# Accelerating the Configuration Tuning of Big Data Analytics with Similarity-aware Multitask Bayesian Optimization

Ayat Fekry*, Lucian Carata*, Thomas Pasquier†, Andrew Rice*
*Computer Laboratory, University of Cambridge, UK
†Department of Computer Science, University of Bristol, UK

***Abstract*—One of the key challenges for data analytics deployment is configuration tuning. The existing approaches for configuration tuning are expensive and overlook the dynamic characteristics of the analytics environment (i.e. frequent changes in workload due to receiving evolving input sizes or change in the underlying cluster environment). Such workload/environment changes can cause significant performance degradation, with retuning the configuration to accommodate those changes can yield up to 85% potential execution time saving.**

**We propose SimTune, an approach that accommodates such changes through efficient configuration tuning. SimTune combines workload characterization and Multitask Bayesian optimization to identify similarity across workloads and accelerate finding near-optimal configurations. Our experimental results show that SimTune reduces the search time for finding close-to-optimal configurations by 56-73% (at the median) when compared to existing state-of-the-art techniques. This means that the amortization of the tuning cost happens significantly faster, enabling practical tuning in the rapidly changing environment of distributed analytics.**

## I. Introduction

With the ever growing data volume [3], the timely extraction of useful insights from data remains a challenge. The need to analyse large datasets has led to the wide adoption of Data Intensive Scalable Computing (DISC) Systems such as Hadoop [5], Spark [2] and Flink [4]. These DISC systems enable the manipulation and analysis of large amounts of data by distributing work over a cluster of machines. They are used to help organizations make better and faster decisions, as well as in research areas that range from biology [18] and physics [23] to astronomy [42].

One of the challenges in setting up DISC systems is to identify the right configuration in order to optimize processing speed. Misconfiguration can lead to either resource contention/exhaustion or under-utilization, with the former potentially triggering errors hours after the start of execution. Consequently, developers spend significant time and resources identifying the appropriate configuration for their workload. This laborious effort motivated work towards the automation of configuration tuning [12], [19], [28]–[30], [40].

Generally, such explorations follow one of two strategies: either search based or model-based, wherein the latter a performance prediction model is built to guide the tuning. Both strategies require hundreds of executions to perform high dimensional configuration tuning [41], [43], with the cost of performing the optimisation itself starting to amortize as a function of both how fast the algorithm converges to very good configurations and how many bad (slow) configurations are explored in order to get there.

Virtually all of the existing work for DISC systems configuration tuning assume a *static* environment, such as a stable workload running in a fixed underlying cluster. Therefore, it is assumed that changes in the workload/environment are infrequent enough that re-tuning from scratch is cost effective. Re-tuning happens either by re-learning the models or restarting the search algorithm, which is slow and prohibitively expensive. Ignoring any prior knowledge acquired during workload tuning poses poor adaptivity to such changes and slows down the tuning process, incurring high re-tuning costs.

In a rapidly changing environment such as big data systems, it is crucial to perform tuning/retuning efficiently using a small number of executions, which eventually will accelerate the amortization of tuning costs. We propose solving this problem using a similarity-aware multitasked tuning approach (SimTune). We leverage neural encoding of workload execution metrics to detect workload similarities, then share the tuning knowledge gained previously across the similar workloads using multitask Bayesian Optimization (BO). Our main focus is on *how* to leverage workload similarity to accelerate the configuration tuning. We consider an existing *base* tuner and address accelerating this base tuner using our proposed tuning approach. The key contributions of this work are:

- Detecting workload similarity using a novel representation learning approach that transforms workload execution metrics into a low dimensional space using nonlinear neural encoding and minimizes the information loss compared to earlier work.
- Tuning the configuration parameters incrementally using a small number of workload executions, by employing multitask BO to share the gained tuning knowledge across the similar workloads.
- Obtaining comparable configurations to prior work but converging to those significantly faster, enabling a quicker amortization of the tuning costs.

SimTune is the first work to study *how* to share the tun-

978-1-7281-6251-5/20/$31.00 ©2020 IEEE

ing knowledge across similar workloads to enable *data-efficient* [15] tuning of DISC systems configurations. The source code and experiment dataset is available at [9].

## II. Background

BO [31] is a method for minimizing blackbox functions $f$ iteratively, using a limited number of samples. This is useful when it is expensive to evaluate $f$ at a given point (such as running a big-data workload with a given configuration). BO is characterized by its *prior model* and *acquisition function*: the prior model represents a space of possible target functions $f$, and the acquisition function guides the selection of the next evaluation point based on the prior knowledge.

One of the widely accepted prior models for BO is Gaussian Process (GP). It represents a distribution over functions (a sample drawn from this process is a function) with given mean and covariance. Here, the mean function describes expected values at each point and the covariance function defines the smoothness of the functions which can be drawn as samples, encoding prior assumptions about the data that we want to model [32]. The GP maintains a probabilistic belief about what functions $f$ are possible, given known characteristics and already seen data. This belief is updated by using an acquisition function, which determines the best point of $f$ to sample next. After sampling, the prior belief about possible functions $f$ is updated and a new sampling decision can be made, iteratively. At each step, the posterior distribution has filtered-out functions not consistent with the sampled data and will ideally have a narrower candidate function space. The GP acquisition function represents the metric by which the GP picks the next input sample to improve the probabilistic model function. It is typically a function that is cheap to evaluate at a given point $x$ and its value is proportional to how useful evaluating $f(x)$ would be for the optimisation problem. Various acquisition functions have been proposed to define the way the GP samples the input space, e.g., random, sequential, Probability of Improvement (PI), and Expected Improvement (EI) [20], [35], [38].

