning. The temperature is gradually reduced based on a pre-defined *cooling schedule*, thus reducing the probability of accepting bad states over time.

Genetic Algorithms (GA) were inspired by the process of natural selection [34]. It maintains a population of *chromosomes* (configurations) and applies several genetic operators to them. *Crossover* takes two parent chromosomes and generates new ones. As Figure 2 illustrates, two parent Nilfs2 configurations are cut at the same *crossover* point, and then the subparts after the crossover point are exchanged between them to generate two new child configurations. Better chromosomes will have a higher probability to "survive" in future *selection* phases. *Mutation* randomly picks a chromosome and mutates one or more parameter values, which produces a completely different chromosome.

Reinforcement Learning (RL) [72] is an area of machine learning inspired by behaviorist psychology. RL explores how software agents take actions in an environment to maximize the defined cumulative rewards. Most RL algorithms can be formulated as a model consisting of: (1) A set of environment states; (2) A set of agent actions; and (3) A set of scalar rewards. In case of storage systems, *states* correspond to *configurations*, *actions* mean changing to a different configuration, and *rewards* are differences in evaluation results. The agent records its previous experience (history), and makes it available through a *value function*, which can be used to predict the expected reward of state-action pairs. The *policy* determines how the agent takes action, which maintains the *exploration-exploitation* trade-off. The value function can take a tabular form, but this does not scale well to many dimensions. Function approximation is proposed to deal with high dimensionality, which is still known to be unstable or even divergent. With recent advances in Deep Learning [28], deep convolutional neural networks, termed Deep Q-Networks (DQN), were proposed to parameterize the value function, and have been successfully applied in solving various problems [54, 55]. Many variants of DQN have been proposed [46].

Bayesian Optimization (BO) [11, 68] is a popular framework to solve optimization problems. It models

Algorithm	Origin	Exploration	Exploitation	History
Simulated Annealing (SA)	Annealing technology in metallurgy	Allowing moving to worse neighbor states	Neighbor function	N/A
Genetic Algorithms (GA)	Natural evolution	Mutation	Crossover and selection	Current population
Deep Q-Networks (DQN)	Behaviorist psychology and neuroscience	Taking random actions	Taking actions based on action-reward function	Deep convolutional neural network
Bayesian Optimization (BO)	Statistics and experimental design	Selecting samples with high variances	Selecting samples with high mean values	Acquisition function & probabilistic model

Table 1: Comparison and summaries of optimization techniques.

the objective function as a stochastic process, with the argument corresponding to one storage configuration. In the beginning, a set of prior points (configurations) are given to get a fair estimate of the entire parameter space. BO works by computing the *confidence interval* of the objective function according to previous evaluation results, which is defined as the range of values that the evaluation result is most likely to fall into (e.g., with 95% probability). The next configuration is selected based on a pre-defined *acquisition function*. Both confidence intervals and the acquisition function are updated with each new evaluation. BO has been successfully applied in various areas, including hyper-parameter optimization [17] and system configuration optimization [3]. BO and its variants differ mainly in their form of probabilistic models and acquisition functions. In this paper we focus mainly on Gaussian priors and an Expected Improvement acquisition function [68].

Other promising techniques include Tabu Search [27], Particle Swarm Optimization [39], Ant Colony Optimization [20], Memetic Algorithms [52], etc. Due to space limits, we omit comparing all of them in this paper (part of our future work). In fact, as detailed in §2.4, most of these techniques actually share similar traits.

2.3 Other Methods

Although many optimization techniques have been proposed, we feel that not all of them make good choices for auto-tuning storage systems. For example, since many parameters of storage systems are non-numeric, most gradient-based methods (i.e., based on linear-regression) are less suitable to this task [29].

Control Theory (CT). CT was historically used to manage linear system parameters [19, 37, 44]. CT builds a controller for a system so its output follows a desired reference signal [33, 43]. However, CT has been shown to have the following three problems: 1) CT tends to be unstable in controlling non-linear systems [48, 49]. Although some variants were proposed, they do not scale well. 2) CT cannot handle non-numeric parameters; and 3) CT requires a lot of data during the learning phase, called *identification* to build a good controller.

Supervised Machine Learning (ML). Supervised ML has been successfully applied in various domains [9, 10, 56, 81]. However, the accuracy of ML models depends heavily on the quality and amount of training data [81], which is not available or impossible to collect for large parameter spaces such as ours.

Therefore, we feel that neither CT nor supervised ML, in their current state, are the first choice to *directly and efficiently* apply for auto-tuning storage systems. That said, they constantly evolve and new promising results appear in research literature [4, 67, 69, 86]; we plan to investigate them in the future.

2.4 Unified Framework

Most optimization techniques are known to follow the *exploration-exploitation* dilemma [23, 46, 68, 79]. Here we summarize the aforementioned methods by extending the unified framework with a third factor, the *history*. Our unified view thus defines three factors or dimensions: ■ **(1) Exploration** defines how the technique searches unvisited areas. This often includes a combination of pure random and also guided search based on *history*. ■ **(2) Exploitation** defines how the technique leverages *history* to find next sample. ■ **(3) History** defines how much data from previous evaluations is kept. History information can be used to help guide both future exploration and exploitation (e.g., avoiding less promising regions, or selecting regions that have never been explored before). Table 1 summarizes how the aforementioned techniques work by maintaining the balance among these three key factors. For example, GA keeps the evaluation results from the last generation, which corresponds to the concept of *history*. GA then *exploits* the stored information, applying selection and crossover to search nearby areas and pick the next generation. Occasionally, it also randomly mutates some chosen parameters, which is the idea of *exploration*. As shown in §4, the trade-off among exploration, exploitation, and history determines the effectiveness and efficiency of these optimization techniques.

3 Experimental Settings

We now describe details of the experimental environments, parameter spaces, and our implementations of optimization algorithms.

Param.	Abbr.	Values
File System	FS	Ext2, Ext3, Ext4, XFS, Btrfs, Nilfs2, Reiserfs
Block (Leaf) Size	BS	1K, 2K, 4K
Inode Size, Sector Size	IS	n/a, 128, 256, 512, 1024, 2048, 4096, 8192
Block Group	BG	n/a, 2, 4, 8, 16, 32, 64, 128, 256
Journal Option	JO	n/a, order=strict, order=relaxed, data=journal, data=ordered, data=writeback
Atime Option	AO	relatime, noatime
Special Option	SO	n/a, compress, nodatacow, nodatasum, notail
I/O Scheduler	I/O	noop, cfq, deadline

Table 2: Details of parameter spaces.

