Profiling-Based Big Data Workflow Optimization in a Cross-layer Coupled Design Framework

Qianwen Ye[1], Chase Q. Wu[1(✉)], Wuji Liu[1], Aiqin Hou[2], and Wei Shen[3]

[1] Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA
{qy57,chase.wu,wl87}@njit.edu

[2] School of Information Science and Technology, Northwest University, Xi'an 710127, Shaanxi, China
houaiqin@nwu.edu.cn

[3] School of Informatics Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, Zhejiang, China
shenwei@zstu.edu.cn

Abstract. Big data processing and analysis increasingly rely on workflow technologies for knowledge discovery and scientific innovation. The execution of big data workflows is now commonly supported on reliable and scalable data storage and computing platforms such as Hadoop. There are a variety of factors affecting workflow performance across multiple layers of big data systems, including the inherent properties (such as scale and topology) of the workflow, the parallel computing engine it runs on, the resource manager that orchestrates distributed resources, the file system that stores data, as well as the parameter setting of each layer. Optimizing workflow performance is challenging because the compound effects of the aforementioned layers are complex and opaque to end users. Generally, tuning their parameters requires an in-depth understanding of big data systems, and the default settings do not always yield optimal performance. We propose a profiling-based cross-layer coupled design framework to determine the best parameter setting for each layer in the entire technology stack to optimize workflow performance. To tackle the large parameter space, we reduce the number of experiments needed for profiling with two approaches: i) identify a subset of critical parameters with the most significant influence through feature selection; and ii) minimize the search process within the value range of each critical parameter using stochastic approximation. Experimental results show that the proposed optimization framework provides the most suitable parameter settings for a given workflow to achieve the best performance. This profiling-based method could be used by end users and service providers to configure and execute large-scale workflows in complex big data systems.

Keywords: Big data workflows · performance optimization · workflow profiling · stochastic approximation · coupled design

M. Qiu (Ed.): ICA3PP 2020, LNCS 12454, pp. 197–217, 2020.
https://doi.org/10.1007/978-3-030-60248-2_14

1 Introduction

Many large-scale applications in science, industry, and business domains are generating colossal amounts of data on a daily basis, now commonly termed as "big data". The processing and analysis of such data increasingly rely on workflow technologies for knowledge discovery and scientific innovation.

Big data workflows typically consist of several computing modules[1] with intricate dependencies, each of which could be as simple as a serial program as in most traditional applications or as complex as a parallel job in big data systems. The execution of such workflows goes far beyond the capability and capacity of single computers and parallel computing is widely recognized as a viable solution to support data- and network-intensive workflows.

In recent years, significant progress has been made in almost every aspect of the hardware and software for big data computing, including the hardware upgrade of computers, the bandwidth improvement of network infrastructure, and the emergence of reliable and scalable data storage and analysis platforms such as Hadoop MapReduce/Spark that can support parallel processing of colossal amounts of data. For example, in China, the most popular search engine of Baidu, the e-commerce transaction service of Alibaba, and the social network platform of Tencent, all depend on Spark-based solutions at scale. Particularly, Tencent has eight hundred million active users generating over 1 PB of data per day, processed on a cluster consisting of more than 8,000 computing nodes [1].

However, even with these cutting-edge technologies, we have not seen the corresponding performance improvement, because workflow performance largely depends on how big data systems are configured and used. For example, workflow makespan or end-to-end delay, which is the most commonly concerned performance metric, is affected by multiple layers of big data systems, including the inherent properties (scale and topology) of the workflow, the parallel computing engine it runs on, the resource manager that orchestrates distributed resources, the file system that stores data, as well as the parameter setting of each layer.

Optimizing workflow performance is challenging because the compound effects of the aforementioned layers are complex and opaque to end users. Generally, tuning their parameters requires an in-depth understanding of big data systems, and the default settings do not always yield optimal performance. Due to the dynamics of workflow execution in distributed environments and the lack of accurate performance models for parallel computing, it is generally very difficult to analytically derive the best system configuration for a given workflow. Oftentimes, even production systems may run with misconfigured parameters and present performance bottleneck, which significantly limits the productivity of end users and the utilization of expensive computing resources. To end users, who are primarily domain experts, it is a daunting task to troubleshoot the performance issue and identify possible system misconfiguration.

[1] A module is a processing unit, executed in serial or parallel, in a workflow, and is also referred to as a job or subtask in some context.

In this paper, we propose a profiling-based cross-layer coupled design framework to determine the most suitable configuration with a recommended parameter settings for each layer, including workflow structure, computing engine, and cluster infrastructure to optimize workflow performance. Such a profiling-based framework requires the exploration of a large parameter space across the entire technology stack of big data systems by running a large set of workflow experiments with different parameter settings. However, sweeping through the entire parameter space is practically infeasible, due to the large number of tunable parameters and their corresponding seemingly infinite value ranges. Thus, exhaustive profiling is generally unacceptable, especially when the overhead of profiling is much higher than the time needed for actual workflow execution.

Therefore, we design and employ two approaches to reduce the number of experiments needed in our framework to accelerate the exploration of the parameter space for optimal performance: i) identify a subset of critical parameters with the most significant influence on workflow performance through feature selection; and ii) minimize the search process within the value range of each critical parameter using stochastic approximation. Experimental results show that the proposed optimization framework provides the most suitable parameter settings for a given workflow to achieve the best performance. This profiling-based method could be used by end users and service providers to configure and execute large-scale workflows in complex big data systems.

In sum, our work makes the following contributions to the field of big data computing with a focus on workflow execution:

- We design a profiling-based workflow optimization framework, which considers multiple layers in the entire technology stack of Hadoop MapReduce/Spark and provides end users with configuration recommendations to optimize workflow performance in terms of makespan.
- We employ feature selection to determine critical parameters with the most significant influence on workflow performance and apply stochastic approximation to minimize the time needed to explore the parameter space.
- We implement and test the proposed method with extensive experiments, which show that it yields satisfactory performance while significantly reducing profiling overhead compared with traditional methods.
- The proposed workflow optimization framework and the profiling minimization approaches are generic and hence applicable to other big data systems to systematically determine the most suitable system configuration.

The rest of this paper is organized as follows. In Sect. 2, we survey related work. In Sect. 3, we present a coupled design framework for cross-lay optimization of big data workflows. In Sect. 4, we investigate the effects of different parameters and propose a profiling-based approach to parameter setting. In Sect. 5, we discuss the rationale of using the Simultaneous Perturbation Stochastic Approximation algorithm and provide its convergence in our problem. In Sect. 6, we present the overall system configuration recommendation framework. We conduct experiments for performance evaluation in Sect. 7.

