
# Carver: Finding Important Parameters for Storage System Tuning

Zhen Cao, *Stony Brook University;* Geoff Kuenning, *Harvey Mudd College;*
Erez Zadok, *Stony Brook University*

https://www.usenix.org/conference/fast20/presentation/cao-zhen

This paper is included in the Proceedings of the
18th USENIX Conference on File and
Storage Technologies (FAST '20)
February 25–27, 2020 • Santa Clara, CA, USA
978-1-939133-12-0

Open access to the Proceedings of the
18th USENIX Conference on File and
Storage Technologies (FAST '20)
is sponsored by

# Carver: Finding Important Parameters for Storage System Tuning

Zhen Cao,[1] Geoff Kuenning,[2] and Erez Zadok[1]
[1]*Stony Brook University* *and* [2]*Harvey Mudd College*

## Abstract

Storage systems usually have many parameters that affect their behavior. Tuning those parameters can provide significant gains in performance. Alas, both manual and automatic tuning methods struggle due to the large number of parameters and exponential number of possible configurations. Since previous research has shown that some parameters have greater performance impact than others, focusing on a smaller number of more important parameters can speed up auto-tuning systems because they would have a smaller state space to explore. In this paper, we propose Carver, which uses (1) a variance-based metric to quantify storage parameters' importance, (2) Latin Hypercube Sampling to sample huge parameter spaces; and (3) a greedy but efficient parameter-selection algorithm that can identify important parameters. We evaluated Carver on datasets consisting of more than 500,000 experiments on 7 file systems, under 4 representative workloads. Carver successfully identified important parameters for all file systems and showed that importance varies with different workloads. We demonstrated that Carver was able to identify a near-optimal set of important parameters in our datasets. We showed Carver's efficiency by testing it with a small fraction of our dataset; it was able to identify the same set of important parameters with as little as 0.4% of the whole dataset.

## 1 Introduction

Storage systems are critical components of modern computer systems that have significant impact on application performance and efficiency. Most storage systems have many configurable parameters that control and affect their overall behavior. For example, Linux's Ext4 [22] offers about 60 parameters, representing over $10^{37}$ potential configuration states. The default settings are often sub-optimal; previous research has shown that tuning storage parameters can improve system performance by a factor of as much as 9× [59].

To cope with the vast number of possible configurations, system administrators usually focus on using their domain expertise to tune a few frequently used and well-studied parameters that are *believed* to significantly impact system performance. However, this manual-tuning approach does not scale well in the face of increasing complexity. Modern storage systems use different file system types [21, 37, 56, 65], new hardware (SSDs [26, 46], SMR [1, 2], NVM [33, 73]), multi-tier and hybrid storage, and multiple virtualization layers (e.g., LVM, RAID). Storage systems range from one or a few identical nodes to hundreds of highly heterogeneous configurations [23, 57]. Worse, tuning results depend heavily on hardware and the running workloads [10, 11, 70].

Recently, several optimization methods have been used to auto-tune storage systems, achieving good performance improvements within reasonable time frames [11, 40]. These auto-tuning techniques model the storage system as a black box, iteratively trying different configurations, measuring an objective function's value, and—based on previously learned information—selecting new configurations to try. However, many black-box auto-tuning techniques have difficulty scaling to high dimensions and can take a long time to converge on good solutions [61]. Therefore, the problem of dealing with the vast number of storage-parameter configurations remains largely unsolved.

In machine learning and information theory, dimensionality reduction is often applied to explosively sized datasets [5, 48]. We believe it can also be applied to storage-parameter selection. Previous research has reported that certain storage parameters have greater impact on performance than others [11]. By eliminating the less important parameters, and ordering parameters by importance, the parameter search space—and thus the number of configurations that need to be considered by either humans or algorithms—can be reduced significantly [28].

Evaluating a single storage configuration is time consuming, and a thorough analysis requires many configurations to be explored; these evaluations can span days or even months. One purpose of a storage parameter-selection algorithm is to be able to pick important parameters by evaluating only a small number of configurations, yet still select the important parameters with high accuracy.

In this paper, we propose Carver, which efficiently selects a subset of important storage parameters. Carver consists of three components: 1) a variance-based metric to quantify the importance of a storage parameter; 2) a sampling method to intelligently pick a small number of configurations representing the whole parameter space; and 3) a greedy algorithm to select important parameters. Carver outputs a set of selected important parameters; these can be used as pre-selected parameters for auto-tuning algorithms, as well as helping human experts better understand the behaviors of targeted storage systems. As shown in Section 5, the aforementioned three components give Carver the ability to select a near-optimal subset of important parameters by exploring relatively few configurations. With this efficiency, Carver could complete its parameter selection in a relatively short period of time in a real deployment.

Carver was thoroughly evaluated on (publicly available)

experimental data collected from our previous work [11], in which we conducted benchmarks on 7 file systems under 4 workloads over a time span of around four years. In that work, for each file system we picked 8–10 frequently tuned parameters and evaluated *all* possible storage configurations resulting from changing the values of these selected parameters. We collected I/O throughput and latency data throughout the evaluation. The data set consists of more than 500,000 benchmark runs (data points) in total. One advantage of having collected the datasets from the whole configuration space is that they can be used as the ground truth when testing Carver with only a small subset of configurations.

With the collected datasets, we first confirmed that certain parameters have more impact on system throughput or latency than other parameters, using Carver's proposed importance metric. We found that in all datasets there is always a small set of parameters that have significantly more impact on throughput than all the others. For example, under a *File-server* workload, the two most important parameters for Ext4 were *Journal Option* and *I/O Scheduler*. We also observed that the set of important parameters varies with different workloads. In the same Ext4 example, the two most important parameters became *Block Size* and *Inode Size* when the workload changed to *Dbserver*. We also demonstrated that our variance-based metric can always find a near-optimal set of important parameters in these datasets.

We then demonstrated Carver's efficiency in identifying important parameters by applying it to different measurements, such as I/O throughput and latency. Carver can easily be extended and applied equally well to other quantifiable objectives such as energy consumption, and even composite cost functions [41]. In our evaluation, Carver uses Latin Hypercube Sampling (LHS) as the sampling method. LHS allows Carver to identify the set of important parameters using a small number of experimental runs that explore only a fraction of all configurations. For instance, among all 1,000 repeated runs, Carver was able to find the two most important parameters for Ext4 using only 0.4% of the evaluation results. We believe Carver's efficiency in finding the most important parameters quickly and accurately is critical and promising, since (1) it can be applied to new storage systems or environments, and (2) the parameters it identifies can then be used by storage administrators or auto-tuning algorithms to further optimize the system.

The three key contributions of this paper are:

1. We provide a thorough quantitative analysis of the effects of storage parameters on system performance, for 7 different file systems across 4 representative workloads.

2. We propose Carver, which uses a variance-based metric of storage-parameter importance and Latin Hypercube Sampling to drive a greedy algorithm that can identify the most important parameters using only a small number of experimental runs.

3. We thoroughly evaluated Carver's ability to identify important parameters in terms of I/O throughput and latency. We demonstrated that Carver successfully chose a near-optimal set of important parameters for all datasets used.

## 2 Motivation

In this paper, we define a **storage system** as the entire storage stack from file systems to physical devices, including all intermediate layers. Storage systems have many configurable options that affect their performance [10, 66], energy consumption [59], reliability [63], etc. We define a *parameter* as one configurable option, and a *configuration* as a combination of parameter values. For example, Ext4's *Journal Option* parameter can take three values: *data=writeback*, *data=ordered*, and *data=journal*. Based on this, *[journal="data=writeback", block_size=4K, inode_size=4K]* is one *configuration* with three specific parameter values (*Journal Option*, *Block Size*, and *Inode Size*). The list of all possible (legal) configurations forms a *parameter space*.

