JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 36, 469-493 (2020)
DOI: 10.6688/JISE.202005_36(3).0001

Big Data Platform Configuration Using Machine Learning*

CHAO-CHUN YEH[1,3], HAN-LIN LU[1], JIAZHENG ZHOU[3], SHENG-AN CHANG[3], XUAN-YI LIN[3], YI-CHIAO SUN[3] AND SHIH-KUN HUANG[1,2]
[1]*Department of Computer Science*
[2]*Information Technology Service Center*
National Chiao Tung University
Hsinchu, 300 Taiwan
[3]*Computational Intelligence Technology Center*
Industrial Technology Research Institute
Hsinchu, 300 Taiwan
E-mail: {avainyeh; zhou; changshengan; xylin; icsun}@itri.org.tw;
{luhl; skhuang}@cs.nctu.edu.tw

By ensuring well-developed complex big data platform architectures, data engineers provide data scientists and analysts infrastructure with computational and storage resources to perform their research. Based on such supports, data scientists are provided an opportunity to focus on their domain problems and design the required intelligent modules (*i.e.*, prepare the data; select, train, and tune the machine-learning modules; and validate the results). However, there are still gaps between system engineering and data scientist/engineering teams. Generally, system engineers have limited knowledge on the application domains and the purposes of an analytical program. On the contrary, both data scientists and engineers are usually unfamiliar with the configuration of a computational system, file system, and database. However, the performance of an application can be affected by a system's configuration, and the data scientists and engineers have little information and knowledge about which of the system's properties can affect the application's performance. As a typical example, for Internet-scale applications that have thousands of computing nodes or billions of Internet of Things devices, even a slight improvement may have an enormous influence on energy management and environmental protection issues. To bridge the gap between system engineering and data scientist/engineering teams, we proposed the concept of a configuration layer based on a big data platform, Hadoop. We built a configuration tuner, BigExplorer, to collect and preprocess data. Furthermore, we also created golden configurations for performance improvement. Based on the processed data, we used a semi-automatic feature engineering technique to provide more features for data engineers and developed the performance model using three different machine learning algorithms (*i.e.*, random forest, gradient boosting machine, and support vector machine). Using the commonly used benchmarks of WordCount, TeraSort, and Pig workloads, our configuration tuner achieved a significant performance improvement of 28%-51% for different workloads than using the rule-of-thumb configuration.

***Keywords*:** big data platform, machine learning, configuration optimization, learning by design, algorithms

1. INTRODUCTION

With the emergence of big data and machine learning, smart applications, such as Go with alphaGo [1], healthcare with Watson [2], and virtual voice-controlled assistants

Received August 14; revised June 18 & August 2, 2019; accepted September 18, 2019.
Communicated by Hung-Yu Kao.
* Part of this work was presented in TAAI 2016 Conference, Hsinchu, Taiwan, Nov, 2016.

such as Siri [2], are becoming part of our daily lives. Applications with human-like intelligence have shown extraordinary achievement in three important aspects: data, algorithms, and platforms. Data generated from social media, search history, and system logs not only make diverse applications practical but also make machine learning (*i.e.*, algorithms) more robust and accurate for these applications.

In addition to data and algorithms, the selection of a platform plays an important role in such applications. Currently, there are many platform architectures (such as Hadoop [3], Spark [4], Flink [5], and H_2O [5]) for different application requirements (such as high-volume data, and low-latency and high-frequency iteration data processing).

Because platform architectures have become more complicated, knowledge workers (such as data scientists, data engineers, and system engineers) perform their tasks according to a function-based division of labor. For instance, system engineers provide the infrastructure with computation and storage resources for data scientists and engineers. With their generous support, data engineers can focus on data pre-processing, whereas data scientists can focus on their domain problems and on designing intelligent modules (such as feature engineering, machine-learning module selection, and result validation).