2 Related Work

Application-layer configuration for performance improvement of scientific workflows has been investigated in various contexts. Grid-based workflow management systems such as Taverna [2], Kepler [3] and Pegasus [4] seek to minimize workflow makespan by adjusting workflow-level parameters to group and map workflow components. Kuma *et al.* proposed an integrated framework, which is capable of supporting performance optimization along multiple dimensions of the parameter space [5]. In [6], in order to support domain-specific parameter-based optimization, end users are required to provide performance and quality models of expected application behaviors to the system. This approach is not readily applicable to many situations, where the relationships between parameters and performances cannot be directly modeled with an analytical form. In [7], Holl *et al.* proposed an optimization phase between the planning phase and the execution phase of the common scientific workflow life cycle to assist end users in determining the optimal setup of a specific workflow. This phase indeed helps end users configure workflows properly, however, in a try-and-error manner, which could be very time-consuming. In the aforementioned work, only parameters at the workflow layer are considered, while neglecting the influence of the configuration of other layers in the execution system.

There also exist a number of efforts on parameter setting in the layer of computing engine for parallel processing. In [8], Gounaris *et al.* investigated how parameters in Spark affects the performance of Spark jobs and proposed a systematic method for parameter tuning.In addition, many machine learning-based approaches have been proposed to tune Spark and Hadoop parameters. In [9], a binary classification and multi-classification machine learning model is proposed to accelerate the tuning of Spark parameters. In [10], Liao *et al.* proposed a machine learning model to automatically tune Hadoop MapReduce configuration parameters. In [11], Wu *et al.* proposed a profiling and performance analysis-based framework for Hadoop configuration, which considers parameters not only in the MapReduce layer but also in the Hadoop Distributed File System (HDFS) layer. In [12], Li *et al.* proposed an online performance tuning system that monitors a job's execution, tunes its associated performance-related parameters based on historical statistical data, and provides fine-grained control over parameter configuration.

Different from the existing work, we consider a generic end-to-end delay minimization problem for big data workflows consisting of multiple MapReduce/Spark jobs, investigate the cross-layer execution stack, analyze the compound effects of parameters in all layers, and propose a coupled design framework that determines and recommends the most suitable parameter setting.

3 A Coupled Design Framework for Cross-layer Optimization of Big Data Workflows

Large-scale workflows executed in big data systems such as Hadoop are complex processes that involve distributed storage and parallel computing.

Figure 1 illustrates a technology stack of Hadoop with multiple layers, each of which has a non-negligible impact on workflow performance, as detailed below:

- The top layer defines abstract big data workflows comprised of data-intensive computing jobs with execution dependencies. Such workflows are typically modeled as a Directed Acyclic Graph (DAG), where each vertex represents either a MapReduce or a Spark job, and each directed edge represents the execution dependency and data flow between two adjacent jobs.
- The second layer is the parallel computing engine, which orchestrates data processing by marshaling distributed nodes, running multiple tasks in parallel, managing all communications and data transfers between various parts of the system, and meanwhile providing redundancy and fault tolerance. Here, we consider MapReduce and Spark as they are the two most widely used parallel computing engines in Hadoop system.
- The third layer is the resource manager of the cluster, which is responsible for keeping track of the resource use on each data node, allocating distributed system resources to various jobs or applications submitted by different users, and scheduling tasks for execution on different nodes. We focus on the Yet Another Resource Negotiator (YARN), which was first introduced in Hadoop 2 to improve MapReduce implementation, and has been generalized to support different computing engines including Spark and Storm.
- The fourth layer is the distributed file system, which lays down the foundation for big data processing, and is responsible for managing data storage across multiple nodes and making it tolerant to node failures without suffering data losses. In this paper, we consider Hadoop Distributed File System (HDFS) as it is commonly used for distributed storage of big data in Hadoop.

Correspondingly, we list in Table 1 all performance-related parameters in these technology layers, namely, MapReduce, Spark, YARN, and HDFS. Note that any of these parameters could create a bottleneck and hence limit workflow performance. All of these parameters can be controlled in the system configuration, such as the degree of parallelism, the amount of computing resources allocated to each task, and the job scheduling policy.

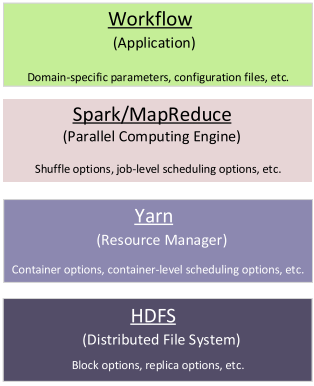


Fig. 1. A coupled design framework for big data workflow optimization.

In general, parameter tuning helps improve the performance of workflow execution in big data systems, but it usually requires considerable familiarity with the system and in-depth domain knowledge in parallel and distributed computing, which, unfortunately, many end users lack. Researchers have achieved remarkable results through parameter tuning in various layers, including job scheduling scheme [13–16], Spark and MapReduce parameter tuning [17–22], and resource manager configuration [21,23]. However, the parameter setting in one layer may have an impact on the parameter setting in other layers,

which has not been largely explored. The proposed coupled design framework is to account for the coupled effects of different parameters across multiple layers in the technology stack to achieve optimal workflow performance.

Table 1. Performance-related parameters in MapReduce/Spark, YARN, and HDFS layers.

Layers	Parameters
MapReduce	The number of reduce tasks per job
	The fraction of the number of maps in the job, which should be completed before reduces are scheduled for the job
	Should the outputs of the maps be compressed before being sent across the network?
Spark	The number of executors for one Spark application
	The amount of memory to use per executor process
	The number of cores to use on each executor
	The number of partitions in RDDs returned by transformations such as join and reduceByKey, and parallelize when not set by user
	The amount of storage memory immune to eviction
	Whether to compress map output files?
	How long to wait to launch a data-local task before giving up and launching it on a less-local node?
	The scheduling mode between jobs submitted to the same SparkContext, which can be set to FAIR or FIFO
	The number of cores to allocate for each task
YARN	The type of resource scheduler
	The amount of physical memory, in MB, that can be allocated for containers
	The number of vcores that can be allocated for containers
	The maximum number of applications in the system, which can be concurrently active both running and pending
	The maximum percentage of resources in the cluster, which can be used to run application masters (it controls the number of concurrent active applications)
HDFS	The number of block replications
	The Sblock size for new files, in bytes

4 Parameter Effects and Workflow Profiling

4.1 An Empirical Study of Parameter Effects

In the workflow layer, the inherent properties of the workflow such as input data size, workflow topology, and implementation language would affect workflow

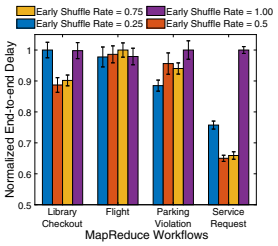
performance. In addition, the performance of workflow execution is also affected by the parameter settings in various layers.