Multitask Bayesian Optimization (MTBO) is an extension of the standard BO to enable modelling and minimizing different tasks. This modelling principally relies on defining a covariance function between input-task pairs. While the covariance function of the standard GP defines the relationship between inputs, the covariance function of Multitask GP (MTGP) defines the relationship between inputs and tasks. It learns the degree of correlation between tasks and utilizes this information to guide the search for the best point to sample. Figure 1 shows an example MTGP with three tasks, 1(a) shows the actual functions of the three tasks. Task 2 and 3 are strongly correlated, 1 and 3 are anti-correlated, and 1 and 2 are not correlated. The goal is to model task 3, the dots represent observations and the dashed lines represent the predictive mean function. The curve at the bottom represents the acquisition function(e.g., Expected Improvement(EI)) for each input location on Task 3, the next point to sample is the one with the maximum acquisition (maximum EI). The confidence interval indicates the range that the actual function should fall in with a high probability, the narrower the better. Whereas the independent GP in 1(b) has a wide confidence interval and optimizing EI leads to a false minimum as the next point to sample, the multitask GP in 1(c) leverages the correlation across the tasks, thus the confidence interval is improved (narrowed) and the maximal EI point corresponds to the true minimum. We discuss MTGP as it applies to SimTune in § III.

## III. Similarity-aware Multitask Tuning

To accommodate the need for efficient configuration tuning, we propose a similarity-aware tuning approach that leverages the tuning knowledge between similar workloads and effectively accelerate the tuning process.

**Assumptions:** We assume the existence of readily available workloads tuned using a *base* tuner.

We chose Tuneful [17] as our *base* tuner, since it finds a configuration comparable to the-state-of-the-art significantly faster and determines workload-specific influential parameters. Moreover, it tunes the configurations using single tasked GP (STGP), which has been widely adopted in several configuration tuning systems ( [12], [16]).

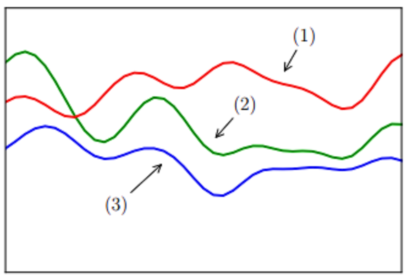
Our work addresses *how* to define similarity across workloads and *efficiently* leverage this similarity in twofold: 1) tune a new similar workload, 2) retune already seen workload to accommodate the ever growing input sizes. We start with learning how to characterize workloads to detect workload similarity, then employ MTBO to accelerate the tuning of these similar workloads.

### A. Workload Characterization

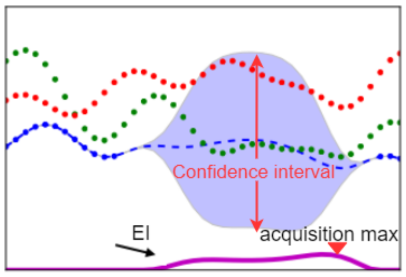
As a first step, we identify the main components distinguishing each workload so that we detect the similarity across workloads. To achieve this, we monitor the workload's execution metrics then learn the main aspects that represent each workload.

*1) Workload Monitoring::* We capture metrics that involve CPU time, number of tasks per stage, number of stages per job, input and output size, Garbage Collection (GC) time, execution time, data serialization/deserialization time, the size of shuffled data, memory spilled data and disk spilled data. We collect all the numeric metrics that relate to those workload execution features, the number of collected metrics is 24. The overhead of collecting these metrics is minimal as we leverage the existing Spark logs.

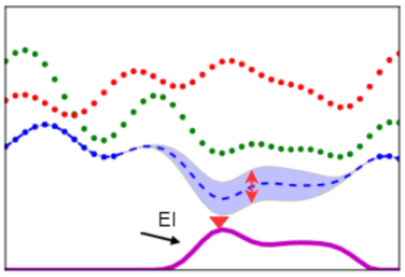
To build representative statistics of each metric, for each stage in the analytics workload, we calculate the 95% quantile of the metric across the tasks of the stage. Then, we average the captured metric across all the stages and represent them in terms of ratios. For example, we consider the amount of GC time relative to the total CPU time and the amount of shuffled data with respect to the total input data, rather than the quantitative values of each metric. A decision made since we ultimately use this characterization to detect similarity between workloads, in our context two workloads are similar if they



(a) Multitask GP sample functions



(b) Independent GP Modelling



(c) Multitask GP Modelling

Fig. 1: (a)The actual functions of a Multitask GP with three tasks. Task 2 and 3 are strongly correlated, 1 and 3 are anti-correlated, and 1 and 2 are not correlated. (b) Independent single tasked GP modelling for Task 3. (c) Multitask GP modelling for Task 3, utilizing the other tasks (figure source is [36] with minor edits applied for more clarity).

inherit similarity in terms of their relative resource usage- not the quantitative resources usage. Intuitively, the workloads that are similar in their relative resource usage are expected to have similar influential parameters.

*2) Workload Representation Learning::* The existing work on detecting similarities relies on first projecting the workload execution features in a low dimensional space then calculating the distance within this space. This is due to the fact that distance metrics such as $L_p$ norm become less informative in the high dimensional space [10]. We embrace the same approach but differentiate in the way we represent the workload, by leveraging neural encoding to encode workload execution features in a low dimensional space. Previous work employs PCA for this purpose. However, by projecting the data in a linear space some of the nonlinear dependencies in the latent space are lost, leading to a higher data reconstruction loss. We employ nonlinear neural autoencoder to lower this loss. More details on evaluating workload representation are in § IV-B.

An autoencoder (AE) is a neural network that learns how to represent data in a low dimensional space. It has been widely adopted to encode data in different domains such as image processing, natural language processing and signal compression. It consists of two components: encoder and decoder, the encoder is a neural layer(s) learns $f(x)$, which transforms input $x \in \mathbb{R}^d$ into $h \in \mathbb{R}^n$ latent space, such that $n < d$. The decoder learns $g(h)$, which reconstructs the latent low dimensional $h$ back to $\hat{x}$ of the same dimensionality as the original input space. The AE compares the decoder's output (reconstructed input) $\hat{x}$ to the encoder's original input $x$ and update the neurons' weights accordingly to minimize the loss of the re-constructed input $L$ as shown in Equation 1, where $L$ is a function that penalizes $g(f(x))$ for not being similar to $x$ (e.g., the mean square error).

$$L(x, \hat{x}) = ||x - g(f(x))||^2 \tag{1}$$

As illustrated in Figure 2, our AE learns the workload representation as an offline phase. We use a one layer nonlinear AE to encode the execution features into a low dimensional latent space. We made this decision as it minimizes the reconstruction error compared to linear AE and PCA, moreover it has a smaller number of model parameters compared to multilayer AE.