Hardware. We performed experiments on two sets of machines with different hardware categorized as low-end (M1) and mid-range (M2). We list the hardware details in Table 3. We also use Watts Up Pro ES power meters to measure the energy consumption [82].

Workload. We benchmarked storage configuration with four typical macro-workloads generated by Filebench [25, 75]. ■ **(1) Mailserver** emulates the I/O workload of a multi-threaded email server. ■ **(2) Fileserver** emulates the I/O workload of a server that hosts users' home directories. ■ **(3) Webserver** emulates the I/O workload of a typical static Web server with a high percentage of reads. ■ **(4) Dbserver** mimics the behaviors of Online Transaction Processing (OLTP) databases. Before each experiment run, we formatted and mounted the storage devices with the targeted file system.

The working set size affects the duration of an experiment [74]. Our goal in this study was to explore a large set of parameters and values quickly (although it still took us over two years). We therefore decided to trade the working set size in favor of increasing the number of configurations we could explore in a practical time period. We used the default working set sizes in Filebench, and ran each workload for 100 seconds; this is long enough to get stable evaluation results under this setting. The experiments demonstrate a wide range of performance numbers and are suitable for evaluating different optimization methods.

Parameter Space. Since the main goal of our paper is to compare multiple optimization techniques, we want our storage parameter spaces to be large and complex enough. Alas, evaluations for storage systems take a long time. Considering experimentation on multiple hardware settings and workloads, we decided to experiment with a reasonable subset of the most relevant storage system parameters. We selected parameters in close collaboration with several storage experts that have either contributed to storage stack designs or have spent years tuning storage systems in the field. We experimented with 7 Linux file systems that span a wide range of designs and features: *Ext2* [13], *Ext3* [77], *Ext4* [24], *XFS* [73], *Btrfs* [61], *Nilfs2* [42], and *Reiserfs* [60].

Hardware	M1	M2
Model	Dell PE SC1425	Dell PE R710
CPU	Intel Xeon single-core 2.8GHz CPU × 2	Intel Xeon quad-core 2.4GHz CPU × 2
Memory	2GB	24GB
Storage	HDD1 (73GB Seagate ST373207LW SCSI drive)	HDD2 (147GB SAS), HDD3 (500GB SAS), HDD4 (250GB SATA), SSD (200GB)

Table 3: Details of experiment machines.

Our experiments were mainly conducted on two sets of parameters, termed as *Storage V1* and *Storage V2*. We started with seven common file system parameters (shown in the first 7 rows of Table 2), and refer it as *Storage V1*. *Storage V1* was tested on M1 machines. We then extended our search space with one more parameter, the *I/O Scheduler*, and refer to it as *Storage V2*. *Storage V2* was evaluated on M2 servers. Note that certain combinations of parameter values could produce invalid configurations. For example, for Ext2, the journaling option makes no sense because Ext2 does not have a journal. To handle this, we added a value *n/a* to the existing range of parameters. Any parameter with *n/a* value is considered invalid. Invalid configurations will always come with evaluation results of zero (i.e., no throughput); this ensures they are purged in an upcoming optimization process. There are 2,074 valid configurations in *Storage V1* and 6,222 in *Storage V2* for each workload and storage device. We believe our search spaces are large and complex enough to demonstrate the difference in efficiency of various optimization algorithms. Furthermore, many of the chosen parameters are commonly tuned and studied by storage experts; having a basic understanding of such parameters helped us understand auto-tuning results.

Experiments and implementations. Our experiments and implementation consist of two parts. First, we exhaustively ran all configurations for each workload and device on M1 and M2 machines, and stored the results in a relational database. We collected the throughput in terms of I/O operations per second, as reported by Filebench, the running time (including setup time), as well as power and energy consumption. To acquire more accurate and stable results, we evaluated each configuration under the same environment for at least 3 runs, resulting in more than 450,000 total experimental runs. This data collection benefited our evaluation on auto-tuning as we can simply simulate a variety of algorithms by just querying the database for the evaluation results for different configurations, without having to rerun slow I/O experiments. The exhaustive search

Hardware-Workload-Device	File System	Block Size	Inode Size	BG Count	Journal Options	Atime Options	Special Options	I/O Scheduler	Throughput (IOPS)
M1-Mail-HDD1	Nilfs2	2K	n/a	256	order=relaxed	relatime	n/a	-	3,677
M2-Mail-HDD3	Ext2	4K	256	32	n/a	relatime	n/a	noop	18,744
M2-File-HDD3	Btrfs	4K	4,096	n/a	n/a	relatime	compress	deadline	16,549
M2-Mail-SSD	Ext2	4K	256	8	n/a	relatime	n/a	noop	18,845
M2-DB-SSD	Ext4	1K	128	2	data=ordered	noatime	n/a	noop	41,948
M2-Web-SSD	Ext4	4K	128	4	data=ordered	noatime	n/a	noop	16,185

Table 4: Global optimal configurations with different settings and workloads.

also lets us know exactly what the global optimal configurations are, so that we can calculate how close each optimization method gets to the global optimum.

Second, we simulated the process of auto-tuning storage systems by running the desired optimization method and querying the database for the average evaluation results from multiple (3+) repeated runs. We focused on optimizing for throughput in this paper. The computation cost of optimization algorithms are ignored in our experiments. We believe our observations are applicable to other optimization objectives as well. Our implementations of optimization methods are mostly based on open-source libraries. We use Pyevolve [58] for Genetic Algorithms, Scikit-Optimize [70] for Bayesian Optimization, and TensorFlow [76] for the DQN implementation. We implemented a simple version of Simulated Annealing, with both linear and geometric cooling schedules. (We also fixed bugs in Pyevolve and plan to release our patches.) Most of our implementation was done by converting storage-related concepts into algorithm-specific ones. For example, for GA, we defined each storage parameter as a *gene*, and each configuration as a *chromosome*. For DQN we provided storage-specific definitions for states, actions, and rewards. The complete implementation uses around 10,000 lines of Python code.

4 Evaluations

Our evaluation mainly focuses on comparing the effectiveness and speed of applying multiple optimization techniques on auto-tuning storage systems, and providing insights into our observations. §4.1 overviews the data sets that we collected for over two years. §4.2 compares five popular optimization techniques from several aspects. §4.3 uses GA as a case study to show that hyper-parameters of these methods can also impact the auto-tuning results. §4.4 takes the first step towards explaining these black-box optimization methods, based on our evaluation results and our storage expertise.