To examine such parameter effects, we conduct a set of exploratory experiments. For illustration, we collect and present the experimental results of workflow execution on a local cluster consisting of eight virtual machine instances to show how different early shuffle rates affect workflow performance. We plot in Fig. 2(a) the workflow end-to-end delay measurements with different early shuffle rates using four MapReduce workflows, including 1) library checkout data analysis from New York City Library (labeled as library checkout), 2) flight data analysis (labeled as flight), 3) parking violation analysis in New York City (labeled as parking violation), and 4) 311 service request analysis in New York City (labeled as service request), all of which are based on online public data repositories [24–27]. If the shuffling process starts earlier, at each time point, there is less data on the fly, and data transfer is less likely to become the bottleneck of workflow execution. However, if reduce tasks are scheduled earlier, then the computing resources allocated to reduce tasks are reserved and may lead to resource waste and consequently undermine the workflow execution performance. From these performance measurements, we observe that this parameter affects the workflow performance of flight data analysis the least and exhibits different performance patterns for parking violation analysis, service request analysis, and library checkout analysis.

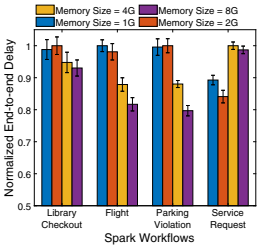
To illustrate the effects of different parameters in Spark on job execution performance, we compare the execution time of these four different workflows with three different executor memory sizes, as shown in Fig. 2(b). Theoretically, a larger memory size allocated to each executor would lead to a shorter execution time for a given Spark job. We observe in Fig. 2(b) that: i) for library checkout, flight data analysis, and parking violation analysis, it is clear that increasing memory improves workflow performance; ii) for 311 service request analysis, which has many independent modules, there is an intensive resource competition, which leads to performance degradation as the memory size increases.

We compare the end-to-end delay of these four workflows in Fig. 2(c) with different node locality delays, which define the number of missed scheduling opportunities, after which the scheduler attempts to schedule rack-local containers. On a busy cluster, when an application requests a particular node where its input data are stored, there is a good chance that other containers are running on it at the time of the request. The obvious measure is to immediately loosen the locality requirement and allocate a container on the same rack. However, waiting a short time before scheduling to another node may significantly increase the chance of getting a container on the requested node, and therefore increase the efficiency of the cluster. On the other hand, waiting too long may exceed the time saved by avoiding data transfer. In Fig. 2(c), we observe that these workflows exhibit different performance patterns in response to this parameter.

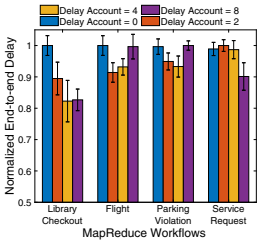
Similar to native file systems, HDFS processes data in blocks, but with a larger size (e.g., 128 MB in most systems). By default, each block is replicated three times to increase reliability. Therefore, when a file is written into HDFS, it is firstly split into multiple blocks according to the configured block size and each



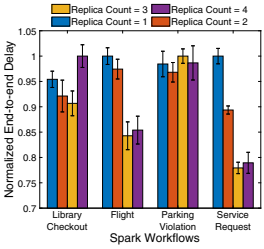
(a) MapReduce layer: comparison of end-to-end delay with different early shuffle rates



(b) Spark layer: comparison of execution time with different executor memory sizes



(c) YARN layer: comparison of end-to-end delay with different node locality delays



(d) HDFS layer: comparison of end-to-end delay with different replica numbers

Fig. 2. Illustration of parameter effects in different layers on the performance of four workflows.

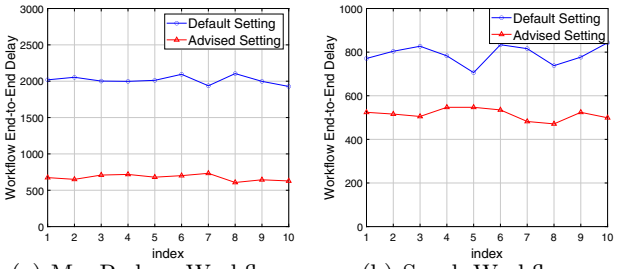
block would be replicated and distributed across the entire cluster. Figure 2(d) shows the impact of block replication number on the end-to-end delay of the aforementioned four workflows. Theoretically, if there are more replicas stored on the cluster, there would be a higher chance that each node has a local data block to process. In this case, With sufficient computing resources, the node does not have to retrieve data over the network to process, thereby reducing the end-to-end delay. On the other hand, with a larger replication number, the time for writing a file into HDFS would increase, hence negatively affecting workflow performance. Also, since the output of MapReduce and Spark jobs is written into HDFS automatically, the time for writing the output of each module in the workflow would increase, hence negatively affecting workflow performance. We observe that the impact of this parameter on the performance of these workflows is different. Particularly, the parameter effect for library checkout data analysis is not monotonous: the end-to-end delay decreases in the beginning and then increases as the replication number increases.

4.2 Workflow Performance Profiling

A workflow profile $WP_w(G(\theta), C, \theta)$ is a control-response plot illustrating how a set θ of system parameters affect the performance $G(\theta)$ of workflow w running

on cluster C. The workflow profile can be obtained by varying θ to exhaust the combination of parameter values on cluster C and measuring the corresponding average end-to-end delay T.

Effectiveness of Workflow Profiling. To illustrate the effectiveness of workflow profiling in improving workflow execution performance, we run the Spark and MapReduce workflows for 311 service request analysis for 10 times, each with the default setting and a customized setting, respectively, and plot the performance measurements in Fig. 3. These measurements show that the customized parameter setting significantly reduces the workflow end-to-end delay in comparison with the default setting. We observe in Fig. 3 that the customized setting makes more improvement over the default setting on MapReduce workflows. This is because, for a MapReduce job, the degree of parallelism in the reduce phase is 1 by default, which hurts the performance. On the other hand, Spark distributes the job workload to all executors, which results in a higher degree of parallelism.