Storage systems usually come with many configurable parameters that control and affect their overall behavior. An earlier study [59] showed that tuning even a tiny set of parameters could improve performance and energy efficiency by as much as 9×. However, tuning storage systems is not an easy task; we believe its challenges arise from at least the following four aspects:

1. **Large parameter spaces**. Storage systems are complex, incorporating numerous file system types [21, 37, 56, 65], devices [1, 2, 26, 33, 46, 73], and intermediate layers [52, 54]. They often span large networks and distributed environments [6, 23, 30, 57]. Modern storage systems have hundreds or even thousands of tunable parameters—and networks are also parameterized. Worse, evaluating a single configuration can take many minutes or even hours, making experimental tuning unusually time-consuming.

2. **Nontransferable tuning results**. Evaluation results depend on the specific environment, including the hardware, software, and workload [10, 11, 59]. A good configuration for one setup might perform poorly when the environment changes even slightly [60].

3. **Nonlinear parameters**. A system is *nonlinear* when the output is not directly proportional to the input. Many computer systems are nonlinear [16], including storage systems [66]. This makes traditional regression-based analysis more challenging [50, 58].

4. **Discrete and non-numeric (categorical) parameters**. Some storage parameters are continuous, but many are

discrete and take only a limited set of values. Worse, some are *categorical* (e.g., the I/O scheduler name or file system type). Many optimization techniques perform poorly on discrete values, and often cannot address categorical values efficiently or at all [24, 49].

Given these challenges, manually tuning storage systems becomes nearly impossible, and automatic tuning can be computationally infeasible. Recent efforts have used black-box optimization techniques to auto-tune storage configurations [11, 40], addressing several of the above challenges and achieving useful performance improvements. However, we believe that the challenge of tuning storage systems is far from being solved. It has been shown that several of these black-box optimization techniques have scalability problems in high-dimensional spaces [61]. Therefore, directly applying them to tuning systems with hundreds or thousands of parameters would be difficult.

In machine learning and information theory, *dimensionality reduction* is a common technique for coping with large-sized datasets [5, 48]. If it can be applied in storage systems, it will significantly reduce the search space [28], making it easier for humans or algorithms to tune storage systems.

Previous work has reported that not all storage parameters have an equally important performance impact: a few have much greater effect than others [11]. We observed similar trends from our collected datasets. Figure 1 demonstrates the impact of the parameters *Block Size* and *I/O Scheduler* on the throughput of an Ext4 file systems under a typical file server workload. Each boxplot in the figure represents a median and range of throughput that any Ext4 configuration can produce after fixing the value of one parameter (shown on the X axis). We see that setting the *I/O Scheduler* to different values (blue bars) makes little difference, resulting in nearly equal medians and ranges of throughput. However, setting the value of *Block Size* has a greater impact on both the median and the throughput range; specifically, to reach the maximum throughput, *Block Size* must be set to 4K. Although choosing a large *Block Size* is a decision that may be obvious to an expert, we have made similar observations in other storage systems and with different workloads. This naturally led us to investigate how we can quantify the impact or importance of each storage parameter, and how we can select important parameters efficiently.

## 3 Dimensionality Reduction in a Nutshell

In this section we briefly discuss some commonly applied approaches to *dimensionality reduction*, and argue that some metrics are not suitable for quantifying storage parameters' importance. Note that different disciplines might use somewhat different terminology than storage systems. For example, parameters are analogous to *features* in machine learning, *independent variables* in regression analysis, and *dimensions* in mathematics; optimization objectives can be called *dependent variables* or *target variables*. When discussing

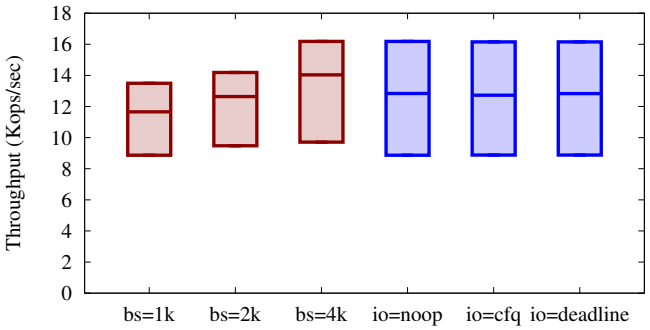



*Figure 1: Range of throughput after fixing the value of one parameter. Red bars represent setting the block size to 1K, 2K, or 4K, respectively, while blue bars represent setting the I/O scheduler to noop, cfq, or deadline.*

different techniques (Section 3), we use the field-appropriate terms.

Many approaches have been proposed to address the *curse of dimensionality*, which refers to the fact that data become sparse in high-dimensional spaces and thus make algorithms designed for low-dimensional spaces less effective. Dimensionality-reduction approaches can be generally summarized into two categories: *feature extraction* and *feature selection* [25, 39].

*Feature extraction* refers to projecting high-dimensional data into low-dimensional spaces; the newly constructed features are usually linear or nonlinear combinations of the originals. Common feature-extraction methods include Principal Component Analysis (PCA) [62], Independent Component Analysis [29], and Linear Discriminant Analysis [47]. One major drawback of feature extraction is that the physical meaning of each feature is lost by the projection and the nonlinear combination of many dimensions into fewer ones [39]. Common feature-extraction techniques thus conflict with our goal in this paper, which is to *select a few original storage parameters that can be understood and interpreted*.

Conversely, *feature selection* directly selects a subset of features from the original ones, with the intention of finding only those that are important. Feature-selection methods can be classified as *supervised* or *unsupervised* [39]. Unsupervised feature selection, such as Principle Feature Analysis [43], chooses a subset that contains most of the essential information based on relationships among features. It does not consider the impact of features on optimization objectives during the selection phase. In contrast, supervised feature selection chooses a subset that can discriminate between or approximate the target variables. Examples include Lasso [68] and decision-tree based algorithms [31]. Since we are interested in finding parameters that have significant impact on our optimization objectives, such as I/O throughput, supervised feature selection best fits our needs.

Several intrinsic properties of our project also limit our choice of feature-selection methods. Many storage parameters are discrete or categorical (see Sections 2 and 5.1). The

performance of storage systems is usually presented as I/O throughput or latency, which are continuous. Therefore, an ideal feature-selection method should work with categorical features and continuous targets. Although there are discretization techniques that can break continuous target variables into discrete sections, feature-selection results depend heavily on the quality of discretization [39]. One common approach for dealing with categorical features is to transform each of them into dummy binary parameters that take values of 0 or 1. For instance, *io_scheduler* with three possible values (*noop*, *deadline*, and *cfq*) can be converted into three binary features: *"io_scheduler = noop"*, *"io_scheduler = deadline"*, and *"io_scheduler = cfq"*. All the binary features can take on values 0 or 1. This approach is unsatisfactory because it selects the individual binary features instead of the original categorical ones. Moreover, converting a categorical parameter with $N$ values into $N$ separate binary parameters would *expand* the parameter space exponentially. For this reason, we feel that Lasso [68] is not suitable for our problem, even though it has been successfully applied to selecting important knobs in databases [70]. Although Group Lasso has been proposed to partially address this deficiency [14,34,74], the computational cost of the Lasso-based methods is still high [39].

Another popular category of feature-selection methods has been built upon information theory [8, 20, 31, 39]. These approaches usually define a metric for the *homogeneity* of the target variable within certain subsets. Commonly used metrics include Gini impurity [39] and Entropy [5] for discrete target variables, and Variance [7] for continuous variables. In this paper we propose Carver, which applies a *variance-based* metric for parameter importance, as described in Section 4.1.