However, for big data platforms, there are gaps among system engineering, data engineering, and data scientist teams. System engineers do not have adequate knowledge of the application domains and the purposes of an analytical program, while data scientists and engineers do not know the configuration of the computation and file systems as well as the database. Certain application performance issues are related to system configurations. Generally, data scientists and data engineers do not have adequate information and knowledge regarding system properties. For instance, certain workloads might perform better (such as low latency or high throughput) if the platform system is adjusted to the proper configuration. Alternatively, a workload might crash because of misconfiguration [6]. Using results obtained from our cluster, we observed the same workload with different configurations and discovered that certain workloads became strugglers (an execution time of > 1,000,000s) because of a misunderstanding of configuration properties. Fig. 1 shows such a situation.

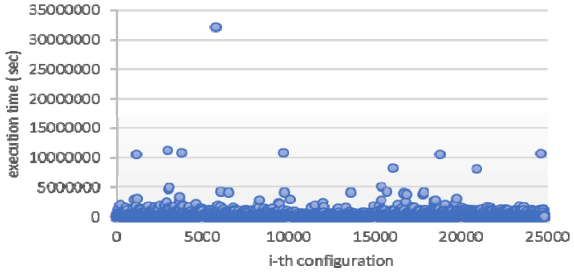

Fig. 1. Execution time for different configurations.

Fig. 2 shows an overview of a big data platform with a configuration layer. Unlike traditional architectures, the configuration layer is added to fill the gap between the data engineering and data scientist teams. For example, the configuration layer can provide better parameters for various analytical applications and make the system learnable for performance improvement. In a learnable system, this is a key component as to be explained in detail on our system, BigExplorer, in Section 3.

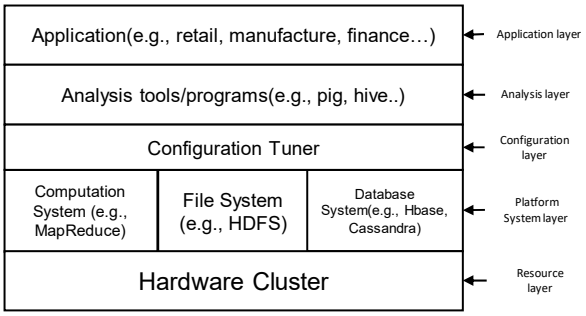

Fig. 2. The overview of a big data platform with configuration layer.

In a complex platform, there are often many optimization targets such as memory usage, disk storage, network bandwidth, and workload execution time. For smart applications such as gaming or voice-controlled assistants, the environment is often in a cloud or cluster. This indicates that the application needs to use on-demand hardware resources; therefore, the response time is critical for such applications.

To validate the abovementioned claim using execution time optimization as an example, we collected 648 samples (Fig. 3) running the same analytical workload with different configurations (nine properties) in our cluster. Using the same workload, we observed that 17.90% of the configurations yielded better performance in terms of execution time than default configurations [1] (dotted line). Interestingly, for Internet-scale applications that have thousands of computing nodes or billions of Internet of Things devices, it is reported that even a slight improvement could have an enormous influence on energy management and environmental protection issues. For instance, to save energy, Google [7] and Facebook [8] manage their data centers using a data-driven approach.

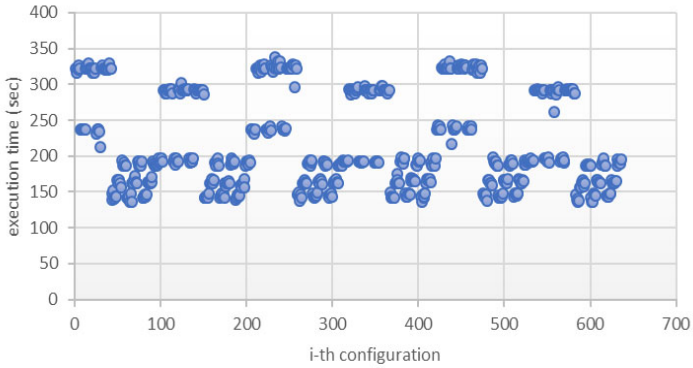

Fig. 3. Execution times of the sample workloads.