(a) MapReduce Workflow (b) Spark Workflow

Fig. 3. Performance comparison of 311 service request analysis workflow with the default setting and a customized setting.

Overhead of Exhaustive Search-Based Workflow Profiling. The goal of workflow performance profiling is to determine the control parameter values θ^*, at which the workflow end-to-end delay T reaches the global minimum. An exhaustive search-based approach for workflow profiling is prohibitively time-consuming. Take a MapReduce workflow as an example, where we consider only one control parameter in each layer, including the number of reduce tasks ($m \in \{1, 2, ..., M\}$) in MapReduce, memory size constraint ($n \in \{1, 2, ..., N\}$) in Gigabytes for containers in YARN, and block size ($b \in \{32, 64, ..., 2^k\}$ MBytes) in HDFS. It would take a total time of $O(k \cdot N \cdot M \cdot \Delta T)$, where ΔT is the time taken by a single workflow profiling test, and is typically on the order of minutes or even hours. In the experiment of flight data analysis based on MapReduce, even if we only vary the reduce task count from 1 to 5, the memory size constraint from 1 GB to 8 GB, and the block size from 32 MB to 512 MB, a single profiling test takes around 9 min and the exhaustive search takes 1,125 min (around 19 h)

in total to complete. Such a time overhead may be comparable with or even longer than the total time required by big data workflow execution, which is typically on the order of hours, days, or weeks. As the number of control parameters increases, the time to obtain a complete workflow profile rapidly increases, making the exhaustive search approach practically infeasible.

5 SPSA-Based Profiling Optimization

To obviate the need for conducting exhaustive profiling, we propose a Simultaneous Perturbation Stochastic Approximation (SPSA) [28] based profiling method to quickly determine the most suitable parameter setting in each layer.

5.1 Rationale on the Use of SPSA

As shown in Table 1, many components and factors are involved in the process of workflow execution and may affect workflow performance observed by the end user. Since our work is focused on workflow profiling with respect to end-to-end delay, the execution dynamics could be treated as a "black box" system, where the input is the set of control parameters θ and the output is the workflow end-to-end delay $G(\theta)$. Based on this model, stochastic approximation algorithms are suitable for quickly determining the optimal control parameter values, because they i) do not require an explicit formula of $G(\theta)$, which is essentially unknown, but only its measurement with random noise $G(\theta) + \xi$, which can be obtained by running a "one-time profiling" with a set of specific values; ii) do not require any additional information about system dynamics or input distribution; and iii) are of low time complexity. These are highly desirable features as they not only account for the dynamics of workflow execution and the randomness in performance measurements, but also significantly reduce the computational overhead compared with exhaustive profiling.

SPSA algorithms are similar to gradient descent methods to find a global minimum. In comparison with other methods, it has distinctive desirable features: i) the gradient approximation of SPSA requires only two measurements of the objective function, regardless of the dimension of the control parameters; ii) SPSA produces instant results without requiring a large amount of historical data; iii) SPSA has a proven convergence property, and thus its performance is theoretically guaranteed.

5.2 Stochastic Approximation (SA) Methods

In our profiling problem, the average workflow end-to-end delay is denoted as a function of $G(\theta)$ of control parameters θ. The goal is then to find the most suitable control parameters value θ^* that minimize $G(\theta)$ within the feasible space Θ. Following the standard Kiefer-Wolfowitz Stochastic Algorithm (KWSA) [29], the standard stochastic approximation form is given by,

$$\hat{\theta}_{k+1} = \hat{\theta}_k - a_k \cdot \hat{g}_k(\hat{\theta}_k), \tag{1}$$

where $\hat{\theta}_k$ is the set of control parameter values in the k-th iteration, $\hat{g}_k(\hat{\theta}_k)$ is the simultaneous perturbation estimate of $G(\theta)$'s gradient $g(\theta) = \frac{\partial G}{\partial \theta}$ at the iteration $\hat{\theta}_k$ based on the measurements of $G(\theta)$, and a_k is a nonnegative scalar gain coefficient.

The measurement of workflow end-to-end delay with noise, denoted as $y(\theta)$, is available at any value $\theta \in \Theta$, and given by $y(\theta) = G(\theta) + \xi$, where ξ is the noise incurred by system dynamics during workflow execution. In fact, $y(\theta)$ is the observed average workflow end-to-end delay of a single profiling test with a specific θ.

The essential part of Eq. 1 is the gradient approximation $\hat{g}_k(\hat{\theta}_k)$. With simultaneous perturbation, all elements of $\hat{\theta}_k$ are randomly perturbed at the same time to obtain two loss measurements of $y(\cdot)$. For the two-sided SP gradient approximation, this leads to

$$\hat{g}_k(\hat{\theta}_k) = \frac{y(\hat{\theta}_k + c_k\Delta_k) - y(\hat{\theta}_k - c_k)}{2c_k\Delta_k} \begin{bmatrix} \Delta_{k1}^{-1} \\ \Delta_{k2}^{-1} \\ \vdots \\ \Delta_{kp}^{-1} \end{bmatrix}, \tag{2}$$

where p is the number of control parameters under consideration, the mean-zero p-dimensional random perturbation vector, $\Delta_k = [\Delta_{k1}^{-1}, \Delta_{k2}^{-1}, \cdots, \Delta_{kp}^{-1}]^T$, is independent and symmetrically distributed around 0 with finite inverse $E\left|\Delta_{ki}^{-1}\right|, (i = 1, 2, \cdots, p)$, and c_k is a positive coefficient. Because the numerator is the same in all p components of $\hat{g}_k(\hat{\theta}_k)$, the number of loss measurements needed to estimate the gradient in SPSA is two, regardless of the dimension of p.

5.3 Convergence of SPSA-Based Workflow Profiling

The convergence of the proposed SPSA-based workflow profiling method is important as it affects both the quality of workflow profiling results and the efficiency of parameter setting recommendation. To explore the applicability of SPSA in the profiling optimization problem and investigate its convergence property, we validate the conditions that lead to the convergence in the context of workflow profiling.

As pointed out by Spall in [30] (pp. 161), the conditions for convergence can hardly be all checked and verified in practice due to the lack of knowledge on $G(\theta)$. We provide the following arguments to justify the appropriateness of SPSA in solving the profiling optimization problem.

According to Theorem 7.1 in [30] (pp. 186), if Conditions $B.1'' - B.6''$ hold and θ^* is a unique minimum of $G(\theta)$, then for SPSA, θ_k almost surely converges to θ^* as $k \to \infty$.