## 4 Design of Carver

In this section we detail the design of Carver. Carver consists of three components: 1) a variance-based metric for measuring storage parameters' importance (Section 4.1), 2) a sampling method to select a small number of configurations from huge parameter spaces—in this paper using Latin Hypercube Sampling (Section 4.2), and 3) a greedy algorithm for finding important parameters (Section 4.3). A good sampling method allows Carver to select a near-optimal subset of important parameters while having to evaluate relatively few configurations. In this section we use throughput as an example of the target (objective) variable, but Carver is also applicable to many other metrics.

### 4.1 Measuring Parameter Importance

Carver uses a variance-based metric to quantify storage-parameter importance. The variance of a set $S$ of storage configurations is defined as usual:

$$\mathrm{Var}(S) = \frac{1}{|S|}\sum_{i=1}^{|S|}(y_i - \mu)^2, \tag{1}$$

where $y_i$ is the throughput of the i-$th$ configuration; $|S|$ is number of configurations in $S$; and $\mu$ is the average throughput within $S$. Inspired by CART (Classification and Regression Trees) [7], we use the *reduction in variance* to measure parameter importance. We extend CART's original definition to support categorical parameters taking an arbitrary but finite number of values, as compared with only two in CART.

We define the parameter importance $PI$ of a parameter $P$ that can take a finite number of categorical values, $\{p_1, ..., p_n\}, n > 1$, as:

$$PI(P) = \mathrm{Var}(S) - \sum_{i=1}^{n}\frac{|S_{P=p_i}|}{|S|}\mathrm{Var}(S_{P=p_i}) \tag{2}$$

Here $S$ is the original set of configurations, and $S_{P=p_i}$ is the subset of configurations with the parameter $P$ taking the value $p_i$. Intuitively, an important parameter $P$ divides a set $S$ of configurations into multiple subsets, and the weighted sum of variances within each subset should be much smaller than the variance of $S$. Thus, a high $PI$ indicates a parameter that has a significant effect on performance.

The variance-based metric defined in Carver uses a greedy approach, where the next important parameter will be picked by calculating its importance when fixing the values of previously selected parameters. Therefore, for parameter $Q$ with a total of $m$ possible categorical values $\{q_1, ..., q_m\}, m > 1$, we define the *conditional parameter importance* for $Q$, given $P = p$ as:

$$CPI(Q|P=p) = \mathrm{Var}(S_{P=p}) - \sum_{j=1}^{m}\frac{|S_{Q=q_j,P=p}|}{|S_{P=p}|}\mathrm{Var}(S_{Q=q_j|P=p}) \tag{3}$$

where $S_{Q=q_j,P=p}$ denotes the set of configurations with parameters $P$ and $Q$ taking values $p$ and $q_j$, respectively. Similar to Equation 2, given $P = p$, the next most important parameter $Q$ divides $S_{P=p}$ into multiple subsets, and if $Q$ is important then the weighted sum of variances within each subset will be much smaller than variance of $S_{P=p}$. To remove the restriction to a given value $p$, we define $CPI(Q|P)$ as the maximum of $CPI(Q|P = p_i)$ over all possible values $p_i \in \{p_1, ..., p_n\}$ that parameter $P$ can take:

$$CPI(Q|P) = \max_{i=1}^{n} CPI(Q|p = p_i) \tag{4}$$

Note that in this paper we use only variance-based metrics to measure parameter importance and select the most critical subset. We leave storage-performance prediction, which requires a large amount of training data [71], for future work.

## 4.2 Sampling

Given the large parameter space and the time needed to evaluate a single storage configuration, we must limit the number of experimental runs required to select important parameters. Therefore, Carver needs an exploratory method that can cover the space uniformly and comprehensively, yet sparsely. In this work, we chose Latin Hypercube Sampling (LHS) [45].

LHS is a stratified sampling method [13]. In two dimensions, a square grid containing samples is a *Latin Square* iff there is only one sample in each row and each column. A *Latin Hypercube* is the generalization of a Latin Square to higher dimensions, where each sample is the only one in each axis-aligned hyperplane containing it [36]. LHS has been shown to be more effective in exploring parameter spaces than random sampling [45] and Monte Carlo sampling [15]. It has been successfully applied in sampling configurations of storage [27] and cloud systems [42].

Previous work has also applied Plackett-Burman (P&B) Design [53] to evaluate the impact of parameters in storage benchmarks [51] and databases [18]. However, P&B design requires each parameter to have only two possible values, and the target variable must be a monotonic function of the input parameters. Neither requirement holds in our problem.

We demonstrated that LHS enables Carver to pick important storage parameters with only a small number of evaluations; see Section 5.4.

## 4.3 Parameter-Selection Algorithm

Based on our proposed measurements of parameter importance and on Latin Hypercube Sampling (LHS), the pseudocode for Carver's parameter-selection algorithm is as follows:

**Algorithm 1** Parameter Selection

**Input:** $P$: set of parameters, $S$: initial set of configurations; stop(S, selected): user-defined stopping function.
  $selected \leftarrow \{\}$
  $S^* \leftarrow LHS(S)$
  **repeat**
    $p^* \leftarrow \text{argmax}\ CPI(p|selected), p \in P$
    $selected.insert(p^*)$
    $P.remove(p^*)$
  **until** $stop(S, selected)$ **is** true **or** $P$ **is** empty
**Output:** $selected$

In this algorithm, Carver takes a set of initial parameters and configurations. It first uses LHS to pick a small number of configurations and evaluates them. Carver then greedily selects the current most important parameters based on the evaluation results for the selected configurations. The most-important parameter is selected based on the highest *parameter importance* value. Carver fixes the value of the most important parameter and calculates the *conditional parameter importance* (CPI) values for the remaining parameters; the parameter with the highest CPI is selected as the second-most important. Carver continues evaluating important parameters by fixing the values of previously selected parameters, until the *stop function* returns true. A naïve stop function could be $sizeof(selected) \geq N$, which would select the $N$ most important parameters. An alternative variance-based stopping function might stop when the variances of subsets of configurations (given the current *selected* parameters) are below a certain threshold $\vartheta$. This stopping condition indicates that by setting the values of the *selected* parameters, the system throughput already falls into a small enough range that there is little potential gain from additional tuning. In our experiments, we applied this idea and used the Relative Standard Deviation (RSD) [13], or Coefficient of Variation, to define our stopping condition. The RSD of a set $S$ of configurations is defined as:

$$\text{RSD}(S) = \frac{1}{\mu}\sqrt{\frac{\text{Var}(S)}{N-1}} \tag{5}$$

where $N$ is the number of configurations and $\mu$ is the mean throughput of configurations within $S$. We chose RSD because it is normalized to the mean throughput and is represented as a percentage; that way the same threshold can be used across different datasets. We used a threshold of $2\%$ in our experiments; as seen in Section 5, parameters selected by this criterion gave us near-optimal and stable throughput.

# 5 Evaluation

In this section we detail our evaluation of Carver. We first cover the experimental settings we used for collecting datasets in Section 5.1. Section 5.2 provides an overview of storage-parameter importance using our variance-based metric. Section 5.3 demonstrates that the subset of important parameters selected by Carver's importance metric is near-optimal. We show the efficiency of Carver's parameter-selection algorithm in Section 5.4, from multiple perspectives.

## 5.1 Experiment Settings

To thoroughly study the problem of storage parameter selection and evaluate Carver, we used datasets originally collected for our previous work [11]. The whole dataset consists of more than half a million benchmark results on typical storage systems. We describe the experimental settings and collected datasets in this section.