*3) Similarity Analysis::* The aim of the similarity Analysis is to find a *source* -already tuned- workload and share its influential parameters with the *target* workload (the one under tuning). The source workload is the most similar one based on the Manhattan distance between the encoded execution features $h$. We use Manhattan distance as it provides more contrast compared to other $L_p$ norm distance metrics [10].

The distance is calculated between workloads running under the *same* configuration. We use a *fixed* representative configuration to run the workload once and extract workload execution metrics using the learnt workload representation (this works as workload fingerprint). We then match workload fingerprint to one of the seen workloads. We experimented with varying the number of workload executions used for workload fingerprinting and how this influences the accuracy of workload matching. Using a single representative configuration was enough to provide the same matching as multiple executions, so we opted to use a single execution to limit the cost of workload fingerprinting. Two workloads are considered similar if they have the smallest distance in terms of their relative resource usage, since this implies that they have similar influential parameters. Figure 2 shows how the similarity analysis happens online, upon receiving a new *target* workload, as a first step, the Tuning Manager checks if this workload has a known similar workload or not. If not, it suggests the fingerprinting configuration to the Spark driver, which then forwards to the Spark workers to execute the workload. After workload execution, the Tuning Manager provides the workload's execution metrics to the Similarity Analyzer, which encodes the execution metrics of this workload- leveraging the learned workload representation from the offline phase. Then the workload with the smallest distance is selected as the source workload and shares its influential parameters with the target workload.

## B. Multitask Configuration Tuning

After finding a similar *source* workload- tuned using a *base* tuner that we address accelerating it using SimTune.

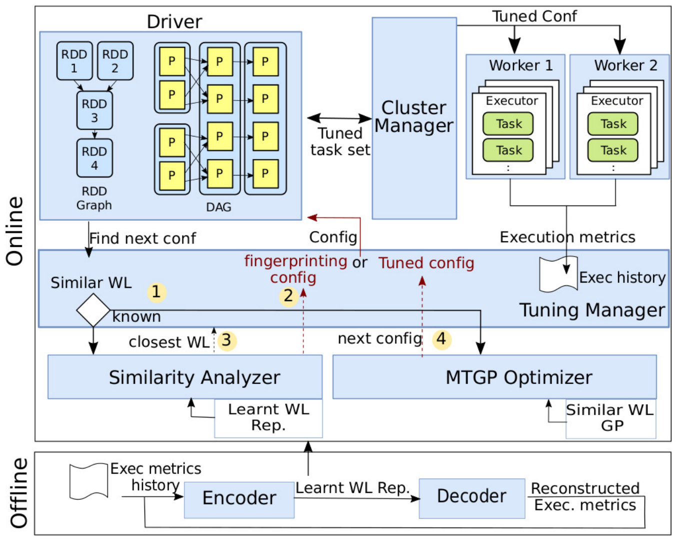


Fig. 2: SimTune in the offline and online phases to tune Spark

We transfer the knowledge gained during tuning the *source* workload to the *target* workload. This knowledge includes: 1) the significant configuration parameters. In DISC systems, not all the workloads share the same significant parameters, this is due to the high diversity of the applications served (e.g., graph analytics, machine learning, SQL, text analysis). SimTune shares the knowledge of these workload-specific significant parameters across similar workloads. we assume that those significant parameters are determined using our base tuner . 2) the execution samples evaluated during tuning this source workload.

The Tuning Manager passes the tuning knowledge of the source workload to the MTGP Optimizer to reuse the significant configuration parameters and leverages the previously seen execution samples. This knowledge guides the MTGP Optimizer to suggest the next configuration sample, with the highest potential to maximize the EI. In the beginning, it suggests the configuration samples that help learning the covariance across the different workloads (modelled as tasks). Then it decides the next sample to try based on this covariance (i.e. if the covariance is strong, it picks the sample that maximizes the EI of the source task, otherwise it picks the sample that maximizes the EI of the target task). More details on how MTGP works is in [36].

In Alg. 1, we show the steps of our multitask configuration tuning, its input arguments are as follows:

- $w$ target workload;
- $W_{seen} = \{w_1, w_2, ..\}$ the matrix of the encoded execution features for the seen workloads;
- $\alpha$ the acquisition function;
- $GP = \{GP_1, GP_2, ..\}$ the Gaussian process models of the seen workloads;

Upon receiving a new *target* workload $w$ and following fingerprinting it to find its execution metrics $w_{metrics}$ (as described in § III-A3), the algorithm works as follows: We start at line 2 with calculating the distance between the execution metrics of the target workload $w_{metrics}$ and the auxiliary workloads $W_{seen}$, the distance is calculated based on the learnt low dimensional encoding of workload execution metrics. We then select workload $s$ with the smallest distance to use as the *source* task for the multitask tuning at line 3. The significant parameters of $s$ is reused to tune $w$ and a new task $t_j$ is added to the GP of $s$ for $w$ (line 5). At line 6, we leverage the GP of $s$ to select the next configuration to execute $w$, the next configuration is selected so that it maximizes $\alpha$ acquisition function over task $t_j$. Lastly, workload $w$ is executed using the selected configuration (line 7) and the execution cost $C$ is used to update the GP model of task $t_j$( line 8).

**Algorithm 1:** Similarity-aware Multitasked configuration tuning

```
  Input  : $\alpha, w, W_{seen}, GP$
1 forall $x_i \in W_{seen}$ do
2     calculate distance $d$ between $w_{metrics}$ and $x_i$;
3 Find workload $s$ with the smallest $d$ ;
4 Transfer $s$ significant parameters to $w$ ;
5 Add new task $t_j$ for workload $w$ to $GP_s$ ;
6 Find config: $x_i \leftarrow \arg\max_x \alpha(GP_s(x, t_j))$ ;
7 Evaluate config: $y_i \leftarrow C_w(x_i)$ ;
8 Update: $GP_s \leftarrow GP_s|(t_j, x_i, y_i)$ ;
```

Figure 2 shows how the tuning takes place online during the normal workload executions. The MTGP Optimizer suggests the next configuration to explore, then the Tuning Manager feeds the execution cost[1] back to the MTGP optimizer to update the MTGP model. The MTGP optimizer detects if it is time to stop the optimization. It stops modelling a workload after suggesting a minimum of n samples (e.g 10 samples) and then after the expected improvement (EI) drops below 10%. A decision made to make sure that we balance between the exploration of the tuning space and exploitation of the best configuration found.

