EXHIBIT B

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*

Appears in the Proceedings of the 2018 Annual USENIX Technical Conference (ATC'18)

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, nonlinear 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 $9\times$ [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

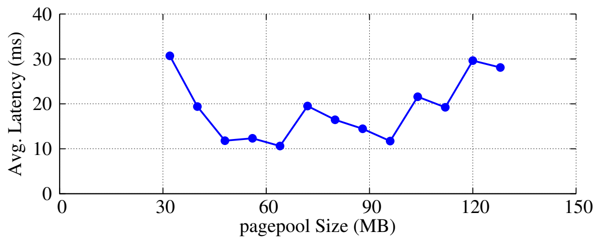

Figure 1: Storage systems are non-linear.

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 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 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.

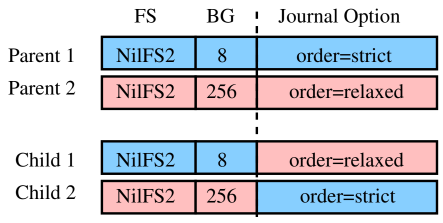

Figure 2: Crossover in Genetic Algorithm (GA).

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 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.

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.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase®
Smarter legal research.