Conditions $B.1''$, $B.4''$, and $B.6''$ are the most relevant since we govern the gains sequence a_k and c_k and the random perturbation Δ_k. The coefficient sequences a_k and c_k we choose (See Sect. 6.2) and the symmetric Bernoulli ± 1

distribution we follow to generate the simultaneous perturbations $\{\Delta_{ki}\}$ easily validate Conditions $B.1''$. We generate the simultaneous perturbations $\{\Delta_{ki}\}$ to ensure that Δ_{ki} is a mutually independent sequence, which is independent of $\hat{\theta}_1, \hat{\theta}_2, \cdots, \hat{\theta}_k$. The noise in workflow execution is mainly caused by the dynamics and randomness of computer systems and network devices. Since the loss measurement $y(\cdot)$ captures both positive noise and negative noise, the long-term conditional expectation of the observed noise is considered to be zero, i.e., for all k, $E[\varepsilon_k^+ - \varepsilon_k^- | \{\hat{\theta}_1, \hat{\theta}_2, \cdots, \hat{\theta}_k\}, \Delta_k] = 0$. Since $\{\Delta_{ki}\}$ is generated following the symmetric Bernoulli ± 1 distribution with a probability of 0.5 for each outcome of either $+1$ or -1, $E[\Delta_{ki}^{-1}]$ is uniformly bounded. In addition, the loss measurements $y(\hat{\theta}_k \pm c_k \Delta_k)$ are bounded by the full computing/networking capacity of a given cluster, so the ratio of measurement to perturbation $E[y(\hat{\theta}_k \pm c_k \Delta_k)\Delta_{ki}]$ is uniformly bounded over i and k. Hence, Condition $B.4''$ holds.

Condition $B.2''$ and $B.3''$ impose the requirement that $\hat{\theta}_k$ (including the initial condition) is close enough to θ^* so that there is a natural tendency for an analogous deterministic algorithm to converge to θ^*. These two conditions are intuitively valid in the profiling scenario because: i) the main requirement for these conditions, i.e., $sup_{k \geq 0} \left\| \hat{\theta}_k \right\| < \infty$, can be satisfied since the control parameter values in all layers are both finite positive integer numbers; ii) since the feasible regions of the control parameters in workflow execution is finite and mapped to a limited range of the iterative variables, $\hat{\theta}_k$ (including the starting point) is sufficiently close to θ^*; iii) in workflow execution environments, due to the system dynamics and randomness, different runs with identical parameter settings may yield different end-to-end delays, which makes θ^* be not a single point but an "acceptable area" in order to tackle this profiling problem.

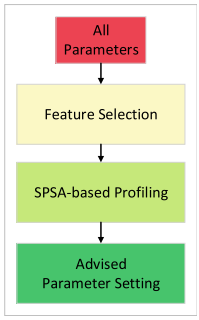


Fig. 4. Architecture of the proposed workflow configuration advisor.

$B.5''$ asks $G(\theta)$ to be three-times continuously differentiable and bounded by R^p. The end-to-end delay is obviously bounded by the cluster computing power and network capacity among all dimensions of the control parameter θ. However, the smoothness and differentiability of $G(\theta)$ is very difficult to verify since $G(\theta)$ is practically unknown and spans over a parameter space of discrete values. For this profiling problem, we assume $G(\theta)$ meets this condition.

In addition, it is worth pointing out that although the end-to-end delay should have a unique theoretical peak over the feasible control parameter space in a given distributed system, it has been observed that in our environments, due to the system dynamics and randomness, identical parameter values may yield different end-to-end delays in different runs, which makes the uniqueness of θ^* practically unverifiable. However, the loss measurement $y = G(\theta) + \xi$ indeed shows a trough property over the feasible control parameter space in the experiments.

6 A Workflow Configuration Advising Framework

We design a workflow configuration advising framework as shown in Fig. 4, which conducts workflow profiling across the entire technology stack of big data systems and provides end users with recommended configurations to deploy big data workflows for actual execution. This framework integrates two strategies to avoid the need of conducting exhaustive profiling and minimize the number of profiling runs while still ensuring the recommendation of a satisfactory parameter setting: i) it identifies a subset of critical parameters to be investigated; ii) it reduces the search space within the value range of each critical parameter.

6.1 Information-Theoretic Feature Selection

Not all parameters play an equally important role in workflow performance, and we, therefore, should focus on the parameters with the most significant influence. Towards this goal, we employ an information-theoretic feature selection method, the nearest neighbor method [31], to filter out a dominating subset of control parameters using historical dataset.

For the control parameter X, we take random samples x_i and repeat workflow execution for multiple times, collect the end-to-end delay measurements y_i and construct a historical dataset $D = \{x_i, y_i\}_{i=1}^{N}$, where N denotes the total number of data points collected in D. For a given control-response data point i, we first identify the k-th nearest neighbor of this point among data points with the same configuration x_i in terms of Euclidean distance, and denote the distance as d. Note that k is a customizable parameter. Then count the number m_i of nearest neighbors in D that lie within distance d to data point i (including the k-th nearest neighbor with the same configuration). The amount of information I_i data point i carries is calculated as

$$I_i = \phi(N) - \phi(N_{x_i}) + \phi(k) - \phi(m_i), \tag{3}$$

where N_{x_i} is the number of data points who has configuration x_i, ϕ is the digamma function [32]. The mutual information of X and Y is calculated by averaging I_i over all N data points in D:

$$I(X, Y) = \phi(N) - \langle \phi(N_x) \rangle + \phi(k) - \langle \phi(m) \rangle, \tag{4}$$

where $\langle\rangle$ is an averaging operator. Based on the estimated mutual information between X and Y, we decide to select X as a candidate critical parameter by thresholding the score of the mutual information.

6.2 SPSA-Based Profiling Process

The SPSA-based workflow profiling process is described as follows.

Step 1: Set counter index $k = 0$. Choose an either fixed or random starting point within the feasible space of the control parameters. Also, choose

nonnegative coefficients a, c, A, α, and γ in the SPSA gain sequences $a_k = \frac{a}{(K+1+A)^\alpha}$ and $c_k = \frac{c}{(k+1)^\gamma}$. Practically effective (and theoretically valid) values for α and γ are 0.602 and 0.101, respectively [33]. For the choice of a, A and c, please refer to Sect. 6.2.

Step 2: Generate by Monte Carlo a p-dimensional random perturbation vector Δ_k, where each of p components of Δ_k is independently generated from a zero-mean symmetric Bernoulli ± 1 distribution with probability of $\frac{1}{2}$ for each ± 1 outcome.