**Resilience to workload mismatching:** we can detect inaccuracies in workload matching through monitoring the gap between the predicted execution time by the MTGP model and the actual execution time. If a continuous degradation takes place, then we trigger workload matching again (given that we have seen sufficient workloads, otherwise we perform a standalone tuning for this workload using the base tuner). Even if a workload mismatching happens due to the limited number of seen workloads, SimTune guarantees that the found configuration is not worse than the reused state-of-the-art's configuration, which implies reusing the configuration suggested by the state-of-the-art for a similar workload (we refer to this as Direct Transfer and illustrate that more in § IV).

[1]We focus on execution time through the paper, but the algorithm works similarly for any defined cost function.

| Application | Abbr. | Input data sizes (DS) |
|---|---|---|
| Pagerank | PR | 5, 10, 15, 20, 25 (million pages) |
| TPC-H benchmark | TPCH | 20, 40, 60, 80, 100 (compressed GB) |
| Terasort | TR | 20, 40, 60, 80, 100 (GB) |
| Bayes Classifier | Bayes | 5, 10, 30, 40, 50 (million pages) |
| Wordcount | WC | 32, 50, 80, 100, 160 (GB) |

TABLE I: The set of applications and input sizes used to learn workload representation.

SimTune is implemented on top of Spearmint [33], a Python BO framework that implements MTBO based on [36].

## IV. Evaluation

### A. Experimental setup

**Cluster and configuration specification:** We use a cluster of 4 AWS *h1.4xlarge* instances with 16 vCPUs, 64 GB memory, and 2TB storage each. We use HDFS [7] version 2.7 for accessing the shared data and Spark version 2.2.1 as the system under tuning. Our base tuner tunes 30 configuration parameters, with approximatively $2 \cdot 10^{40}$ configurations possible in total (this represents the size of the search space), a list describing each configuration parameter and its range can be found in the project repository [8]. We use the same ranges when evaluating the other tuning approaches.

We start with evaluating the workload representation learning then evaluate SimTune effectiveness and efficiency.

### B. Workload Representation Learning

**Environment setup:** We used Keras v2.3.0 to build the AE, with epoch (the number of complete passes over the dataset to update the internal model parameters) of 300 and a sigmoid activation function. We tried the other possible nonlinear activation functions and selected the one with the minimal information loss. For setting the epoch, we searched till 1000 and the information loss converged at 300, we use the same number of epoch for the linear AE.

**Dataset:** We built a dataset of execution metrics using two well known big data benchmarks (Hibench [22] and TPC-H [1]). We selected five applications of heterogeneous characteristics from these benchmarks. For each application we experimented with five different input sizes, we captured the execution metrics of each application and input size pair under 100 random configurations for the 30 configuration parameters in [8], with a total of 2200 application executions that took 2188 compute hours to execute. We do not consider this time as an overhead of our tuning approach as this data will be readily available over time in a deployed system. Table I lists the applications and input sizes, the dataset is publicly available at the project repository [9].

To evaluate the accuracy of learning how to represent workload's execution features in a low dimensional space, we compare the loss of our representation using nonlinear AE against PCA and linear AE. Figure 3 shows the reconstruction loss over the number of components/encoded dimensions during learning the representation using PCA, linear AE and

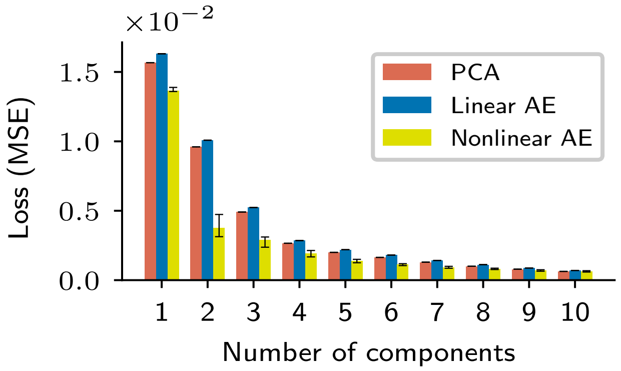


Fig. 3: Reconstruction loss over the number of encoded low dimensional components, showing median, 90th and 10th percentile of 10 experiments.

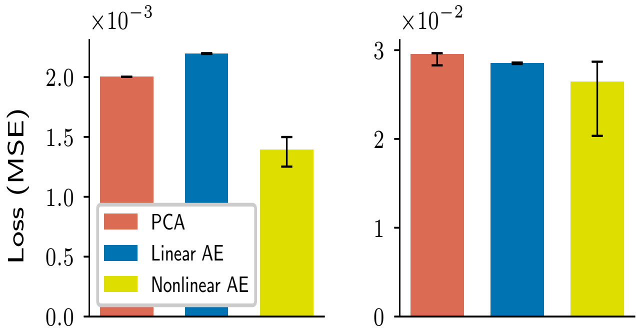


Fig. 4: Reconstruction loss for 20% holdout validation dataset (left) and test dataset of 500 workloads execution metrics running on a different cluster (right).

nonlinear AE at the median, 90th and 10th percentile. The nonlinear encoder can represent the execution metrics using a smaller number of components/dimensions while maintaining less reconstruction loss compared to PCA and the linear AE. For example, at the 90th percentile the nonlinear AE can encode the execution metrics using 5 dimensions with a loss less than PCA and linear AE using 6 dimensions, since they restrict the encoding of data into a linear space that does not capture the nonlinear relationships across the execution metrics. We chose to represent the workload execution metrics using 5 dimensions, representing a good compromise between the reconstruction loss and the number of components for all the three approaches (e.g., PCA represents 92% of the data variance using 5 components).