**Hardware.** We performed experiments using several Dell PE R710 servers, each with two Intel Xeon quad-core 2.4GHz CPUs, 24GB RAM, and four storage devices: two SAS HDDs, one SATA HDD, and one SSD. Ubuntu 14.04 was installed on all machines with Linux kernel 3.13. We denote this configuration as *S1*. We also collected several datasets on a slightly different configuration, *S2*, where we used the GRUB boot loader to limit the available memory to

4GB. We explain the reasons for this change below. We also upgraded the system to Ubuntu 16.04 with kernel 4.15. Experiments on *S2* were only conducted on the SSD, given the increasing use of SSDs in production systems.

**Workload.** We benchmarked storage configurations with four common macro-workloads generated by Filebench [3, 67]:

1. **Mailserver** mimics the I/O workload of a multi-threaded email server;
2. **Fileserver** emulates a server hosting users' home directories;
3. **Webserver** emulates a typical static Web server with a high percentage of reads; and
4. **Dbserver** mimics the behavior of an Online Transaction Processing (OLTP) database.

Before each experimental run, we formatted and mounted the storage devices with the selected configuration. In setting *S1* we chose Filebench's default workload profiles, limiting the working-set size so we could evaluate more configurations within a practical time period. We call those profiles *Mailserver-default*, *Fileserver-default*, etc. Our previous study's goal, which applies to this work as well, was to allow us to explore a large set of parameters and values quickly. By evaluating each configuration once, saving the results, and later looking them up in our database, we could test Carver in seconds instead of waiting for several hours to run the benchmarks selected by Algorithm 1. Clearly, a real-world deployment would not have such a database available and a search for the most important parameters would require running actual benchmark tests, each of which would take significant time. However, as shown in Section 5.4, Carver tests few enough configurations that even these experiments can be completed in a short time, ranging from a few hours to a few days. An additional benefit of the full database is that we were able to compare configurations found by Carver with the true best configuration found by our complete datasets.

Because we wanted our database to record results of as many experiments as possible, we decided to trade off a smaller working set size in favor of increasing the number of configurations we could explore in a practical time period. Our experiments demonstrated a wide range of performance numbers and are suitable for the purpose of studying storage-parameter importance. As shown in Table 2, storage parameters do have a wide range of importance under these workloads. We first ran each workload for up to 2 hours to observe its behavior, and then chose a running time long enough for the cumulative throughput to stabilize; we found 100 seconds sufficient for this purpose. In setting *S2*, we increased the working-set size to 10GB and the running time to 300 seconds, but used relatively fewer total configurations, which we denote *Mailserver-10GB*, *Fileserver-10GB*, etc. The RAM size was set to 4GB in *S2* so that the benchmark working set could *not* fit into memory completely, thus forcing more I/Os.

**Parameter space.** To evaluate our parameter-selection algorithm, we ideally want our parameter spaces to be large and complex. Considering that evaluating storage systems takes a long time, we decided to experiment with a reasonably sized set of frequently studied and tuned storage parameters. We selected them in close collaboration with several storage experts who have either contributed to storage-stack designs or have spent years tuning storage systems in the field. We chose seven Linux file systems that span a wide range of designs and features: *Ext2* [12], *Ext3* [69], *Ext4* [21], *XFS* [65], *Btrfs* [56], *Nilfs2* [35], and *Reiserfs* [55]. We experimented with various types of parameters, including file-system formatting and mounting options and some Linux kernel parameters. Table 1 lists all our file systems, their (abbreviated) parameters, and the number of possible values that each parameter can take. Note that under *S1* we conducted benchmarks on four storage devices, and we treat the device as one of the parameters. Under *S2* we focused on Ext4 and XFS experiments with an SSD, but evaluated a wider variety of parameters. Cells with "–" mean that the parameters are inapplicable for the given file system. Cells with "dflt" mean we used the default value for that parameter, and so that parameter was not considered during the parameter-selection phase. Note that the total number of configurations for each file system does not necessarily equal the product of the number of parameter values, because some parameter combinations are invalid (e.g., in Ext4 the inode size cannot exceed the block size). The total number of configurations across all datasets is 29,544. We ran all configurations in each parameter space under four workloads. We repeated each experiment at least three times to get a stable and representative measurement of performance. Over a time span of more than two years, we collected data from more than 500,000 experimental runs.

Although we have been collecting benchmarking data over a time span of 4 years, we focused on one dataset at a time, where we benchmarked one file system on the same hardware under the same workload. Each dataset's collection took 1–2 months. Therefore, there may be minor hardware wear-out effects. We repeated each experiment for at least 3 runs, and made sure the variation among the results of these repeated runs were acceptable [10]. We used the average throughput and latency numbers among repeated runs when evaluating Carver.

## 5.2 Parameter Importance: an Overview

We have collected experimental data from 9 different parameter spaces (Table 1) under 4 representative workload types. Having the complete datasets allowed us to accurately cal-

| Setting | File System | Blk Size | Inode Size | Block Grp | Journal | Flex Grp | Read-ahead | XFS Sctr Size | Allc Grp Cnt | Log Buf Cnt | Log Buf Size | Allc Size | Node Size | Spec Opt | Atime Opt | I/O Schd | Drty Bg Ratio | Drty Ratio | Dev | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S1 | Ext2 | 3 | 7 | 6 | – | – | – | – | – | – | – | – | – | – | 2 | 3 | dflt | dflt | 4 | 2,208 |
| S1 | Ext3 | 3 | 7 | 6 | 3 | – | – | – | – | – | – | – | – | – | 2 | 3 | dflt | dflt | 4 | 6,624 |
| S1 | Ext4 | 3 | 7 | 6 | 3 | dflt | dflt | – | – | – | – | – | – | – | 2 | 3 | dflt | dflt | 4 | 6,624 |
| S1 | XFS | 3 | 5 | – | – | – | – | dflt | 9 | dflt | dflt | dflt | – | – | 2 | 3 | dflt | dflt | 4 | 2,592 |
| S1 | Btrfs | – | 5 | – | – | – | – | – | – | – | – | – | 3 | 4 | 2 | 3 | dflt | dflt | 4 | 288 |
| S1 | Nilfs2 | 3 | – | 9 | 2 | – | – | – | – | – | – | – | – | – | 2 | 3 | dflt | dflt | 4 | 1,944 |
| S1 | Reiserfs | dflt | – | – | 3 | – | – | – | – | – | – | – | – | 2 | 2 | 3 | dflt | dflt | 4 | 192 |
| S2 | Ext4 | 3 | 3 | dflt | 3 | 3 | 3 | – | – | – | – | – | – | – | dflt | 3 | 2 | 3 | SSD | 3,888 |
| S2 | XFS | 3 | 2 | – | – | – | – | 3 | 4 | 2 | 2 | 2 | – | – | dflt | 3 | 2 | 3 | SSD | 5,184 |

*Table 1: Details of parameter spaces. Each cell gives the number of settings we tested for the given parameter and file system; empty cells represent parameters that are inapplicable to the given file system and "dflt" represents those that were left at their default setting. We evaluated 29,544 configurations in total under four workloads, and each experiment was repeated 3+ times.*

culate and evaluate the importance of different storage parameters, which serves as the ground truth when evaluating Carver's parameter-selection algorithm, whose goal is to explore only a small fraction of the parameter space yet find the same subset of important parameters as if we had explored it all. In this section, we first provide an overview of the importance of storage parameters.