Step 3: Conduct workflow profiling to collect two performance observations $y_k^+ = y(\hat{\theta}_k + c_k\Delta_k)$ and $y_k^- = y(\hat{\theta}_k c_k \Delta_k)$ with c_k and Δ_k from Step 1 and Step 2.

Step 4: Compute the simultaneous perturbation approximation to the unknown gradient $g(\hat{\theta}_k)$ using Eq. 2.

Step 5: Use the standard stochastic approximation form (see Eq. 1) to update $\hat{\theta}_k$ to a new value $\hat{\theta}_{k+1}$.

Step 6: Return to Step 2 with $k+1$ replacing k, and repeat the above process until the termination condition is met (please refer to Sect. 6.2).

Termination Condition. Many efforts have been made to establish the termination condition for SA-based methods since the Kiefer-Wolfowitz SA algorithm was first proposed [29]. Our algorithm employs the following two practical rules to guarantee its termination with satisfactory performance.

- **Upper bound - the $\mathcal{M}$ rule:** our algorithm terminates the profiling process when the total number of profiling iterations exceeds a threshold $\mathcal{M}$, which is typically set to be much less than the maximum number of profiling tests of the exhaustive profiling approach, i.e., $\mathcal{M} \ll \prod_{i=1}^{k} NV_i$, where k is the number of control parameters and NV_i is the number of possible values of the k-th control parameter. Note that when $\mathcal{M} = \prod_{i=1}^{k} NV_i$, our algorithm rolls back to the exhaustive search.
- **Impeded progress - the $\mathcal{R}$ rule:** if the number of consecutive profiling iterations that do not produce any performance improvement compared with the best one observed so far exceeds an upper bound $\mathcal{R}$, our algorithm terminates the profiling process for the given workflow.

Restrictions on Parameters. Restrictions have to be placed on some parameters to prevent unnecessary profiling runs and reaching the upper limit of the cluster capacity. For the parameters in the discussion, we confine the CPU count and memory size for a YARN container to be smaller than the capacity of computing nodes. Similarly, we confine the block size of HDFS to be smaller than the storage capacity of computing nodes and to be larger than 32 MB as an excessively small block size would jeopardize workflow performance [34,35].

Choice of Gain Sequences. The coefficients a_k and c_k of the SPSA-based method need to satisfy the following conditions to guarantee the convergence:

$$a_k > 0, c_k > 0; a_k \to 0, c_k \to 0; \sum_{k=0}^{\infty} a_k = \infty, \sum_{k=0}^{\infty} \frac{a_k^2}{c_k^2} < \infty. \tag{5}$$

The choice of the gain sequence is critical to the performance. With the Bernoulli ± 1 distribution of Δ_k, α and γ are specified in Step 1 with values commonly adopted in practice. Step sizes a and c are empirically determined based on the size of the search space.

Control Parameter Calculation. The control parameters used in our SPSA-based profiling algorithm are denoted as $\theta = \left[s_1^{'}\ s_2^{'}\ ...\ s_p^{'}\right]^T$, and set to be positive values within a reasonably selected range to ensure a comparable magnitude for each parameter dimension. These parameters are scaled and mapped to decide the actual parameter values in each iteration. In calculating the actual values of the parameters, we perform rounding operations in the case where the intermediate results are fractional.

The profiling unit of each critical parameter is denoted by $\mu_i, i \in 1, 2, \cdots, p$. For the profiling unit, we calculate its corresponding actual value for performance observation (i.e., y_k^+ and y_k^-) as $s_i = round(\lambda_i(s_i^{'}) \cdot \mu_i)$, where $\lambda(\cdot)$ are a scaling function that scales all control parameters into the same magnitude. For example, with $\lambda_i(s_i^{'}) = 2^{s_i^{'}}$, $s_i^{'}$ increases exponentially as the original value of $s_i^{'}$ increases.

We apply different scaling functions to different control parameters as their actual values exhibit different patterns. For instance, we consider i) linear type, where the actual values of such parameters grow linearly, such as early shuffle rate in MapReduce, and replication count in HDFS; ii) exponential type, where the actual values of such parameters grow exponentially, such as block size in HDFS. For linear type, we apply a min-max normalization approach to implement the scaling function; while for exponential type, we apply both an exponential scaling function and a min-max normalization approach to perform scaling.

7 Performance Evaluation

In this section, we evaluate the performance of the proposed workflow optimization framework. We first conduct experiments to test our profiling algorithm, and then demonstrate the advising procedure and illustrate the performance benefits of our algorithm compared with random walk [36], tabu search [37] and linear regression [38] plus dual-simplex algorithm [39].

7.1 Experiment Setting and Workflow Description

In our experiment, we establish a homogeneous cluster of three racks, each of which has two computer nodes installed. On the cluster, we install Apache

Hadoop 2.9.5 [40], Apache Spark 2.4.4 [41] and Apache Oozie 4.3 [42], which is a workflow engine that automatically dispatches each component MapReduce/Spark job in a workflow once all of its preceding jobs finish execution.

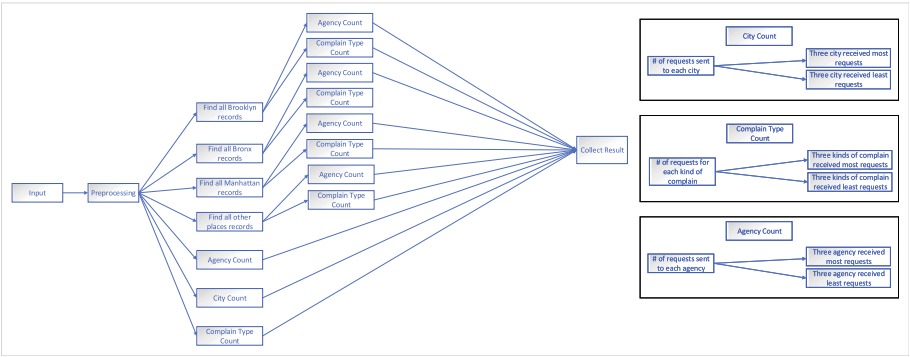


Fig. 5. The topology of the workflow for 311 service request data analysis.

We download four public datasets online: i) airline on-time performance dataset [24], ii) New York public library checkout records [25], iii) New York 311 service request record [27], and iv) New York parking violation record [26]. We implement MapReduce and Spark workflows with complex topologies for each dataset. For illustration, Fig. 5 shows a representative topology of the workflow for 311 service request data analysis.