We excluded 20% of the dataset as a hold out validation dataset to evaluate the learnt representation. Further, to evaluate the generalization of the learnt representation in a different environment, we built a test dataset that includes execution metrics of 5 applications running on a cluster of a different characteristics: a 20 Google Compute Engine [27] instances (1 driver + 19 workers), with the driver being an

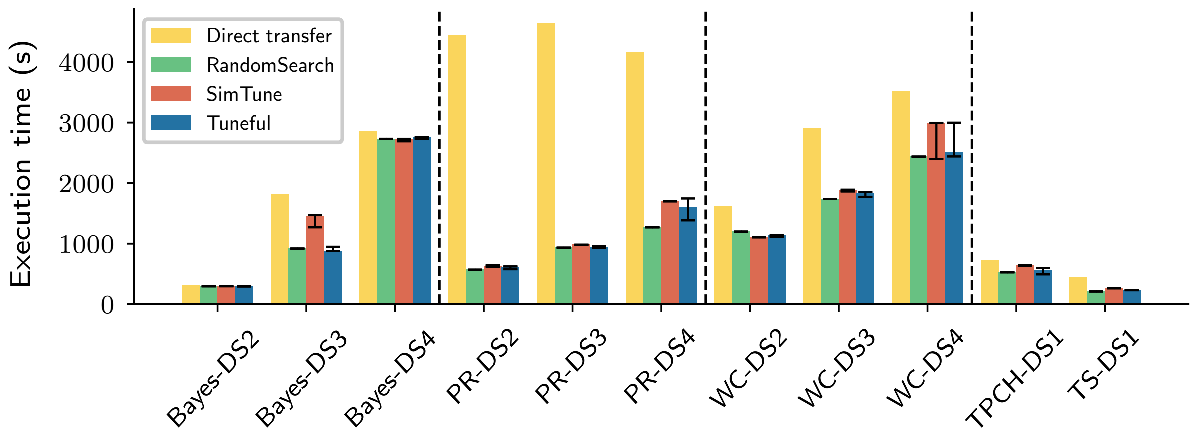


Fig. 5: Execution time of the direct transferred configuration, the tuned configurations using Tuneful, Random search and SimTune.

| Workload | Bayes-DS2 | Bayes-DS3 | Bayes-DS4 | PR-DS2 | PR-DS3 | PR-DS4 |
|---|---|---|---|---|---|---|
| **RandomSearch** | 12 | 1915 | 769 | 330 | 3147 | 4265 |
| **Tuneful** | 12 | 1168 | 684 | 416 | 1261 | 2452 |
| **SimTune** | 5 | 511 | 46 | 229 | 258 | 739 |

| Workload | WC-DS2 | WC-DS3 | WC-DS4 | TPCH-DS1 | TS-DS1 |
|---|---|---|---|---|---|
| **RandomSearch** | 2351 | 452 | 1161 | 151 | 423 |
| **Tuneful** | 719 | 452 | 952 | 54 | 180 |
| **SimTune** | 111 | 612 | 1247 | 208 | 117 |

TABLE II: The median search time (m) till finding 10% of the optimal configuration using Tuneful, Random search and SimTune. The green values represent the total search time when a configuration within 10% of the optimal is not found.

*n1-highmem-8* instance and the 19 workers being *n1-standard-16* instances. The dataset contains the execution metrics of 500 executions of TPC-H, PR, bayes, WC, TR and kmeans workloads. Figure 4 shows the reconstruction loss for the validation and test dataset. The nonlinear AE maintains the lowest loss for the validation dataset at the 90th percentile and generalizes better than the other approaches for the test dataset, with a smaller reconstruction loss at the median.

### C. Tuning Effectiveness and Efficiency

We compare the configurations picked by our approach with the configuration tuned by: 1)*Direct transfer*, which implies transferring the tuned configuration from the source workload directly to the target workload. We compare against this to show the importance of retuning the configuration. 2)*Tuneful*, our base tuner and the state-of-the-art GP-based DISC system tuning approach, it starts with detecting workload-specific significant parameters then tunes them using single tasked GP [17]. We chose Tuneful to assess the accuracy of reusing the significant parameters of a similar workload against performing an independent detection of the significant parameters and tuning. 3) *Transfer learning and single tasked GP tuning* (TL+STGP), which implies transferring the significant parameters from a similar workload then performing configuration tuning using Single Tasked Gaussian Process (STGP). We compare against this approach to evaluate the benefits of using MTGP against STGP. 4) For the sake of completeness, we also compare the tuned configuration against *RandomSearch*, with a budget of 100 execution samples and the random configurations are generated using low-discrepancy sequences [34], since they cover the search space quicker and more evenly than the standard random numbers.

**Metrics:** We use three metrics to evaluate our proposed approach: 1) The execution time of the tuned configuration, akin to the running cost of the tuned configurations. The target here is to obtain tuned configurations similar to what state-of-the-art achieve. 2) Search Cost, the amount of time and actual cost (in $) required by each approach to find good configurations while repeatedly running workloads in a cloud environment. The target is to get close-to-optimal configurations (within 10% of the estimated best configuration) significantly faster than the state-of-the-art. 3) *Amortization speed*, the number of needed workload executions to amortize the tuning cost. The target is to amortize the tuning cost after a small number of workload executions.

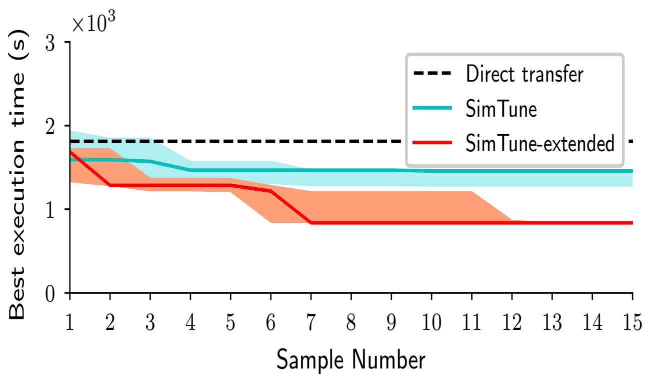


Fig. 6: Convergence speed of the Bayes workload when the significant parameters are transferred from Bayes-DS2.

The results are always presented as the median of 10 experiments, bars represent the 10th and 90th percentile.

**Methodology:** To mimic the big data analytics environment when they receive similar workloads or growing input sizes, we consider a set of *Auxiliary workloads* that represents the seen workloads, then evaluate tuning a set of similar workloads with evolving input sizes.