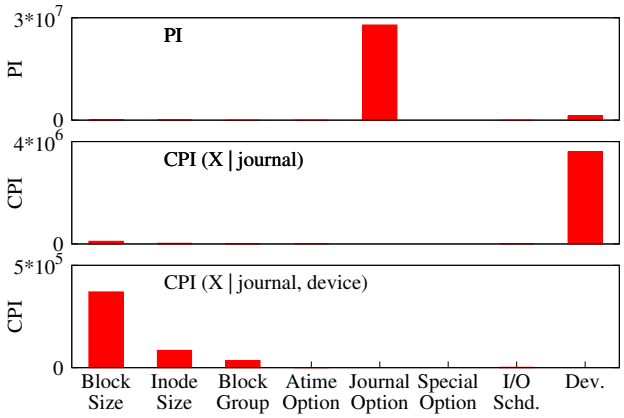



*Figure 2: Top 3 most important Ext4 parameters under S1, Fileserver-default. The most important parameter is measured by its PI; the second and third parameters are evaluated by their CPI given higher-ranked parameters. The Y-axis scales in the three sub-figures are different.*

Figure 2 shows the three most important parameters for Ext4 under *S1, Fileserver-default*. The parameter with the highest importance was evaluated and selected by its Parameter Importance (PI), as defined in Section 4.1. The second most important parameter was chosen by its Conditional Parameter Importance (CPI) given the most important one, in this case $CPI(X|journal)$. Similarly, the 3rd most important parameter was evaluated by comparing its $CPI(X|journal, device)$. Note that the Y-axis scales in the three sub-figures are different (but higher is always better). The X axis shows the Ext4 parameters that we experimented with. As shown in the top subfigure in Figure 2, *Journal Option* turns out to be the most important parameter for Ext4 under *S1, Fileserver-default*. It has the highest variance reduction, $2.7\times10^7$. In comparison, the $PI$ of *Device* is around $10^6$, while all other parameters are under $5 \times 10^4$. Similarly, the second and third most important parameters are *Device* and *Block Size*, respectively, both with a much higher $CPI$ value than other parameters.

We discovered that parameter importance depends heavily on file system types and on the running workload. Table 2 lists the top 4 important parameters for Ext4, XFS, and Btrfs under various workload types; the column header *#N* identifies the N$^{th}$ most important parameter. We also applied the stopping criterion described in Section 4.3. Cells marked as "–" here indicate that no parameter gave a large reduction in variance, and thus no parameter was considered important. To avoid cluttering the paper, we only list 3 file systems under 4 workloads here, and we show only the top 4 ranked parameters under each case.

As we can see in Table 2, the important parameters are quite diverse and depend significantly on the file system types and workloads. For Ext4 under *S2* and *Dbserver-10GB*, the top 4 ranked parameters are *Block Size*, *Inode Size*, *I/O Scheduler*, and *Journal Option*. When the workload changes to *Webserver-10GB*, the top 4 parameters become *Inode Size*, *Flex BG*, *Block Size*, and *Journal Option*. For *Fileserver-10GB* under Ext4, we found only three important parameters, indicating that fixing the values of these three parameters already resulted in quite stable throughputs; we discuss this observation in more detail in Section 5.3. We found similar results on XFS: the values and number of important parameters depended heavily on the workloads. Interestingly, for Btrfs under *S1, Webserver-default*, we did not find any important parameters. That is because the *Webserver-default* workload consists primarily of read operations, and the default working-set size used by Filebench is small. All Btrfs configurations actually produce quite similar throughput under *Webserver-default*. For this reason, we also collected datasets from workloads with a much larger working-set size (10GB), denoted as *S2*.

| Setting | Workload | File System | Parameter #1 | Parameter #2 | Parameter #3 | Parameter #4 |
|---|---|---|---|---|---|---|
| S2 | Fileserver-10GB | Ext4 | Journal Option | I/O Scheduler | Inode Size | – |
| S2 | Dbserver-10GB | Ext4 | Block Size | Inode Size | I/O Scheduler | Journal Option |
| S2 | Mailserver-10GB | Ext4 | I/O Scheduler | Inode Size | Journal Option | Block Size |
| S2 | Webserver-10GB | Ext4 | Inode Size | Flex Block Group | Block Size | Journal Option |
| S2 | Fileserver–10GB | XFS | I/O Scheduler | Inode Size | Allocation Group Count | – |
| S2 | Dbserver-10GB | XFS | Block Size | Log Buffer Size | Dirty Ratio | Alloc Group Count |
| S2 | Mailserver-10GB | XFS | Inode Size | I/O Scheduler | Log Buffer Size | Allocation Size |
| S1 | Fileserver-default | Btrfs | Special Option | Inode Size | Device | – |
| S1 | Mailserver-default | Btrfs | Inode Size | Device | – | – |
| S1 | Webserver-default | Btrfs | – | – | – | – |

*Table 2: Top-ranked important parameters for various file systems. The column header #N identifies the $N^{th}$ most important parameter.*

## 5.3 Evaluating The Greedy Algorithm

In Section 5.2 we used Carver's variance-based metric to pick a set of important parameters for our datasets. However, we must also establish that the selection results are good, i.e., whether there exists another set of parameters, with equal or smaller size, that can lead to an even narrower range of throughput. We demonstrate the effectiveness of Carver's variance-based metric in this section.

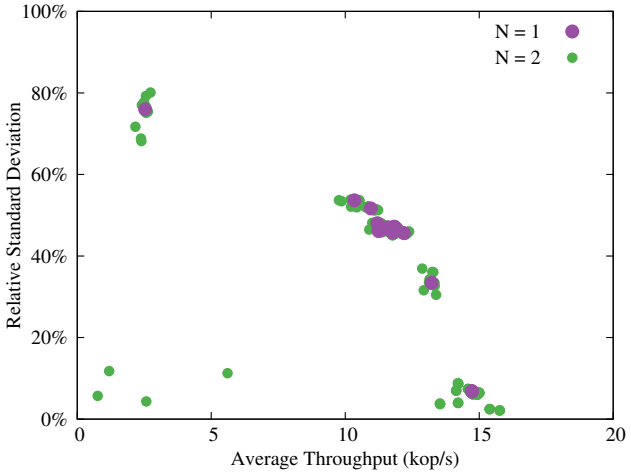



*Figure 3: Impact of parameters on performance and stability (Ext4, S1, Fileserver-default). Each dot represents a set of configurations created by fixing N parameters, where different dot sizes and colors are used for different values of N. Performance is measured by the average throughput (X axis) of all possible configurations within each set; stability is measured by the relative standard deviation (Y axis; lower is better) of the throughput within each set.*

Figure 3 shows the results for Ext4 under *S1, Fileserver-default*, where each point represents a set of configurations that fixes the values of $N$ parameters. For $N = 1$, we have 28 points, which equals the sum of possible value counts for each parameter, as shown in Table 1. There are 374 points for $N = 2$. We use different point colors and sizes for different numbers of parameters. We only plot up to $N = 2$ here; we extend to $N = 4$ in Figure 4. Larger points are used for smaller $N$ values, since fixing fewer parameter values would result in a larger number of usable configurations. For example, fixing *journal_option* = *ordered* in our datasets leads to a set of 2,208 configurations; fixing *journal_option* = *ordered, device=ssd* reduces that number to 552.