Table 2. Critical parameters through feature selection.

Layers	Parameters
MapReduce	mapreduce.job.reduce.slowstart.completedmaps
	number of reducers
Spark	num-executors
	spark.executor.memory
	spark.executor.cores
Yarn	yarn.scheduler.capacity.node-locality-delay
	maximum-am-resource-percent
HDFS	dfs.replication
	dfs.blocksize

7.2 Experimental Results

We consider all parameters tabulated in Table 1, which do not play an equally important role in workflow performance. In fact, the combined setting of some critical parameters across different layers dominates the workflow execution time. Since exhaustive profiling by sweeping through the entire value range of all parameters is extremely time-consuming, we first use information-theoretic feature selection to identify a subset of critical parameters as shown in Table 2, all of which have a non-negligible influence on workflow performance.

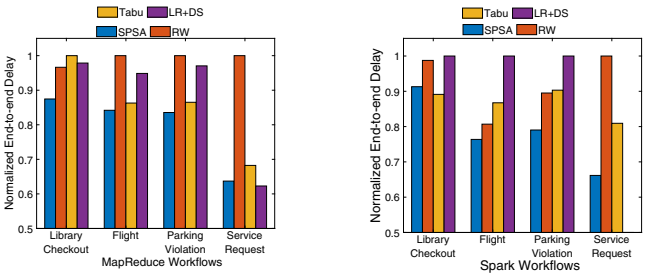


Fig. 6. Comparison of end-to-end delay between different settings produced by SPSA-based profiling (SPSA), random walk (RM), Tabu search (Tabu) and linear regression plus dual-simplex optimization algorithm (LR+DS).

We compare our SPSA-based profiling with Random Walk, Tabu Search, Linear Regression plus dual-simplex and default Hadoop/Spark in terms of workflow execution time (ED). The ED performance improvement of the SPSA-based profiling algorithm over other algorithms in comparison is defined as:

$$Imp(Other) = \frac{ED_{Other} - ED_{SPSA}}{ED_{Other}} \times 100\%, \tag{6}$$

where ED_{other} is the ED achieved by other algorithms, and ED_{SPSA} is the ED achieved by the SPSA-based profiling algorithm. The improvement result is tabulated in Table 3, where the upper side of the table is for MapReduce workflow (referred to as MR WFs in the table) and the lower side is for Spark workflow. Note that there is no result listed for linear regression plus dual-simplex for Spark service request workflow, because the proposed setting leads to the depletion of all cluster resources and the workflow execution is therefore suspended. Figure 6 shows the comparison of normalized end-to-end delay between these algorithms. These measurements show that our algorithm consistently outperforms the other algorithms in comparison.

Table 3. The percentage of ED improvement made by SPSA-based profiling over Default Setting (D), Random Walk (RW), Tabu Search (Tabu) and Linear Regression plus Dual-Simplex (LRDS).

MR/Spark Workflows	imp(D)	imp(RW)	imp(Tabu)	imp(LRDS)
Library	38.95	15.81	2.43	11.24
Flight	32.71	9.49	12.54	10.62
Parking	24.17	16.45	3.42	13.9
Service	66.62	36.29	6.65	−2.28
Library	48.84	5.36	11.96	23.63
Flight	29.83	7.55	−2.44	8.70
Parking	44.63	11.71	12.50	20.97
Service	32.04	33.84	18.25	–

7.3 Efficiency and Effectiveness of SPSA-Based Profiling

As mentioned in Sect. 4.2, each profiling run takes from a few minutes to hours. Therefore, to limit the time for identifying the best parameter setting, we consider the following constraints: i) for all algorithms, we set the maximum number of iterations to be 20; ii) for SPSA-based algorithm, we set $\mathcal{R} = 3$, which is the number of consecutive profiling iterations that do not produce any performance improvement over the best one observed so far; iii) additionally, for Tabu search, the tabu list size is set to be 3. We tabulate the average number of profiling runs in Table 4 for comparison.

Table 4. The average number of profiling runs for SPSA-based profiling (SPSA), Random Walk (RW), Tabu Search (Tabu), Linear Regression plus Dual-Simplex (LRDS), and Exhaustive Profiling (EP).

	SPSA	Random	Tabu	LRDS	EP
Number of runs	20	20	200	21	77,725

We further investigate if the parameter setting computed by the optimization framework from small-scale workflows can be applied to large-scale workflows. We sample a small amount of data from the original dataset and use the proposed profiling method to provide advice on the most suitable parameter settings. We apply these recommended settings to workflows processing the original dataset and plot the corresponding performance measurements in Fig. 7. These results show that the parameter setting recommended by the profiling approach from a small-scale workflow can significantly improve the performance of large-scale workflows in comparison with the default settings.

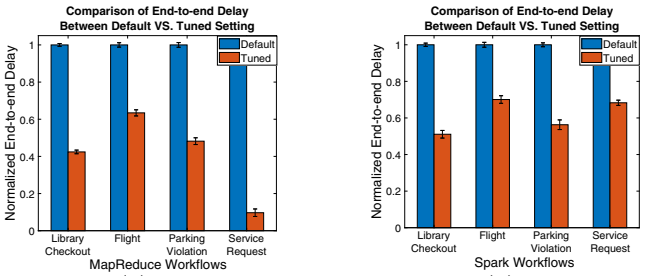


Fig. 7. Comparison of end-to-end delay between the default and recommended parameter settings across MapReduce/Spark, YARN and HDFS on large-scale workflows.

8 Conclusion and Future Work

We proposed a profiling-based coupled design framework for big data workflows, which provides end users with the most suitable system configuration to improve workflow performance in big data systems. Our framework employs feature selection and SPSA algorithm to accelerate the search process by obviating the need for exhaustive profiling.

It is of our future interest to provide configuration advice in finer granularity in the following aspects: i) recommend a suitable parallel computing engine, since for the same workflow, its performance varies considerably in Spark and MapReduce; ii) recommend parameter settings on a per-job basis, since not all jobs in the same workflow achieve its best performance with the same setting; iii) explore the most effective way to map parameter settings from small workflows to large ones; and iv) improve the performance of the SPSA-based profiling method, including gradient approximation averaging, step size adaptation, local optima prevention, convergence speed acceleration, and intelligent termination conditions, for faster profiling and more accurate recommendation.

Acknowledgments. This research is sponsored by U.S. National Science Foundation under Grant No. CNS-1828123 with New Jersey Institute of Technology.