**Auxiliary workload set:** consists of three workloads from three applications (bayes-DS1, PR-DS1 and WC-DS1), each tuned using Tuneful [17] (our base tuner), which performs an independent significant parameter detection and tuning for each workload. We excluded two applications (TPC-H and TS) from the auxiliary set workload to evaluate the effectiveness of tuning unseen applications.

To evaluate the effectiveness of SimTune in finding good configurations, for each application in the auxiliary workload set, we tune three workloads of evolving input sizes using SimTune and compare against direct transfer, Tuneful, and RandomSearch. We also consider tuning two workloads of two applications that were not included in the auxiliary workload set (TPC-H-DS1 and TS-DS1).

**SimTune finds configurations comparable to the state-of-the-art outperforming the direct transfer by 32% at median and 79% at the 90th percentile:** Figure 5 shows the execution time of the configurations found by the different tuning approaches, the direct transfer of the configurations does not guarantee near-optimal performance, with a potential performance degradation of up to 85% compared to retuning the configuration. Across all workloads, SimTune finds on average a configuration with an execution time within 8% of Tuneful's and 12% of RandomSearch's. This average is computed over the difference in median execution times for each workload that are shown in Figure 5.

However, for some workloads (e.g., Bayes-DS3 and WC-DS4) Tuneful found configurations outperform the ones found by SimTune by up to 36%. This is due to the limited set of auxiliary workloads. The next section describes how this can be mitigated through extending the set of auxiliary workloads.

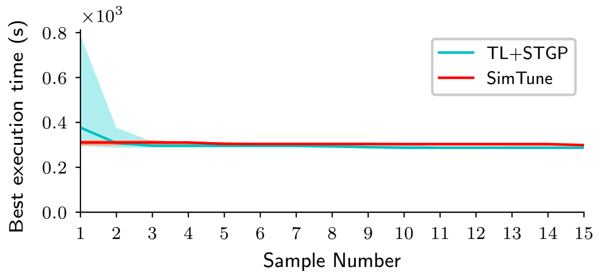
To evaluate differences in time taken to find configurations close-to-optimal, we allow each approach to execute workloads until it finds the first configuration resulting in a runtime within 10% of the one produced by the estimated optimal configuration. This is defined as the best configuration ever found across all our tests for each workload, irrespective of the tuning approach or experiment. Table II shows the search time of each approach. Since for some workloads SimTune and Tuneful do not find a configuration within 10% of the estimated optimal configuration, we color these workloads in green in the table, the value for those workloads represent the total search time to find configuration has the execution time shown in Figure 5.

**The median search time for the-state-of-the-art is 2.3-3.7X longer when compared to SimTune:** Table II shows that overall and given this limited auxiliary workloads, SimTune significantly accelerate the search time of 6 workloads while finding a configuration within 10% of the estimated optimal. For the remaining workloads, the total search time of SimTune is still considerably smaller than the other approaches, while not only notably outperforming direct transfer, but also finding a configuration comparable to Tuneful for most of the workloads and within 17%-37% of RandomSearch.

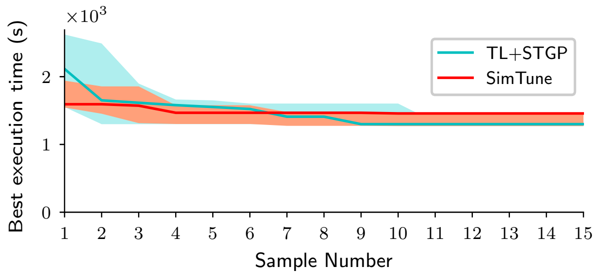
This suggests that a significantly shorter search phase is possible even with a limited number of source workloads to transfer tuning data from. However, in around 45% of the cases this also means that the reached configuration is slightly further away from the optimum. This can be mitigated when a larger set of "auxiliary workloads" exists as shown in Figure 6.

**Extending the tuning knowledge for SimTune enables a faster and more effective tuning:** Table II shows that Tuneful takes 2.5X longer time than SimTune for Bayes-DS3, while finding configuration outperforms direct transfer by 54% and saves execution time by 36% compared to the one found by SimTune. This is due to the limited set of auxiliary workloads, resulting SimTune to find configuration not within 10% of the optimal and the time in the table represents the total search time for this workload. However, Figure 6 shows the convergence speed of SimTune when extending the auxiliary workload set to tune the Bayes-DS3 workload. The area between the 90th and 10th percentile of execution time is shaded, and the line shows the median of 10 experiments. We plot the minimum execution time found by the tuning algorithm until each execution sample on the x axis. Extending the auxiliary workload set to include Bayes-DS2 workload allows SimTune to select Bayes-DS2 as the source workload, this enables the tuning to happen using 38% less search time compared to using a smaller auxiliary workload set, while finding a configuration comparable to Tuneful's outperforming the direct transfer by 54%. In comparison, the one found using a smaller auxiliary workload set outperforms direct transfer by 19%.

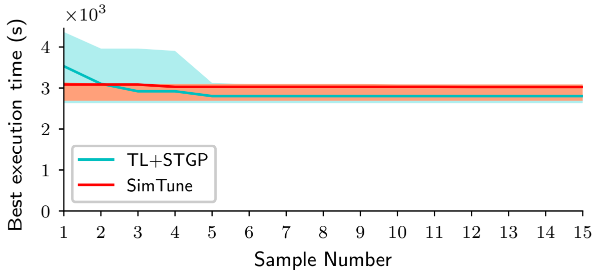
To evaluate the amount of cost saving, we estimate the search cost based on AWS [6] per-second pricing. SimTune saves the median tuning cost by 3.3X and 4.7X compared to Tuneful and RandomSearch.



(a) Bayes-DS2



(b) Bayes-DS3



(c) Bayes-DS4

Fig. 7: Convergence speed of transfer learning and STGP against SimTune. The shaded area represent the space between the 90th and 10th percentile of execution time.

Finally, the algorithm overhead is negligible: a few seconds to select the most similar workload, transfer the significant parameters and pick the next configuration during tuning. This is due to the small number of samples that we use during the MTGP optimization.