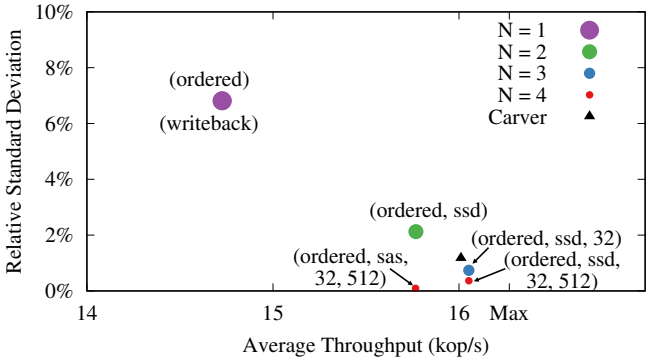
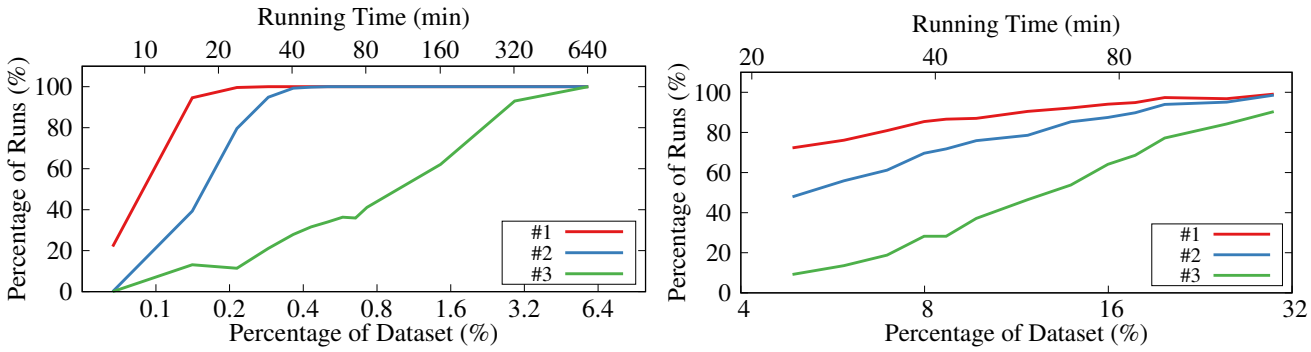



*Figure 4: A zoom into the bottom-right part of Figure 3 (the "best" quadrant), with points for $N = 3, 4$ added. Plotted points show either the highest average throughput or the lowest relative standard deviation among all configurations gotten by fixing the values of N parameters. The labels around the dots show the corresponding fixed parameter values. The parameter values are ordered by (Journal Option, Device, Block Group, and Inode Size). The triangle marks the point achieved by fixing the values of parameters selected by Carver.*

In Figure 3, performance is measured by the average throughput within each set of configurations, as presented on the X axis. The Y axis shows the stability of each set, measured by the Relative Standard Deviation (RSD) of throughput within the set. We chose to use the RSD rather than variance because the figure shows sets of varying numbers of configurations; RSD is normalized by the configuration count and the average throughput, and thus is easier to compare. If a set of parameters is important, it should ideally lead to a larger average throughput and lower RSD; therefore the best points should cluster in the bottom-right quadrant of Figure 3. As we can see from that figure, fixing just one parameter value (purple dots) causes the mean throughput to range from 2.5Kops/s to around 15Kops/s, and the RSD ranges from around 7% to 76%. The upper-left purple point (2,500, 76%) represents the configurations achieved by setting *Journal Option* to *journal*. The other two points, representing *Journal Options* of *ordered* and *writeback*, turn out to be the best among all purple points. Both are seen near the bottom right with mean throughput of around 15K and




(a) Ext4, Fileserver-default

(b) Btrfs, Fileserver-default

*Figure 5: Carver's ability to correctly find the top 3 important parameters within small portions of the dataset. The X1 (bottom) axis ($log_2$ scale) shows the percentage of the dataset that was used; for each percentage we ran Carver 1,000 times on different, random LHS-compatible subsets of that size. The X2 (top) axis ($log_2$) shows the running time that would be needed to benchmark the selected configurations. We used the PI calculated from the whole dataset as ground truth. The Y axis shows the percentage of runs that were able to correctly find the important parameters. The solid, dashed, and dotted lines show the results for finding the parameters ranked 1st, 2nd, and 3rd, respectively. Note that although Btrfs required a larger percentage of the dataset, the absolute numbers are similar in both figures, and the running times for Btrfs are shorter (see text).*

an RSD value of 7%. Clearly, the *Journal Option* parameter has the highest impact on performance; setting it to an improper value could lead to low throughput and high RSD, while setting it correctly provides significant benefits. The points with $N = 2$ form several clusters. All points with mean throughput less than 9K result from setting *Journal Option* to *journal* (and with another parameter set to various valid values). Conversely, all points with mean throughput larger than 14K result from a *Journal Option* of *ordered* or *writeback*. *Journal Option* is actually the most important parameter selected by Carver (as seen in Table 2).

To probe this question further, we zoomed into the bottom-right part of Figure 3 and added points for $N = 3$ and $N = 4$, as shown in Figure 4. The X and Y axes are similar but with narrower ranges (and the X axis starts at 14K). The label "Max" on the X axis, with a small tick mark, shows the global maximum throughput of all Ext4 configurations within the parameter space. For each $N$, we plotted only the point(s) with the highest average throughput or lowest RSD. The labels around each point show the associated parameter values, ordered by (*Journal Option*, *Device*, *Block Group*, and *Inode Size*). The black triangle marks the point with highest mean throughput, gotten by fixing the values of the three most important parameters selected by Carver. For $N = 1$, the best two points resulted from setting *Journal Option* to either *ordered* or *writeback*. These two points overlap with each other in this figure, as they share nearly identical mean throughput and RSD values. Only one point is plotted for $N = 2$, since the point *(journal_option=ordered, device=ssd)* shows both the highest throughput and the lowest RSD among all $N = 2$ points; the same is true for $N = 3$. For $N = 4$, the left red point shows the lowest RSD value while the right red point shows the highest average throughput. In Figure 4, the top three parameters selected by Carver are *Journal Option*, *Device*, and *Block Size*. By setting the values of these three parameters, the best average throughput (denoted as a triangle in Figure 4) is quite close to the global best average throughput achieved by fixing 3 parameter values (blue point). By comparing the two sets of parameters, we can see that Carver successfully identified the top 2 important parameters; the final average throughput and relative standard deviation achieved by the selected top 3 parameters are quite close to the global optimum. We believe the difference in the 3rd selection is due to two reasons:

1. In Carver, the definition of parameter importance focuses on measuring the impact of the parameter on performance, which can be either positive or negative. When discussing "optimality" in Figure 4, we only considered positive impacts.

2. Carver stops after selecting 3 parameters, as the RSD has already dropped below our 2% threshold at that point. If we removed the stopping criterion, the 4th parameter that Carver would select would be *Block Group*, which aligns with the globally optimal set of top 4 parameters, denoted as red dots in Figure 4.

## 5.4 Carver: Evaluation

All evaluations and analysis in Section 5.2 and 5.3 were conducted on the complete dataset of all possible parameter configurations. However, collecting such datasets for storage parameters is usually impractical, given the challenges discussed in Section 2. One design goal of Carver is to select important parameters while evaluating only a small fraction of configurations. Carver does so by utilizing Latin Hypercube Sampling (LHS), which has been effective in exploring system parameter spaces [27, 42]. We demonstrate the effectiveness of Carver's parameter-selection algorithm from the following two perspectives: selecting important parameters for I/O throughput (see Section 5.4.1) and latency (see

Section 5.4.2).

### 5.4.1 Selecting Important Parameters for Throughput

A critical question is whether Carver can reliably find the important parameters of a system, and how many experimental runs are necessary to do so. To answer this question, we used our entire dataset of experimental runs on *Ext4, Fileserver-default* and *Btrfs, Fileserver-default* to represent the "ground truth" of which parameters matter. For *Ext4, Fileserver-default*, the top 3 important parameters are *Journal Option*, *Device*, and *Block Size*. For *Btrfs, Fileserver-default*, they are *Special Option*, *Node Size*, and *Device*.