4.1 Overview of Data Sets

As per §3, our experimental methodology is to first exhaustively run all configurations under different workloads and test machines. We stored the results in a database for future use. This data collection benefits future experiments as we can simulate a variety of algorithms by querying the database for the evaluation results of different configurations. Due to space limits, in this section we show only 6 representative data sets out of 18: 2 workloads on M1 and 4 devices $\times$ 4 workloads on M2. They were picked to (1) show a wide range of hardware and workloads' impact on optimization results and (2) to present more SSD results, given SSDs' increasing popularity.

Figure 3 shows the throughput CDF among all configurations for each hardware setting and workload. The Y-axis is normalized by the maximum throughput under each experiment setting. The symbols on each line mark the default Ext4 configurations. As seen, for most settings, throughput values vary across a wide range. The ratios of the worst throughput to the best one are mostly between 0.2–0.4. In one extreme case, for *File-server* on *M2* machines and with the HDD3 device (abbreviated as *M2-Fileserver-HDD3*), the worst configuration only produces 1% I/O operations per unit time, compared with the global optimal one. This underlines the importance of tuning storage systems: an improperly configured system could be remarkably under-utilized, and thus wasting a lot of resources. However, *M2-Webserver-SSD* shows a much narrower range of throughput, with the worst-to-best ratio close to 0.9. This is attributed mainly to the fact that *Webserver* consists of mostly sequential read operations that are processed similarly by different I/O stack configurations. Figure 3 also shows that default Ext4 configurations are always sub-optimal and, under most settings, ranked lower than the top 40% configurations. For *M1-Mailserver-HDD1*, the default Ext4 configuration shows a normalized throughput of 0.39, which means that the optimal configuration performs 2.5 times better.

Table 4 lists optimal configurations for the same six hardware and workload settings. As we can see, optimal configurations depend on the specific hardware as well as the running workload. For *M1-Mailserver-HDD1*, the global best is a *Nilfs2* configuration. However, if we fix the workload, change the hardware, and get *M2-Mailserver-HDD3*, the optimum becomes an *Ext4* configuration. Similarly, fixing the hardware to *M2-*-SSD* and experimenting under different workloads leads to different optimal configurations. This proves our early

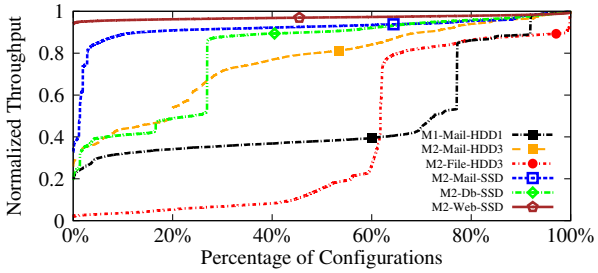

Figure 3: Throughput CDF with different hardware and workloads, with symbols marking the default Ext4 configurations.

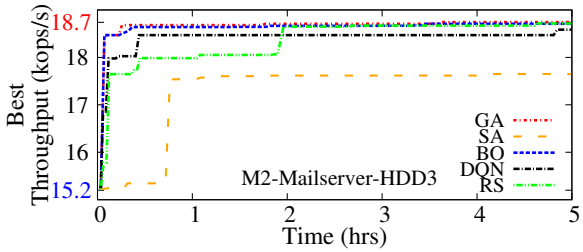

Figure 4: Highest throughput found over time, zooming in the $Y \in [15:19]$ range. The blue number (15.2) on the Y-axis shows the default, and the red one (18.7) shows the optimal.

claim that performance is sensitive to the environment (e.g., hardware, configuration, and workloads); this actually complicates the problem as results from one environment cannot be directly applied in another.

4.2 Comparative Analysis

Many optimization techniques have been applied to various auto-tuning tasks [71, 78]. However, previous efforts picked algorithms somewhat arbitrarily and evaluated only one algorithm at a time. Here we provide the first comparative study of multiple black-box optimization techniques on auto-tuning storage systems. As discussed in §2.2, we focus our evaluations on a representative set of optimization methods, and their common hyper-parameter settings, including 1) Simulated Annealing (SA), with a linear cooling schedule; 2) Genetic Algorithms (GAs) with population size of 8, mutation rate of 2%; 3) Deep Q-Networks (DQN) with experience replay [55] and $\epsilon = 0.2$, where ϵ; represents the probability of an agent taking random actions. 4) Bayesian Optimization (BO) with Expected Improvement (EI) and Gaussian prior; and 5) Random Search (RS), which merely performs random selection without replacement. We provide more discussion on the impact of hyper-parameters in §4.3. Note that SA, DQN, and RS experiments start with the default Ext4 configuration. GA and BO require several initial configurations (*prior points*), which we set to default configurations of all seven file systems. This allows us to simulate real-world use cases, where users often deploy their system with the default settings (and may manually optimize starting from the defaults). In the current experiments we assume that changing parameter values comes at no cost. In reality, parameters like *Block Size* may require re-formatting file systems.

Figure 4 presents one simulated run of each optimization method on *M2-Mailserver-HDD3*; the Y-axis shows the throughput value of the best configuration found so far, and the X-axis is the running time. All time-related metrics in this paper are based on the actual running time of evaluating each storage configuration, which is stored in our database. This includes both setup time and benchmarking time. We are not comparing the running costs (including any necessary training phases) for optimization methods here, which is our future work. Figure 4 is plotted by zooming in the range of $Y \in [15:19]$, with the blue number (15.2) on Y-axis represents the default, while the red one (18.7) shows the global optimal. Here we define a near-optimal configuration as one with throughput higher than 99% of the global optimal value. As shown in Figure 4, all five methods were able to gradually find higher performing configurations, but their effectiveness and speed differed a lot. SA performed the worst, and got stuck in a configuration with throughput value of less than 18K IOps. DQN was able to converge to a good configuration, but spent more time to achieve that than RS. GA and BO performed best out of these five tested optimization methods. They both successfully identified a near-optimal configuration within one hour. Interestingly, we observed that pure Random Search (RS) produced better results than some other optimization methods; the reason is that within the search space *M2-Mailserver-HDD3*, 4.5% of total configurations are near-optimal. RS needs only to hit one of them to reach good auto-tuning results.