Motivated from these results, we separated the configuration layer and built the configuration tuner to generate parameters for learning by design. The configuration tuner is a key learning component of big data platforms because it collects the observable and tunable data for training, builds machine learning models for simulation, and optimizes the simulation results for configuration tuning.

2. BACKGROUND

In this section, we present a technical overview of the selected big data platform and benchmarks. Furthermore, we introduce relevant machine learning techniques and computational optimization methods.

2.1 Big Data Platform

The characteristics of big data such as volume, velocity, and variety, pose new challenges, problems, and opportunities for the research community. For the high-volume characteristic, data engineers use innovative system architectures to improve the system throughput. For the high-velocity characteristic, data engineers leverage special purpose software architectures, such as Storm [9] or Spark Streaming [10], to address possible challenges. For the high-variety characteristic, data are obtained from multiple sources with different formats. Data engineers collect data using new-generation database systems (such as Mongo DB [11] or Titan [10]), and data scientists analyze different datasets to gain insight.

Hadoop is the most popular big data platform that has a complete ecosystem [12], including a distributed filesystem, distributed programming, NoSQL/NewSQL databases, data ingestion, job scheduling, security, and machine learning. Hadoop can process large-scale datasets across a cluster of distributed machines using the MapReduce programming model. In this study, we used Hadoop to re-verify the concept of our proposed configuration layer.

2.2 Resource Management Layer

In an earlier version of Hadoop (*i.e.*, version 0.23), the resource management design was closely coupled with the computational model, thus causing poor scalability [13]. To ameliorate the situation, both YARN [14] and Mesos [15] were proposed.

Note that YARN incorporates a resource management center that coordinates various applications and handles Hadoop system resources with node manager agents that monitor the operations (such as data processing) of individual nodes within the cluster. Among multiple tenants, the resource manager monitors resource usage and node status by separating the central resource allocator in the role of JobTracker. Note that this responsibility is delegated to the master node that coordinates the logical plan of a specific job by demanding resources from the resource manager, thus generating a physical plan from the resources it receives.

Mesos is a lean management layer that allows various cluster computing frameworks to efficiently share resources. Mesos includes two design components: a fine-grained sharing model and a distributed scheduling mechanism, known as resource offers. Using these components, Mesos can achieve high utilization, responsibility, and scalability [15].

2.3 Big Data Benchmarks

There are many well-known big data benchmarks such as TPC-C [16], HiBench [17], Yahoo! Cloud Serving Benchmark (YCSB) [18], and Cloudsuite [19]. The purpose of

having distinct big data benchmarks is to reflect real-life use cases (such as data analytics, media streaming, data serving, and data searching) and involve numerous types of implementation options (such as latency, throughput, scaling, and resource usage). However, we focused on the configuration effects on various applications. We selected the Word Count and TeraSort benchmarks from HiBench because of their different characteristics in terms of resource requirements. Note that Word Count is a CPU-bound benchmark and TeraSort is mostly I/O-bound [20]. In addition to the TeraSort and Word Count benchmarks, we used the Business Intelligence analytic workload (*i.e.*, Pig scripts [21]) for complicated cases.

2.4 Machine Learning

With the emergence of big data, machine learning can now play important roles for solving real-world problems. Typically, depending on the properties of their feedback to learning problems, machine learning algorithms are classified into four categories.

1. Supervised learning
   The feedback is well-defined and can be obtained clearly from a learning system.
2. Unsupervised learning
   The feedback is not well-defined and cannot be obtained from a learning system.
3. Reinforcement learning
   The feedback can be presented as a specific goal without any explicit feedback.
4. Semi-supervised learning
   The feedback is well-defined, and some feedback can be obtained from a learning system. Note that a learning algorithm might be intended to obtain the desired feedback by interactively querying the information source.

Our domain problem is a supervised learning example of the regression type because the feedback data (execution times) are well defined.