We then tested Carver by repeatedly choosing a random subset of the full dataset, simulating a real-world environment in which an experimenter would use LHS to choose configurations to test, and then using the results of those tests to identify important parameters. In all cases we constrained the random subset to be compatible with Latin Hypercube Sampling (LHS), as our hypothetical investigator would do, and tested whether Carver correctly located the first, second, and third most important parameters. We varied the size of the subsets as a percentage of the entire dataset and ran 1,000 iterations of each trial (with different random subsets).

Figure 5 presents the results of running these experiments. The X1 (bottom) axis shows the percentage of the whole dataset that was used by Carver, and is in $\log_2$ scale. The X2 (top) axis shows the actual running time for benchmarking the selected configurations, and is also in $\log_2$ scale. The Y axis shows the fraction of runs that successfully found the same important parameters as the ground truth. The solid, dashed, and dotted lines show the results of finding the 1st, 2nd, and 3rd most important parameters, respectively.

Figure 5(a) shows that even with only 0.1% of the dataset (7 configurations), Carver has a 60% probability of correctly identifying the most important parameter. When using 0.4% (26), Carver was able to find the 1st and 2nd ranked parameter in 100% and 99.8% of the 1,000 runs, respectively. Setting the values of the most important two parameters would already produce high average throughput (97% of the global optimum) with high stability (2% of RSD), as shown in Figure 4. The chance of correctly selecting the third most important parameter increases with the percentage of the dataset used by Carver. With 1% (67) of the dataset, the probability of correctly finding the 3rd parameter is around 50%, while sampling 5% (331) successfully identifies the 3rd parameter in all 1,000 runs.

For Btrfs, shown in Figure 5(b), Carver needed a larger fraction of the dataset to make correct selections. This is because Btrfs has only 288 configurations, compared with 6,624 for Ext4. Yet by evaluating only 16% (45) of all configurations, Carver found the 1st and 2nd parameters with greater than 80% probability. Carver identified the 3rd parameter in more than 80% of runs with 31% (90) sampled.

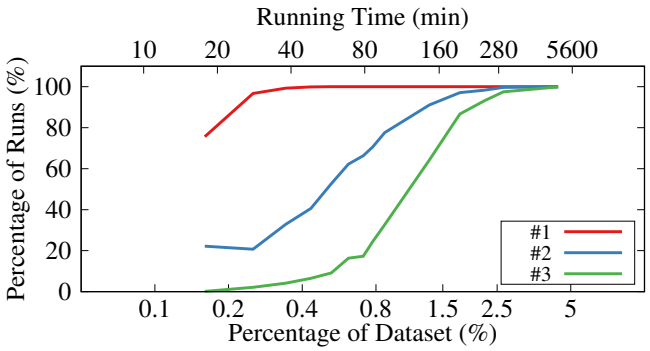



*Figure 6: Carver's ability to correctly find the top 3 important parameters for the latency metric within small portions of the dataset. Experimental settings, graph axes, and legends are the same as in Figure 5.*

### 5.4.2 Selecting Important Parameters for Latency

To further evaluate Carver's effectiveness in selecting important parameters, we collected datasets with latency metrics. The experimental settings were the same as described in Section 5.1. We ran the *Fileserver* workload on the Ext4 configuration with S2 settings (see Table 1). Instead of using the average I/O throughput reported by Filebench, we now used the average latency. Due to a limitation in Filebench's current implementation, it is difficult to collect and calculate accurate tail latency numbers, such as the 99th percentile, so we leave parameter selection for tail latency as future work.

Figure 6 shows the evaluation results of selecting important parameters using the latency metric. The X axis, Y axis, and legends remain the same as in Figure 5. As shown by the red line, with barely 0.2% of all configurations evaluated, Carver was still able to identify the most important parameters in more than 800 out of 1,000 runs. With 1.5% (58 configurations) evaluated, Carver was able to correctly pick the top 2 parameters in almost all the 1,000 runs. Selecting the third most important parameter required a few more evaluation; using 2.5% of the dataset (97 configurations), Carver successfully identified it in 998 runs.

In sum, Carver is effective in selecting parameters using only a few evaluations. In our experiments, Carver found the top 2 important parameters with higher than 80% probability by evaluating fewer than 50 configurations. Fixing the values of the most important two parameters can already result in high and stable system throughput, as shown in Section 5.3. Carver can find the 3rd parameter with about 50% probability using only about 50 evaluations. Furthermore, the total running time for these evaluations is tractable: the worst case, in Figure 6, is under 4 days. Moreover, auto-tuning a storage system with an optimization algorithms often requires an initialization phase to explore the whole space [11, 42]. Carver can use the data collected during the initialization phase to select parameters; in this case, no *extra* evaluation needs to be conducted. Integrating Carver with auto-tuning

algorithms is part of our future work.

## 6 Related Work

**Parameter selection for computer systems.** There have been several attempts to select important parameters for various types of software systems. Aken *et al.* [70] applied Lasso to choose important knobs for databases. They converted categorical parameters into binary dummy features and included polynomial features to deal with parameter interactions. As discussed in Section 3, Lasso does not scale well when the system has many categorical parameters. Plackett-Burman (P&B) design of experiments [53] has been applied to evaluating the impact of parameters in storage benchmarks [51] and databases [18]. However, P&B assumes that each parameter has only two possible values and that the target variable is a monotonic function of the input parameters; neither assumption holds for storage parameter spaces. Adaptive Sampling [19] and Probabilistic Reasoning [64] have been applied to evaluating the impact of database knobs. They either only work for continuous parameters, or have scalability issues in high-dimensional spaces. In comparison, Carver applies variance-based metrics for storage-parameter importance. To the best of our knowledge, we have conducted the first thorough quantitative study of storage-parameter importance by evaluating Carver on datasets collected from a variety of file systems and workloads. Carver also provides insights into the interactions between parameters.

**Auto-tuning storage systems.** Several researchers have built systems to automate storage-system tuning. Strunk *et al.* [63] applied Genetic Algorithms (GAs) to automate storage-system provisioning. Babak *et al.* [4] used GAs to optimize the I/O performance of HDF5 applications. GAs have also been applied to storage-recovery problems [32]. Deep Q-Networks have been successfully applied in optimizing performance for Lustre [40]. More recently, Madireddy *et al.* applied a Gaussian process-based machine learning algorithm to model Lustre's I/O performance and its variability [44]. Our own previous work [11] provided a comparative study of applying multiple optimization algorithms to auto-tune storage systems. However, many auto-tuning algorithms have scalability issues in high-dimensional spaces [61], which is one of the motivations for Carver. Selecting the important subset of parameters could reduce the search space dramatically, which would then benefit either auto-tuning algorithms or manual tuning by experts.

**General feature selection.** Many feature-selection techniques have been proposed in various disciplines. Li *et al.* [39] provide a thorough summary and comparison for most state-of-the-art feature-selection algorithms. Based on our arguments in Section 3, we chose to use variance-based metrics for storage-parameter selection.

## 7 Conclusions

Modern storage systems come with many parameters that affect their behavior. Tuning parameter settings can bring significant performance gains, but both manual tuning by experts and automated tuning have difficulty dealing with large numbers of parameters and configurations. In this paper, we propose Carver, which addresses this problem with the following three contributions:

1. Carver uses a variance-based metric for quantifying storage parameter importance, and proposes a greedy yet efficient parameter-selection algorithm.

2. To the best of our knowledge, we provide the first thorough study of storage-parameter importance. We evaluated Carver across multiple datasets (chosen from more than 500,000 experimental runs) and showed that there is always a small subset of parameters that have the most impact on performance—but that the set of important parameters changes with different workloads, and that there are interactions between parameters.