**Single tasked versus multitasked tuning:** In order to evaluate the benefits of MTGP over STGP, we experiment with TL+STGP for three evolving input sizes of the Bayes workloads. This is to show the benefits of sharing the tuning experience (execution samples tried during tuning) of a similar workload in limiting the exploration time. By leveraging similarity-aware MTGP optimization, as shown in Figure 7, at the 90th percentile TL+STGP tries configurations that are more costly with a wider search interval. SimTune finds configuration close to the ones found by TL+STGP (within 10%) while not only avoiding the cold start problem of TL+STGP but also bounding the search interval overall. It is worth noting that the closer the source workload is from the target workload (e.g., Bayes-DS2), the smaller the inter-percentile range of SimTune and the faster the convergence to near-optimal configuration. This suggests that having more seen workloads in the auxiliary set has the potential to save more search time.

**Our similarity-aware tuning accelerates the amortization of tuning costs:** the previous experiments do not show the full story on how the different approaches compare in behaviour as they perform incremental tuning from one execution to the next. For that, it is useful to have a timeline view. Figure 8 shows the cumulative execution time of running Bayes-DS3 workload over multiple configurations, as determined iteratively by the tuning algorithms considered. Here, we compare against using the directly transferred configuration from the source workload, since it represents the static tuning approach followed by existing tuning approaches, in which the configuration is tuned once and reused- ignoring the need for configuration retuning. The dotted line shows cumulative execution time for this configuration *without any tuning*. The dynamic tuning "pays off" only after the lines intersect the dotted line,

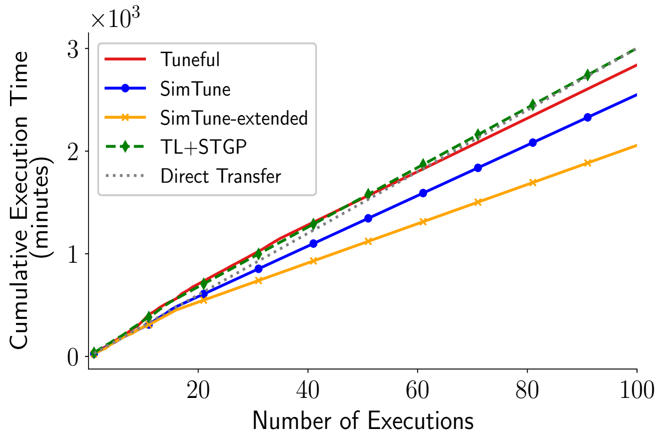


Fig. 8: Amortization speed of the different tuning approaches (Bayes-DS3 workload).

even if good configurations were found much earlier (finding them required exploring some worse configurations). Better configurations are shown as lines with shallower slopes (e.g. SimTune-extended, while equivalent configurations appear as parallel lines (e.g. SimTune and Tuneful). Our proposed tuning approach runs a fingerprinting execution sample once and tunes the configurations incrementally within 15 executions, then picks the best configuration it found and continues only using that.

As shown in Figure 8, it takes less than 5 executions to amortize the tuning cost using SimTune. This enables a quicker adoption of the need for workload retuning in a rapidly changing environment such as the big data environment. On the other hand, other approaches have higher tuning cost and need higher number of executions to amortize this cost (e.g. 60 executions).

This suggests that cloud providers, if able to observe executions similarity across clients, would be in the ideal position to offer tuned-configuration-as-a-service in a way that minimizes costs, even for workloads that are repeated just a couple of times.

## V. Related Work

Some work has been proposed to characterize workloads, aiming to either detect similarity across workloads [11], come up with representative benchmarks of workloads [25], or reveal existing performance issues [14]. Recently, a few work has been proposed to characterize and cluster Hadoop and Spark analytics workloads, with the majority of work employing PCA to identify the most important metrics to lower the dimensionality of the captured execution metrics [25]. Besides finding the important execution metrics, some work has been proposed to cluster the workloads using KMeans, grouping the various workloads based on their execution patterns [11], [25]. Jia et al. [24] characterize data analytics workloads (most of them are Hadoop based workloads) in data centers based on their microarchitectural characteristics (e.g., L1 instruction cache misses per thousand instructions). They concluded that data analytics workloads have inherent different characteristics from traditional workloads (e.g., HPC workloads). Similarly, some work has been proposed characterizing in-memory analytics workloads [13], [26]. Jiang et al. [26] characterize Spark workloads and compare them against Hadoop and traditional HPC workloads. Their work shed light on the significant differences in Spark in terms of memory utilization, memory access and disk I/O frequency. Awan et al. [13] highlight thread scalability issues in Spark (i.e., Spark does not scale linearly when more than twelve threads are running). Chiba et al. [14] characterize the memory, network, JVM, and GC usage to tune the performance TPC-H workloads on Spark.

On the configuration tuning front, several solutions have been proposed for tuning the configurations of Spark and Hadoop workloads. These solutions are either search-based leveraging techniques such as hill climbing [29] and genetic algorithm [30], or model-based, wherein a performance prediction model is built to guide the tuning using SVM [28], regression trees [39], or hierarchical models [41]. Some configuration tuning systems have leveraged similarity to accelerate the tuning process: AROMA [28] clusters the Hadoop workloads then builds a performance model that guides the tuning of each workload cluster. Scout [21] detects workload similarity to explore the search space more effectively for cloud configuration tuning. Lastly, OtterTune [37] utilizes similarity to guide the tuning of similar workloads in DBMS.

None of the existing work studied the need for configuration retuning in a dynamic rapidly changing environment and how to address that efficiently in the high-dimensional configuration space of in-memory data analytics such as Spark workloads. We differ from this earlier work since we go a step further than characterizing the big data analytics workloads or tuning the configurations. Our work focuses on *how* to utilize similarity to accelerate the configuration tuning/retuning of in-memory data analytics workloads. SimTune shares a *wider* tuning knowledge across similar workloads and compares against existing similarity-aware solutions that are either based on static tuning (direct transfer) or transfer learning (TL+STGP).