Since exploration is one critical component of optimizations (see §2.4), their evaluation results could also exhibit some degree of randomness. To compare them more thoroughly, we ran each optimization technique on the same environment for 1,000 runs. Figure 5 shows the results, which evaluate the techniques' probability to find near-optimal configurations, defined the same as in Figure 4. The Y-axis shows the percentage of total runs that found a near-optimal configuration within a certain time (X-axis). Under *M2-Mailserver-HDD3*, seen in the upper part of Figure 5, SA had the lowest probability among 5 algorithms. Even after 5 hours, only around 80% of its runs found one near-optimal configuration, which shows that SA sometimes gets stuck in a local optimum. For other optimization methods, given enough time, over 90% of their runs converged to a near-optimal configuration, with BO outperforming GA, and GA outperforming DQN. RS shows the highest probability of finding near-optimal configurations when approaching 5 hours. This is reasonable because given enough time, a random selection will eventually hit near-optimal points.

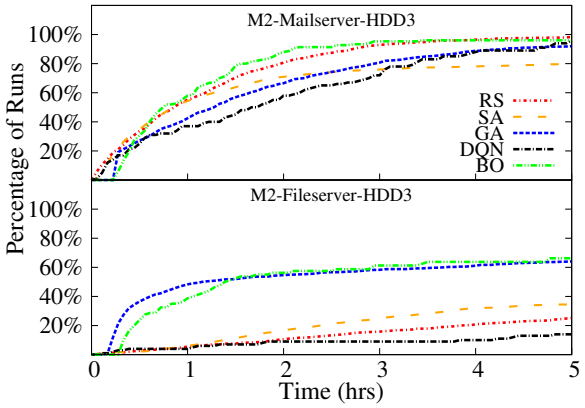

Figure 5: Comparing optimization methods' efficacy in finding near-optimal configurations. The Y-axis shows the percentage of total runs (1,000) that found near-optimal configurations within certain time (X-axis).

However, when conducting the same experiments under *M3-Fileserver-HDD3*, it becomes more difficult to find near-optimal configurations. GA and BO are still the best, though only 65% of their runs were able to find near-optimal configurations within 5 hours. SA, RS, and DQN have a probability of lower than 40% to do so, with DQN perform the worst. This is because the global optimum under *M2-Fileserver-HDD3* is a Btrfs configuration (see Table 4). It is more difficult for optimization algorithms to pick such configurations for the following reasons: 1) Few Btrfs configurations reside in the neighborhood of the default Ext4 configurations; 2) Fewer than 2% of all valid configurations are Btrfs ones, which make them less likely to be selected through mutation; 3) Only 0.2% of all configurations are considered near-optimal, compared with 4.5% in **Mailserver-HDD3**.

The above results focused on finding near-optimal configurations. However, another important aspect to compare is the system's performance *during* the auto-tuning process. This is especially important if the targeted system is deployed and online. Some randomness (exploration) is necessary when searching a complex parameter space, but ideally optimization algorithms should spend less time on bad configurations. To compare this, in Figure 6 we plotted the instantaneous throughput (Y-axis) over time (X-axis) for one run with each method under *M2-Mailserver-HDD3*.

BO and GA are still the best two methods in terms of instantaneous throughput. During the tuning process, occasionally they pick a worse configuration than the current one. However, they both possess the ability to quickly discard these unpromising configurations. GA achieves this by assigning the probability of surviving to next generation based on the fitness values (i.e., throughput). Configurations with low throughput values have a lower chance to be picked as parents, and thus their

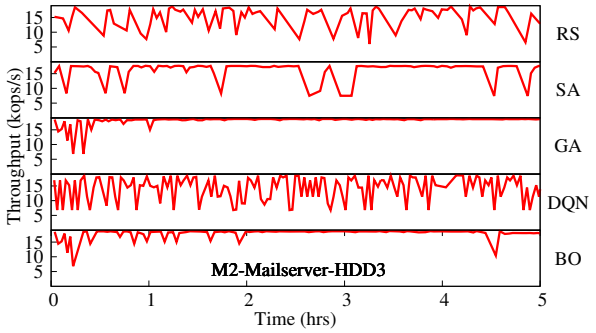

Figure 6: Comparing optimization methods' instantaneous performance (Y-axis) over time (X-axis).

genes (parameter values) have a lower chance of appearing in configurations of the next generation (i.e., "survival of the fittest"). The reason for stable instantaneous throughputs with BO is that it uses an intelligent acquisition function to guide the selection of the next generation, with the goal of maximizing the potential gain; this makes BO less likely to choose a bad configuration. In contrast, SA performs poorly possibly because it lacks a history to guide the exploitation and exploration phases, and only uses its neighborhood information (and current temperature) to pick the next configuration. DQN shows similar results with RS, which is likely caused by the fact that DQN was originally designed as an agent interacting with an unknown environment, and thus a lot of exploration (randomness) occurs in the training phase [55,85].

In conclusion, our results demonstrated that the efficacy of different optimization algorithms vary a lot while applied in auto-tuning storage systems. The trade-off among *exploitation*, *exploration*, and *history* plays an important role in find near-optimal configurations efficiently. However, a well-known problem for many optimization techniques is that their performance depend heavily on hyper-parameter settings. Some of our observations may only apply to our specific settings and search spaces. This paper's main goal is not to provide guidelines on which methods are more suitable for auto-tuning storage systems; rather, we focused on comparing multiple methods and understanding their efficacy under different conditions.

4.3 Impact of Hyper-Parameters

Many optimization methods' efficacy depend on the specific hyper-parameter settings, and choosing the right hyper-parameters has caused headache to researchers for a long time [7, 8]. In this section we use GA as a case study, and show the impact of one hyper-parameter, the *mutation rate*, on auto-tuning results. The mutation rate controls the probability of randomly mutating one parameter to a different value, and aligns with the idea of *exploration*, as per §2.4.

Figure 7 shows the results from 7 sets of GA exper-

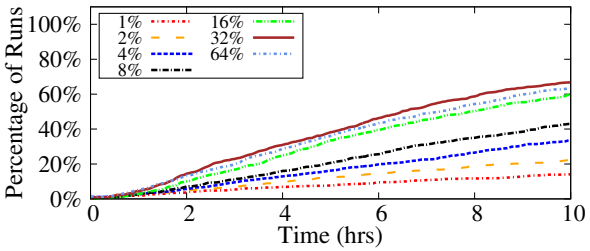

Figure 7: Impact of mutation rates on GA.