3. We demonstrated Carver's efficiency by testing it on small fractions of the configuration space. This efficiency gives Carver the potential to be easily applied to new systems and environments and to identify important parameters in a short time, with a small number of configuration evaluations.

In the future, we plan to extend Carver to support other parameter-selection techniques, such as Group Lasso [14,34,74] and ANOVA [9,13,38,72]. We will evaluate and improve Carver with more optimization objectives (e.g., reliability), and even larger storage-parameter spaces. Currently Carver can only measure storage importance for one objective at a time (e.g., throughput, latency). We plan to investigate how to extend Carver's parameter selection algorithm into the problem of multi-objective optimization [17]. We also plan to integrate Carver with auto-tuning algorithms [11].

## Acknowledgments

We thank the anonymous FAST reviewers and our shepherd, Bill Bolosky, for their valuable comments. This work was made possible in part thanks to Dell-EMC, NetApp, and IBM support; and NSF awards CCF-1918225, CNS-1900706, CNS-1729939, and CNS-1730726.

## 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] George Amvrosiadis and Vasily Tarasov. Filebench github repository, 2016. *https://github.com/filebench/filebench/wiki*.

[4] 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.

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

[6] Dhruba Borthakur et al. HDFS architecture guide. *Hadoop Apache Project*, 53, 2008.

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

[8] Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Luján. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. *Journal of Machine Learning Research*, 13(Jan):27–66, 2012.

[9] Morton B. Brown and Alan B. Forsythe. Robust tests for the equality of variances. *Journal of the American Statistical Association*, 69(346):364–367, 1974.

[10] 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.

[11] Zhen Cao, Vasily Tarasov, Sachin Tiwari, and Erez Zadok. Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems. In *Proceedings of the Annual USENIX Technical Conference*, Boston, MA, July 2018. USENIX Association. Data set at *http://download.filesystems.org/auto-tune/ATC-2018-auto-tune-data.sql.gz*.

[12] 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.

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

[14] Christophe Chesneau and Mohamed Hebiri. Some theoretical results on the grouped variables Lasso. *Mathematical Methods of Statistics*, 17(4):317–326, 2008.

[15] Liu Chu, Eduardo Souza De Cursi, Abdelkhalak El Hami, and Mohamed Eid. Reliability based optimization with metaheuristic algorithms and Latin hypercube sampling based surrogate models. *Applied and Computational Mathematics*, 4(6):462–468, 2015.

[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 (HOTOS '05)*, 2005.

[17] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[18] Biplob K. Debnath, David J. Lilja, and Mohamed F. Mokbel. SARD: A statistical approach for ranking database tuning parameters. In *IEEE 24th International Conference on Data Engineering Workshop (IDEW)*, pages 11–18, 2008.

[19] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. Tuning database configuration parameters with iTuned. *Proc. VLDB Endow.*, 2(1):1246–1257, August 2009.

[20] Pablo A. Estévez, Michel Tesmer, Claudio A. Perez, and Jacek M. Zurada. Normalized mutual information feature selection. *IEEE Transactions on Neural Networks*, 20(2):189–201, 2009.

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

[22] Ext4 documentation. *https://www.kernel.org/doc/Documentation/filesystems/ext4.txt*.

[23] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 29–43, Bolton Landing, NY, October 2003. ACM SIGOPS.

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

[25] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182, 2003.

[26] 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.

[27] Jun He, Duy Nguyen, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Reducing file system tail latencies with Chopper. In *Proceedings of the 13th*

*USENIX Conference on File and Storage Technologies*, FAST'15, pages 119–133, Berkeley, CA, USA, 2015. USENIX Association.

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

[29] Aapo Hyvärinen and Erkki Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

[30] M. Kaminsky, G. Savvides, D. Mazieres, and M. F. Kaashoek. Decentralized user authentication in a global file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, Bolton Landing, NY, October 2003. ACM SIGOPS.

[31] Jalil Kazemitabar, Arash Amini, Adam Bloniarz, and Ameet S Talwalkar. Variable importance using decision trees. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 426–435. Curran Associates, Inc., 2017.

[32] 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*, pages 235–248, New York, NY, USA, 2006. ACM.

[33] 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.

[34] Seyoung Kim and Eric P. Xing. Tree-guided group Lasso for multi-task regression with structured sparsity. In *ICML*, pages 543–550, 2010.

[35] 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.

[36] Latin hypercube sampling. *https://en.wikipedia.org/wiki/Latin_hypercube_sampling*.

[37] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. F2FS: A new file system for flash storage. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST)*, pages 273–286, Santa Clara, CA, February 2015. USENIX Association.

[38] Howard Levene. Robust tests for equality of variances. *Contributions to Probability and Statistics. Essays in Honor of Harold Hotelling*, pages 279–292, 1961.

[39] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Computing Surveys (CSUR)*, 50(6):94, 2017.

[40] Yan Li, Kenneth Chang, Oceane Bel, Ethan L. Miller, and Darrell D. E. Long. 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.

[41] 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.

[42] Zhao Lucis Li, Chieh-Jan Mike Liang, Wenjia He, Lianjie Zhu, Wenjun Dai, Jin Jiang, and Guangzhong Sun. Metis: robustly optimizing tail latencies of cloud systems. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, pages 981–992. USENIX Association, 2018.

[43] Yijuan Lu, Ira Cohen, Xiang Sean Zhou, and Qi Tian. Feature selection using principal feature analysis. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 301–304. ACM, 2007.

[44] Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert Latham, Robert Ross, Shane Snyder, and Stefan M Wild. Machine learning based parallel i/o predictive modeling: A case study on lustre file systems. In *International Conference on High Performance Computing*, pages 184–204. Springer, 2018.

[45] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[46] 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.

[47] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In *Proceedings of the IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing*, pages 41–48. IEEE, 1999.

[48] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[49] John A. Nelder and Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[50] John Neter, Michael H. Kutner, Christopher J. Nachtsheim, and William Wasserman. *Applied Linear Statistical Models*, volume 4. Irwin Chicago, 1996.

[51] Nohhyun Park, Weijun Xiao, Kyubaik Choi, and David J. Lilja. A statistical evaluation of the impact of parameter selection on storage system benchmarks. In *Proceedings of the 7th IEEE International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, volume 6, 2011.

[52] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM SIGMOD*, pages 109–116, Chicago, IL, June 1988. ACM Press.

[53] Robin L. Plackett and J. Peter Burman. The design of optimum multifactorial experiments. *Biometrika*, pages 305–325, 1946.

[54] LVM2 resource page. *http://sources.redhat.com/lvm2/*.

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

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

[57] 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.

[58] George A.F. Seber and Alan J. Lee. *Linear Regression Analysis*, volume 329. John Wiley & Sons, 2012.

[59] Priya Sehgal, Vasily Tarasov, and Erez 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.

[60] Priya Sehgal, Vasily Tarasov, and Erez Zadok. Optimizing energy and performance for server-class file system workloads. *ACM Transactions on Storage (TOS)*, 6(3), September 2010.

[61] 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.

[62] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[63] 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.

[64] David G. Sullivan, Margo I. Seltzer, and Avi Pfeffer. *Using probabilistic reasoning to automate software tuning*, volume 32. ACM, 2004.

[65] 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.

[66] Vasily Tarasov, Saumitra Bhanage, Erez Zadok, and Margo 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.

[67] 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.

[68] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

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

[70] 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.

[71] 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.

[72] Bernard Lewis Welch. On the comparison of several mean values: An alternative approach. *Biometrika*, 38(3/4):330–336, 1951.

[73] 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.

[74] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the*

*Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.