## VI. Conclusion

We presented SimTune, an approach for similarity-aware configuration tuning through leveraging workload characterization and MTBO. We showed that SimTune significantly reduces the exploration cost and accelerates the amortization of tuning costs, while finding configurations that are comparable the ones from the-state-of-the-art. We illustrated how SimTune works both when limited tuning knowledge is available and after extending the tuning knowledge, enabling the configuration tuning to happen faster while finding configuration closer to the optimal.

## VII. Acknowledgement

We thank Google and Amazon for generously supporting us with Google Cloud and AWS research credits to experiment this work.

## References

[1] TPC-H SQL benchmark, 2014. http://www.tpc.org/tpch/.
[2] Apache Spark: fast and general engine for large-scale data processing, 2015. https://spark.apache.org/.
[3] Growth forecast for the worldwide big data and business analytics market through 2020, 2015. https://www.idc.com/getdoc.jsp?containerId=prUS41826116.
[4] Apache Flink, 2016. http://flink.apache.org/.
[5] Apache Hadoop, 2016. http://hadoop.apache.org/.
[6] Amazon EC2 instance Pricing, 2018. https://aws.amazon.com/ec2/pricing/on-demand/.
[7] Hadoop distributed file system, 2018. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
[8] SimTune: configuration parameters under tuning, 2019. https://drive.google.com/file/d/1uHYGQjriw4IQyYTmOQukkWBYtIDD-4Tc/view?usp=sharing.
[9] SimTune source code and dataset, 2020. https://github.com/ayat-khairy/simtune.
[10] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.

[11] Sonali Aggarwal, Shashank Phadke, and Milind Bhandarkar. Characterization of hadoop jobs using unsupervised learning. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 748–753. IEEE, 2010.

[12] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *NSDI*, volume 2, pages 4–2, 2017.

[13] Ahsan Javed Awan, Mats Brorsson, Vladimir Vlassov, and Eduard Ayguade. Performance characterization of in-memory data analytics on a modern cloud server. In *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, pages 1–8. IEEE, 2015.

[14] Tatsuhiro Chiba and Tamiya Onodera. Workload characterization and optimization of TPC-H queries on Apache Spark. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 112–121. IEEE, 2016.

[15] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, 2015.

[16] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment*, 2(1):1246–1257, 2009.

[17] Ayat Fekry, Lucian Carata, Thomas Pasquier, Andrew Rice, and Andy Hopper. Tuneful: An online significance-aware configuration tuner for big data analytics, 2020. https://arxiv.org/pdf/2001.08002.pdf.

[18] Runxin Guo, Yi Zhao, Quan Zou, Xiaodong Fang, and Shaoliang Peng. Bioinformatics applications on Apache Spark. *GigaScience*, 7(8):giy098, 2018.

[19] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. Starfish: A self-tuning system for big data analytics. In *CIDR*, volume 11, pages 261–272, 2011.

[20] Matthew Hoffman, Bobak Shahriari, and Nando Freitas. On correlation and budget constraints in model-based bandit optimization with application to automatic machine learning. In *Artificial Intelligence and Statistics*, pages 365–374, 2014.

[21] Chin-Jung Hsu, Vivek Nair, Tim Menzies, and Vincent W Freeh. Scout: An experienced guide to find the best cloud configuration. *arXiv preprint arXiv:1803.01296*, 2018.

[22] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 41–51. IEEE, 2010.

[23] Kevin Jacobs and Kacper Surdy. Apache Flink: Distributed Stream Data Processing. Technical report, 2016.

[24] Zhen Jia, Lei Wang, Jianfeng Zhan, Lixin Zhang, and Chunjie Luo. Characterizing data analysis workloads in data centers. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*, pages 66–76. IEEE, 2013.

[25] Zhen Jia, Jianfeng Zhan, Lei Wang, Rui Han, Sally A McKee, Qiang Yang, Chunjie Luo, and Jingwei Li. Characterizing and subsetting big data workloads. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pages 191–201. IEEE, 2014.

[26] Tao Jiang, Qianlong Zhang, Rui Hou, Lin Chai, Sally A Mckee, Zhen Jia, and Ninghui Sun. Understanding the behavior of in-memory computing workloads. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pages 22–30. IEEE, 2014.

[27] SPT Krishnan and Jose L Ugia Gonzalez. Google compute engine. In *Building Your Next Big Thing with Google Cloud Platform*, pages 53–81. Springer, 2015.

[28] Palden Lama and Xiaobo Zhou. Aroma: Automated resource allocation and configuration of MapReduce environment in the cloud. In *Proceedings of the 9th international conference on Autonomic computing*, pages 63–72. ACM, 2012.

[29] Min Li, Liangzhao Zeng, Shicong Meng, Jian Tan, Li Zhang, Ali R Butt, and Nicholas Fuller. Mronline: Mapreduce online performance tuning. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 165–176. ACM, 2014.

[30] Guangdeng Liao, Kushal Datta, and Theodore L Willke. Gunther: Search-based auto-tuning of MapReduce. In *European Conference on Parallel Processing*, pages 406–419. Springer, 2013.

[31] Jonas Mockus. *Bayesian approach to global optimization: theory and applications*, volume 37. Springer Science & Business Media, 2012.

[32] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[33] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[34] Ilya M Sobol. On quasi-monte carlo integrations. *Mathematics and computers in simulation*, 47(2-5):103–112, 1998.

[35] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[36] Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. In *Advances in neural information processing systems*, pages 2004–2012, 2013.

[37] 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*, pages 1009–1024. ACM, 2017.

[38] Julien Villemonteix, Emmanuel Vazquez, and Eric Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509, 2009.

[39] Guolu Wang, Jungang Xu, and Ben He. A novel method for tuning configuration parameters of Spark based on machine learning. In *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on*, pages 586–593. IEEE, 2016.

[40] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, Burton Smith, and Randy H Katz. Selecting the best VM across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 452–465. ACM, 2017.

[41] Zhibin Yu, Zhendong Bei, and Xuehai Qian. Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 564–577. ACM, 2018.

[42] Zhao Zhang, Kyle Barbary, Frank Austin Nothaft, Evan Sparks, Oliver Zahn, Michael J Franklin, David A Patterson, and Saul Perlmutter. Scientific computing meets big data technology: An astronomy use case. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 918–927. IEEE, 2015.

[43] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 338–350. ACM, 2017.