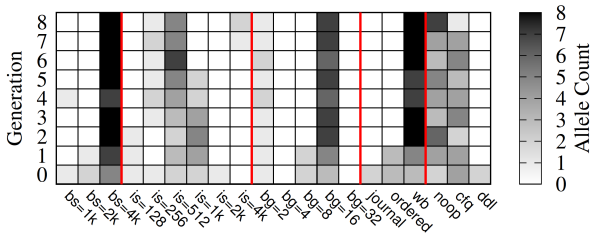

Figure 8: Number of alleles in the first 8 GA generations, with more frequent ones colored with darker colors.

iments with different mutation rates (from 1% to 64%) under *M2-Mailserver-HDD3*. Each experiment was repeated for 1,000 runs. It is similar to Figure 5, but with the goal of finding near-optimal configurations whose throughput values are higher than **99.5%** of the global optimal. This makes the optimization more challenging, as GA already performs quite well on easier tasks (§4.2). As shown in the figure, when increasing the mutation rate, GA has a higher probability to converge to near-optimal configurations within a shorter time period. This is because GA works by identifying promising combination of alleles (parameter values) for the subset of dominant genes (parameters). We define dominant parameters as those having a higher impact on performance than all others. A higher mutation rate means a higher chance of exploration, and thus finding combinations of well-performing alleles for the dominant genes within a shorter time. We explain this effect more in §4.4. However, a mutation rate of 64% actually performs worse than 32%. This is because in order to reach near-optimal configurations, GA needs both exploration and exploitation. Exploration lets GA identify processing subspaces (i.e., combinations of certain parameter values) while exploitation helps GA search within promising subspaces. In this case, with a mutation rate of 64%, GA spends too much time on exploration (too much randomness), resulting in fewer chances for exploitation.

4.4 Peering into the Black Box

Despite some successful applications of black-box optimization on auto-tuning system parameters, few have explained how and why some techniques work better than others for certain problems. Here we take the first step towards unpacking the "black box" and provide some insights into their internals based on our evaluation results and storage domain knowledge.

Our attempts for explanations stem from a somewhat unexpected but beneficial behavior of GA in the experiments. We found that as GA runs, there is often a small set of alleles (parameter values) that dominate the current population and are unlikely to change. We present and explain this observation in Figure 8. The experiment was conducted on a parameter space consisting of 2,208 Ext3 configurations under *M2-Fileserver-SSD*. The X-axis shows 5 genes (parameters) separated by red gridlines, while one column represents one allele (parameter value). The parameters are denoted with their abbreviations from Table 2. The Y-axis shows the generation number, and we plotted only the first 8 generations. Cells were colored based on the number of alleles in each generation. More frequent alleles are colored with darker colors. In the first generation, the gene's alleles (parameter values) were quite diverse. For example, there were 3 alleles (1K, 2K, 4K) for the *Block Size* gene, and 3 alleles (journal, ordered, writeback) for the *Journal Option* gene. However, the diversity of alleles decreased in later generations, and several genes began to dominate and even converged to a single allele. For the *Block Size* gene, only the 4K allele survived and other two became extinct. Since GA was proposed by simulating the process of natural selection, where alleles with better fitness are more likely to survive, this suggests that GA works by identifying the combination of good alleles (storage parameter values), and producing offspring with these alleles. As shown in Figure 8, in the 8^{th} generation, all configurations have a *Block Size* of 4K and *Journal Option* of writeback.

To confirm the above observations, in Figure 9 we plotted all Ext3-SSD configurations, with one dot corresponding to one configuration. Configurations are separated based on the *Journal Option*, shown as the X-axis, and colored based on their *Block Size*. To clearly see all points within each X-axis section, we ordered configurations by their unique identification number in our database. The Y-axis represents throughput values. This resulted in the formation of nine "clusters" on the graph, each corresponding to a fixed ⟨*Journal Option*, *Block Size*⟩ pair. We can see that configurations with *data=ordered* tend to produce higher throughput than those with *data=journal*, and *data=writeback* produces the best throughput. This is somewhat expected from a storage point of view, as Ext3's more fault tolerant journal option (*data=journal*) may hurt throughput by writing data as well as meta-data to the journal first. Moreover, among journal configurations with *data=writeback*, those with a 4K *Block Size* turn out to produce the highest throughput. This aligns with our observation from Figure 8 that GA works by identifying a subset of genes that have a greater impact

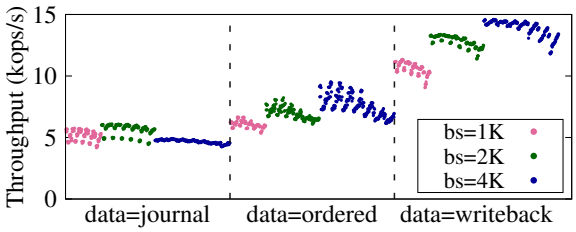

Figure 9: Scatter plot for all Ext3-SSD configurations, with one dot corresponding to one configuration.

on performance—*Block Size* and *Journal Option*—and finding the best alleles for them ([*4K, data=writeback*]).

Based on these observations, one interesting question to ask is whether the conclusion that a subset of parameters have greater impact on performance than other parameters, also holds for other file systems and workloads. To answer this question, we quantified the correlation between each parameter and throughput values. As most of our parameters are categorical or discrete numeric, whereas the throughput is continuous, we took a common approach to quantify the correlation between categorical and continuous variables [14]. We illustrate with the *Block Size* parameter as an example. Since it can take 3 values, we convert this parameter to three binary variables x_1, x_2, and x_3. If the *Block Size* is 1K, we assign $x_1 = 1$ and x_2 and x_3 are set to 0. Let Y represent the throughput values. We then do a linear regression with ordinary least squares (OLS) on Y and x_1, x_2, x_3. R^2 is a common metric in statistics to measure how the data fits a regression line. In our approach, R^2 actually quantifies the correlation between the selected parameter and throughput. We consider $R^2 > 0.6$ as an indication that the parameter has significant impact on performance, as is common in statistics [14]. The same calculation is applied to all parameters for each file system under *M2-Fileserver-SSD* and *M2-Dbserver-SSD*. Parameters with the highest R^2 values are colored in yellow background in Table 5. If all R^2 values are below 0.6, we simply leave the entries blank, meaning no highly correlated parameters were found. To find the second important parameter, the same process is applied to the remaining parameters, but with the value of the most important one fixed (to isolate its effect on the remaining parameters' importance). Taking Ext4 under *M2-Fileserver-SSD* as an example, we calculate R^2 values for all other parameters among configurations with the same *Journal Option*. For one parameter, 3 *Journal Options* lead to three R^2 values; we then take the maximum one as the R^2 value for this parameter. We color the parameter with the highest R^2 in Table 5 with a green background.