References

1. Zaharia, P.W.M., Xin, R.S., et al.: Apache spark: a unified engine for big data processing. Commun. ACM **59**(11), 56–65 (2016)
2. Oinn, J.F.T., Addis, M., et al.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics **20**, 3045–3054 (2004)
3. Ludascher, I.A.C.B.B., et al.: Scientific workflow management and the Kepler system. Spec. Issue Workflow Grid Syst. **18**, 1039–1065 (2005)
4. Deelman, E., Blythe, J., et al.: Pegasus: mapping scientific workflows onto the grid. In: Dikaiakos, M.D. (ed.) AxGrids 2004. LNCS, vol. 3165, pp. 11–20. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28642-4_2

5. Kumar, G.M.V.S., Sadayappan, P., et al.: An integrated framework for performance-based optimization of scientific workflows. In: Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing, Garching, Germany, pp. 177–186 (2009)
6. Chiu, G.A.D., Deshpande, S., et al.: Cost and accuracy sensitive dynamic workflow composition over grid environments. In: Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing, Washington, DC, USA, pp. 9–16 (2008)
7. Holl, M.P.S., Zimmermann, O., et al.: A new optimization phase for scientific workflow management systems. Future Gener. Comput. Sci. **36**, 352–362 (2014)
8. Counaris, A., Torres, J.: A methodology for spark parameter tuning. Big Data Res. **11**, 22–32 (2018)
9. Wang, B.H.G., Xu, J., et al.: A novel method for tuning configuration parameters of spark based on machine learning. In: IEEE 18th International Conference on High Performance Computing and Communications, Sydney, NSW, Austrilia (2016)
10. Liao, G., Datta, K., Willke, T.L.: Gunther: search-based auto-tuning of mapreduce. In: Wolf, F., Mohr, B., an Mey, D. (eds.) Euro-Par 2013. LNCS, vol. 8097, pp. 406–419. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40047-6_42
11. Wu, A.G.D., et al.: A self-tuning system based on application profiling and performance analysis for optimizing hadoop mapreduce cluster configuration. In: 20th Annual International Conference on High Performance Computing, Bangalore, India (2014)
12. Li, S.M., Zeng, L., et al.: MRONLINE: MapReduce online performance tuning. In: Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing, New York, NY, USA, pp. 165–176 (2014)
13. Shu, T., Wu, C.: Performance optimization of *H*adoop workflows in public clouds through adaptive task partitioning. In: Proceedings of the IEEE INFOCOM, Atlanta, GA, USA, 1–4 May 2017
14. Wu, C., Lin, X., Yu, D., Xu, W., Li, L.: End-to-end delay minimization for scientific workflows in clouds under budget constraint. IEEE Trans. Cloud Comp. **3**(2), 169–181 (2015)
15. Yun, D., Wu, C., Gu, Y.: An integrated approach to workflow mapping and task scheduling for delay minimization in distributed environments. JPDC **84**, 51–64 (2015)
16. Ye, Q., Wu, C.Q., Cao, H., et al.: Storage-aware task scheduling for performance optimization of big data workflows. In: The 8th IEEE International Conference on Big Data and Cloud Computing, Melbourne, Australia, 11–13 December 2018
17. Wang, B.H.E.G. , Xu, J.: A novel method for tuning configuration parameters of spark based on machine learning. In: 2016 IEEE 18th International Conference on HPC and Communications, Sydney, NSW, Australia, 12–14 December 2016
18. Petridis, P., Gounaris, A., Torres, J.: Spark parameter tuning via trial-and-error. In: Angelov, P., Manolopoulos, Y., Iliadis, L., Roy, A., Vellasco, M. (eds.) INNS 2016. AISC, vol. 529, pp. 226–237. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-47898-2_24
19. Gounaris, A., Torres, J.: A methodology for spark parameter tuning. Big Data Res. **11**, 22–32 (2018)
20. Jia, G.C.E.Z., Xue, C.: Auto-tuning spark big data workloads on POWER8: prediction-based dynamic SMT threading. In: 2016 International Conference on Parallel Architecture and Compilation Techniques (PACT), Haifa, Israel, 11–15 September 2016

21. Holmes, A.: Hadoop in Practice. Manning Publications Co., Greenwich (2012)
22. Li, S.M.E.M., Zeng, L.: MRONLINE: MapReduce online performance tuning. In: Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing, Vancouver, BC, Canada, 23–27 June 2014
23. Ding, D.Q.E.X., Liu, Y.: Jellyfish: online performance tuning with adaptive configuration and elastic container in hadoop yarn. In: 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), Melbourne, Australia, 14–17 December 2015
24. Flight Data. http://stat-computing.org/dataexpo/2009/the-data.html
25. Library Checkout Data. https://data.seattle.gov/Community/Checkouts-by-Title/tmmm-ytt6
26. Parking Violation Data. https://data.cityofnewyork.us/City-Government/Open-Parking-and-Camera-Violations/nc67-uf89
27. Service Request Data. https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9
28. Spall, J.C.: Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Trans. Autom. Control **37**, 332–341 (1992)
29. Kiefer, J.W.J.: Stochastic estimation of the maximum of a regression function. Ann. Math. Stat. **23**(3), 462–466 (1952)
30. Spall, J.C.: Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control. Wiley, Hoboken (2005)
31. Ross, B.: Mutual information between discrete and continuous data sets. PLOS ONE **9**(2), 1–5 (2014)
32. Abramowitz, M., Stegun, I.: Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. Dover Publishing Inc., New York (1972)
33. Spall, J.C.: Implementation of the simultaneous perturbation algorithm for stochastic optimization. IEEE Trans. Aerosp. Electron. Syst. **34**, 817–823 (1998)
34. Heger, D.: Hadoop performance tuning-a pragmatic & iterative approach. CMG J. **4**, 97–113 (2013)
35. White, T.: Hadoop: The Definitive Guide. O'Reilly Media Inc., Sebastopol (2012)
36. Lawler, G., Limic, V.: Random Walk: A Modern Introduction. Cambridge University Press, Cambridge (2010)
37. Glover, F.: Tabu search: a tutorial. Informs J. Appl. Anal. **20**(4), 1–185 (1990)
38. Montgomery, E.A.P.D.C., Vining, G.: Introduction To Linear Regression Analysis, vol. 821. Wiley, Hoboken (2012)
39. Nocedal, J., Wright, S.: Numerical Optimization. Springer, Heidelberg (2006)
40. Apache, Hadoop (2016). http://hadoop.apache.org
41. Spark (2016). http://spark.apache.org
42. Oozie (2016). https://oozie.apache.org