We can see that the correlated parameters are quite diverse, and depend a lot on file systems. For example, under *M2-Fileserver-SSD*, the two most important

WL-Dev	FS	BS	IS	BG	JO	AO	SO	I/O
File-SSD	Ext2	-	-	-	-	-	-	0.68
	Ext3	0.84	-	-	0.90	-	-	-
	Ext4	0.92	-	-	0.99	-	-	-
	XFS	0.94	-	0.82	-	-	-	-
	Btrfs	-	-	-	-	-	-	-
	Nilfs2	0.99	-	-	-	-	-	0.94
	Reiserfs	-	-	-	0.74	-	-	0.99
Db-SSD	Ext2	-	-	-	-	-	-	-
	Ext3	0.72	-	-	0.96	-	-	-
	Ext4	-	-	-	0.96	0.68	-	-
	XFS	-	-	-	-	-	-	-
	Btrfs	-	-	-	-	-	-	-
	Nilfs2	0.62	-	-	-	-	-	0.80
	Reiserfs	-	-	-	0.99	-	-	-

Table 5: Importance of parameters (measured by R^2), with the most important one colored in yellow and second in green.

parameters for Ext3 (in descending order) are *Journal Option* and *Block Size*; this aligns with our observation in Figures 8 and 9. However, for Reiserfs, the top 2 changes to *I/O Scheduler* and *Journal Option*. Interestingly, all parameters for Btrfs come with low R^2 values, which indicates that no parameter has significant impact on system performance under *M2-Fileserver-SSD* with Btrfs. Correlation of parameters can also depend on the workloads. For instance, the two dominant parameters for XFS under *M2-Fileserver-SSD* are *Block Size* and *Allocation Group*. When the workload changes to *M2-Dbserver-SSD*, all parameters for XFS seem to have minor impact on performance. In this paper we are isolating the impact of each parameter, thus assuming that their effect on throughput is independent. Note that the above observations are made based on our collected data sets, and might change on different workloads and hardware. However, our methodology is generally applicable. Moreover, this paper's main goal is *not* to suggest guidelines on what specific storage configurations to deploy under certain workloads; rather, we focus on comparing multiple optimization methods and providing insights into their operation.

The fact that parameters have varied impact on performance can also help explain the auto-tuning results in §4.2. Although our parameter space comes with 8 parameters, only a subset of them are highly correlated with performance. As long as the optimization algorithm identifies the "correct" combination of values for these dominant parameters, it will be able to find a near-optimal configuration. Similar behavior has been reported in hyper-parameter optimization problems [7]. For the experiments shown in Figure 4, near-optimal configurations take up 4.5% of the whole search space. Random Search (RS) needs to hit only one of them to achieve good auto-tuning results. GA's efficacy comes from assigning a higher chance of survival to configura-

tions with a certain combination of values for the dominant parameters. BO stores its previous search experience (history) in a probabilistic surrogate model that it is building, which eventually encodes the combination of dominant parameter values that can result in good throughput values. SA does not work as well because it lacks history information to identify the dominant parameters: it wastes time changing less useful parameters and converges slowly. Similarly, DQN also spends lots of its effort on exploring unpromising spaces, which slows its ability to find near-optimal configurations.

5 Limitations and Future Work

In this paper we provided the first comparative analysis of applying multiple optimization methods on auto-tuning storage systems. However, auto-tuning is a complex topic and more effort is required. We list some limitations of this work and our future research directions below. ■ **(1)** We plan to extend the scope of evaluation with more complex workloads and search spaces. We will investigate more techniques, such as experiment design [80], as will as the impact of algorithm hyper-parameter settings [8]. ■ **(2)** We plan to improve traditional optimization techniques with new features, such as penalty functions to cope with costly parameter changes, stopping/restarting criteria, workload identification, handling noisy and unstable results [12], etc., which makes auto-tuning algorithms more robust to environment changes and more generally applicable in production systems.

6 Related Work

Auto-tuning computer systems. In recent years, several attempts were made to automate the tuning of storage systems. Strunk et al. [71] proposed to use utility functions combining different system metrics and applied GA to automate storage system provisioning. Babak et al. [5] utilized GA to optimize I/O performance of HDF5 applications. GA has also been applied for storage recovery problems [38]. More recently, Deep Q-Networks has been successfully applied in optimizing performance for Lustre [85]. Auto-tuning is also a hot topic in other computer systems: Bayesian Optimization was applied to find near-optimal configurations for databases [78] and Cloud VMs [3]. Other applied techniques include Evolutionary Strategies [62], Simulated Annealing [26, 35], Tabu Search [63], and more. However, previous work all focused on one or a few techniques. One contribution of our work is to provide the first comparative study of multiple, applicable optimization methods on their efficacy in auto-tuning storage systems from various aspects. We also provide some insights into the working mechanism of auto-tuning.

Hyper-parameter tuning. Evolutionary Algorithms [59], Reinforcement Learning [6], and Bayesian Optimization [22] have been applied to hyper-parameter optimization for ML algorithms. Bergstra and Bengio [8] found that randomly chosen trials are more efficient for hyper-parameter optimization than trials on a grid, and explained the cause as the objective function having a low effective dimensionality. Another direction of research focuses on eliminating all hyper-parameters and tries to propose non-parametric versions of optimization methods. Examples of this include GA [31,51] and BO [68].

7 Conclusions

Optimizing storage systems can provide significant benefits especially in improving I/O performance. Alas, storage systems are getting more complex, contain many parameters and an immense number of possible configurations; manual tuning is therefore impractical. Worse, many of those parameters are non-linear or non-numeric; traditional linear-regression-based optimization techniques do not work well for such problems. Several efforts were made to apply black-box optimization techniques to auto-tune storage systems, but they all used only one or few techniques. In this work, we performed the first comparative study, and offered the following four contributions. **(1)** We evaluated *five* popular but different auto-tuning techniques, varied some of their hyper-parameters, and applied them to storage and file systems. **(2)** We show that the speed at which the techniques can find optimal or near-optimal configurations (in terms of throughput) depends on the hardware, software, and workload; this means that no single technique can "rule them all." **(3)** We explain why some techniques appear to work better than others. **(4)** For more than two years, we have collected a large data set of over 450,000 data points; this data set was used in this study and we plan to release it.

Acknowledgments

We thank the anonymous ATC reviewers and our shepherds Keith Smith and Siddhartha Sen for their valuable comments. This work was made possible in part thanks to Dell-EMC, NetApp, and IBM support; NSF awards CNS-1251137, CNS-1302246, CNS-1305360, CNS-1622832, CNS-1650499, and CNS-1730726; and ONR award N00014-16-1-2264.

References

[1] Abutalib Aghayev, Mansour Shafaei, and Peter Desnoyers. Skylight—a window on shingled disk operation. *Trans. Storage*, 11(4):16:1–16:28, October 2015.

[2] Abutalib Aghayev, Theodore Ts'o, Garth Gibson, and Peter Desnoyers. Evolving ext4 for shingled

disks. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST)*, pages 105–120, Santa Clara, CA, February/March 2017. USENIX Association.

[3] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 469–482. USENIX Association, 2017.

[4] Terry Anderson. *The theory and practice of online learning*. Athabasca University Press, 2008.

[5] Babak Behzad, Huong Vu Thanh Luu, Joseph Huchette, Surendra Byna, Prabhat, Ruth Aydt, Quincey Koziol, and Marc Snir. Taming parallel i/o complexity with auto-tuning. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 68:1–68:12, New York, NY, USA, 2013. ACM.

[6] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[7] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[8] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24*, pages 2546–2554, 2011.

[9] Christopher M Bishop. *Pattern Recognition and Machine Learning*, volume 1. Springer New York, 2006.

[10] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[11] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[12] Zhen Cao, Vasily Tarasov, Hari Raman, Dean Hildebrand, and Erez Zadok. On the performance variation in modern storage stacks. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST)*, pages 329–343, Santa Clara, CA, February/March 2017. USENIX Association.

[13] R. Card, T. Ts'o, and S. Tweedie. Design and implementation of the second extended filesystem. In *Proceedings to the First Dutch International Symposium on Linux*, Amsterdam, Netherlands, December 1994.

[14] George Casella and Roger L Berger. *Statistical Inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[15] Vladimír Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.

[16] Yvonne Coady, Russ Cox, John DeTreville, Peter Druschel, Joseph Hellerstein, Andrew Hume, Kimberly Keeton, Thu Nguyen, Christopher Small, Lex Stein, and Andrew Warfield. Falling off the cliff: When systems go nonlinear. In *Proceedings of the 10th Conference on Hot Topics in Operating Systems - Volume 10*, HOTOS'05, 2005.

[17] Valentin Dalibard, Michael Schaarschmidt, and Eiko Yoneki. Boat: Building auto-tuners with structured bayesian optimization. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 479–488. International World Wide Web Conferences Steering Committee, 2017.

[18] Kenneth Alan De Jong. *Analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, Ann Arbor, MI, USA, 1975.

[19] Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano. Using MIMO linear control for load balancing in computing systems. In *2004 American Control Conferences*, 2004.

[20] Marco Dorigo, Mauro Birattari, and Thomas Stützle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, 2006.

[21] Fred Douglis, Deepti Bhardwaj, Hangwei Qian, and Philip Shilane. Content-aware load balancing for distributed backup. In *Large Installation System Administration Conference (LISA)*, 2011.

[22] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, 2013.

[23] A.E. Eiben and C.A. Schippers. On evolutionary exploration and exploitation. *Fundam. Inf.*, 35(1-4):35–50, January 1998.

[24] Ext4. *http://ext4.wiki.kernel.org/*.

[25] Filebench, 2016. *https://github.com/filebench/filebench/wiki*.

[26] Terry L Friesz, Hsun-Jung Cho, Nihal J Mehta, Roger L Tobin, and G Anandalingam. A simulated annealing approach to the network design problem with variational inequality constraints. *Transportation Science*, 26(1):18–26, 1992.

[27] Fred Glover and Manuel Laguna. *Tabu Search*. Springer, 2013.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. *http://www.deeplearningbook.org*.

[29] Gradient descent. *https://en.wikipedia.org/wiki/Gradient_descent*.

[30] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A Chien, and Haryadi S Gunawi. The tail at store: a revelation from millions of hours of disk and ssd deployments. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 263–276, 2016.

[31] Georges R Harik and Fernando G Lobo. A parameter-less genetic algorithm. In *GECCO*, volume 99, pages 258–267, 1999.

[32] Weiping He and David H.C. Du. Smart: An approach to shingled magnetic recording translation. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST)*, pages 121–134, Santa Clara, CA, February/March 2017. USENIX Association.

[33] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tibury. *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.

[34] J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U. Michigan Press, 1975.

[35] Young-Jae Jeon, Jae-Chul Kim, Jin-O Kim, Joong-Rin Shin, and Kwang Y Lee. An efficient simulated annealing algorithm for network reconfiguration in large-scale distribution systems. *Power Delivery, IEEE Transactions on*, 17(4):1070–1078, 2002.

[36] N. Joukov, A. Traeger, R. Iyer, C. P. Wright, and E. Zadok. Operating system profiling via latency analysis. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI 2006)*, pages 89–102, Seattle, WA, November 2006. ACM SIGOPS.

[37] M. Karlsson, C. Karamanolis, and X. Zhu. Triage: Performance differentiation for storage systems using adaptive control. *ACM Trans. Storage*, 1(4), 2005.

[38] Kimberly Keeton, Dirk Beyer, Ernesto Brau, Arif Merchant, Cipriano Santos, and Alex Zhang. On the road to recovery: Restoring data after disasters. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys'06, pages 235–248, New York, NY, USA, 2006. ACM.

[39] James Kennedy and Russell C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[40] H. Kim, S. Seshadri, C. L. Dickey, and L. Chiu. Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, pages 33–45, Berkeley, CA, 2014. USENIX.

[41] S. Kirkpatrick, C D. Gelatt, M. P Vecchi, et al. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[42] Ryusuke Konishi, Yoshiji Amagai, Koji Sato, Hisashi Hifumi, Seiji Kihara, and Satoshi Moriai. The linux implementation of a log-structured file system. *ACM SIGOPS Operating Systems Review*, 40(3):102–107, 2006.

[43] Ernest Bruce Lee and Lawrence Markus. Foundations of optimal control theory. Technical report, DTIC Document, 1967.

[44] H. D. Lee, Y. J. Nam, K. J. Jung, S. G. Jung, and C. Park. Regulating I/O performance of shared storage with a control theoretical approach. In *NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST)*. IEEE Society Press, 2004.

[45] Yin Li, Hao Wang, Xuebin Zhang, Ning Zheng, Shafa Dahandeh, and Tong Zhang. Facilitating magnetic recording technology scaling for data center hard disk drives through filesystem-level transparent local erasure coding. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST)*, pages 135–148, Santa Clara, CA, February/March 2017. USENIX Association.

[46] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[47] Z. Li, M. Chen, A. Mukker, and E. Zadok. On the trade-offs among performance, energy, and endurance in a versatile hybrid drive. *ACM Transactions on Storage (TOS)*, 11(3), July 2015.

[48] Z. Li, K. M. Greenan, A. W. Leung, and E. Zadok. Power consumption in enterprise-scale backup storage systems. In *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST '12)*, San Jose, CA, February 2012. USENIX Association.

[49] Z. Li, R. Grosu, K. Muppalla, S. A. Smolka, S. D. Stoller, and E. Zadok. Model discovery for energy-aware computing systems: An experimental evaluation. In *Proceedings of the 1st Workshop on Energy Consumption and Reliability of Storage Systems (ERSS'11)*, Orlando, FL, July 2011.

[50] Z. Li, A. Mukker, and E. Zadok. On the importance of evaluating storage systems' $costs. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'14, 2014.

[51] Fernando G Lobo and David E Goldberg. The parameter-less genetic algorithm in practice. *Information Sciences*, 167(1):217–232, 2004.

[52] Peter Merz. Memetic algorithms for combinatorial optimization problems: Fitness landscapes and effective search strategies, 2001.

[53] Justin Meza, Qiang Wu, Sanjev Kumar, and Onur Mutlu. A large-scale study of flash memory failures in the field. In *Proceedings of the 2015 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2015)*, pages 177–190, Portland, OR, June 2015. ACM.

[54] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[55] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[56] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[57] Iyswarya Narayanan, Di Wang, Myeongjae Jeon, Bikash Sharma, Laura Caulfield, Anand Sivasubramaniam, Ben Cutler, Jie Liu, Badriddine Khessib, and Kushagra Vaid. SSD failures in datacenters: What? when? and why? In *Proceedings of the Second ACM Israeli Experimental Systems Conference (SYSTOR '16)*, pages 7:1–7:11, Haifa, Israel, May 2016. ACM.

[58] Christian S. Perone. Pyevolve: A python open-source framework for genetic algorithms. *SIGEVOlution*, 4(1):12–20, November 2009.

[59] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

[60] H. Reiser. ReiserFS v.3 whitepaper. *http://web.archive.org/web/20031015041320/http://namesys.com/*.

[61] Ohad Rodeh, Josef Bacik, and Chris Mason. BTRFS: The Linux B-tree filesystem. *Trans. Storage*, 9(3):9:1–9:32, August 2013.

[62] Anooshiravan Saboori, Guofei Jiang, and Haifeng Chen. Autotuning configurations in distributed systems for performance improvements using evolutionary strategies. In *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, ICDCS '08, pages 769–776, Washington, DC, USA, 2008. IEEE Computer Society.

[63] Sadiq M Sait, Mahmood R Minhas, Junhaid Khan, et al. Performance and low power driven vlsi standard cell placement using tabu search. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, pages 372–377. IEEE, 2002.

[64] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST '02)*, pages 231–244, Monterey, CA, January 2002. USENIX Association.

[65] Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. Flash reliability in production: The expected and the unexpected. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST)*, pages 67–80, Santa Clara, CA, February 2016. USENIX Association.

[66] P. Sehgal, V. Tarasov, and E. Zadok. Evaluating performance and energy in file system server workloads. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pages 253–266, San Jose, CA, February 2010. USENIX Association.

[67] Burr Settles. *Active Learning*. Morgan & Claypool Publishers, 2012.

[68] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[69] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2011.

[70] Scikit-Optimize. *https://scikit-optimize.github.io/*.

[71] John D. Strunk, Eno Thereska, Christos Faloutsos, and Gregory R. Ganger. Using utility to provision storage systems. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 313–328, Berkeley, CA, USA, 2008. USENIX Association.

[72] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[73] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. In *Proceedings of the Annual USENIX Technical Conference*, pages 1–14, San Diego, CA, January 1996.

[74] V. Tarasov, S. Bhanage, E. Zadok, and M. Seltzer. Benchmarking file system benchmarking: It *is* rocket science. In *Proceedings of HotOS XIII:The 13th USENIX Workshop on Hot Topics in Operating Systems*, Napa, CA, May 2011.

[75] Vasily Tarasov, Erez Zadok, and Spencer Shepler. Filebench: A flexible framework for file system benchmarking. *;login: The USENIX Magazine*, 41(1):6–12, March 2016.

[76] TensorFlow. *https://www.tensorflow.org/*.

[77] Stephen Tweedie. Ext3, journaling filesystem. In *Ottawa Linux Symposium*, July 2000. *http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html*.

[78] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1009–1024, 2017.

[79] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv.*, 45(3):35:1–35:33, July 2013.

[80] Shivaram Venkataraman, Zongheng Yang, Michael J Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation*, pages 363–378, 2016.

[81] Mengzhi Wang, Kinman Au, Anastassia Ailamaki, Anthony Brockwell, Christos Faloutsos, and Gregory R. Ganger. Storage device performance prediction with cart models. In *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems. (MASCOTS)*, pages 588–595, 2004.

[82] Watts up? PRO ES power meter. *www.wattsupmeters.com/secure/products.php*.

[83] H-S Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E Goodson. Phase change memory. *Proceedings of the IEEE*, 98(12):2201–2227, Dec 2010.

[84] Bowei Xi, Zhen Liu, Mukund Raghavachari, Cathy H. Xia, and Li Zhang. A smart hill-climbing algorithm for application server configuration. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 287–296, New York, NY, USA, 2004. ACM.

[85] Oceane Bel Ethan L. Miller Darrell D. E. Long Yan Li, Kenneth Chang. Capes: Unsupervised system performance tuning using neural network-based deep reinforcement learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, 2017.

[86] Yang Yu, Hong Qian, and Yi-Qi Hu. Derivative-free optimization via classification. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2286–2292. AAAI Press, 2016.

[87] Erez Zadok, Aashray Arora, Zhen Cao, Akhilesh Chaganti, Arvind Chaudhary, and Sonam Mandal. Parametric optimization of storage systems. In *HotStorage '15: Proceedings of the 7th USENIX Workshop on Hot Topics in Storage*, Santa Clara, CA, July 2015. USENIX, USENIX.