2.5 Computational Optimization

Optimization is aimed at selecting the best element (*i.e.*, maximum or minimum value) concerning specified constraints in the given data space. There are various computational optimization techniques, such as iterative methods (Newton's method [22, 23]), gradient descent [24], and heuristics (evolutionary algorithms [25], genetic algorithms [26], hill climbing [27], and particle swarm optimization (PSO) [28]). We used PSO as our optimization method in this study because it can search huge spaces of candidate solutions based on few assumptions regarding the problem [29].

3. SYSTEM DESIGN

In this section, we introduce our configuration tuner design. In Section 3.1, we will discuss the overview of system design. Subsequently, we introduce data management, performance model, and configuration optimization designs in Sections 3.2, 3.3, and 3.4, respectively.

3.1 Overview of System Design

We generalize the platform parameter tuning problem as follows. Given a platform with parameters and a corresponding performance index, we will produce a set of better-fine-tuned configurations on the given platform. For data gathering and as a proof of concept, we used a big data platform (Hadoop) as a specific example. From abstract representations to a big data platform, a platform behavior depends on a workload set $\boldsymbol{W}$ (such as input data, and source or binary programs to process the data), as shown in Eq. (1), and a parameter set $\boldsymbol{P}$ (*e.g.*, hardware and software parameters), as shown in Eq. (2). Note that the HiBench [17] benchmark includes seven different purposes of programs such as Sort, WordCount, TeraSort, PageRank, *K*-means, Bayes classification, and Index. The other commonly-used benchmark for big data applications BigDataBench [30], covers six applications with the different program sets such as micro utilities (sort, grep, WordCount, and BFS), datastore operations (read, write, and scan), relational query (*i.e.*, select, aggregate, and join), search engine (index and PageRank), social network (*i.e.*, *K*-means and connected components), and E-commerce (collaborative filtering and naïve Bayes).

$$\text{Workload set } W = \{w_1, w_2, w_3, \ldots, w_i\} \tag{1}$$

$$\text{Parameter set } P = \{p_1, p_2, p_3, \ldots, p_j\} \tag{2}$$

However, because it is difficult to understand and represent a platform behavior $\boldsymbol{S}$, system engineers use a representative performance index set $\boldsymbol{PX}$ (including throughput, response time, latency, utilization, and saturation) to describe system behavior, as shown in Eqs. (3) and (4).

$$\mathbf{S} \approx \boldsymbol{PX} \tag{3}$$

$$\text{Performance Index set } \boldsymbol{PX} = \{px_1, px_2, \ldots, px_k\} \tag{4}$$

In a learning system, we assume that the performance index behavior $\boldsymbol{PX}$ can be represented by a function $\boldsymbol{F}$ of the given parameter set $\boldsymbol{P}$ and workload $\boldsymbol{W}$, as shown in Eq. (5). Then, we can employ a set of machine learning algorithms $\boldsymbol{MFL}$ to determine the best algorithm that minimizes the error between the outcome of F and the outcome of $\boldsymbol{MFL}$ as shown in Eqs. (5) and (6).

$$\boldsymbol{PX} = \boldsymbol{F}(\boldsymbol{W}, \boldsymbol{P}) \tag{5}$$

$$\text{Machine-Learning Algorithm set } \boldsymbol{MFL} = \{a_1, a_2, a_3, \ldots, a_k\} \tag{6}$$

$$\boldsymbol{S} \approx \boldsymbol{PX} = \boldsymbol{MFL}_{\text{opt}}(\boldsymbol{W}, \boldsymbol{P}) \tag{7}$$

In the next section, we present the Hadoop platform as an example (Fig. 4). We used a performance index $\boldsymbol{PX}$ to represent the system behavior $\boldsymbol{S}$, and we assumed $\boldsymbol{PX}$ (such as execution time or resource usage) could be represented by an unknown function *F*, which is related to the given parameter set $\boldsymbol{P}$ (such as configuration of MapReduce or HDFS) and workload $\boldsymbol{W}$ (such as TeraSort and WordCount). Based on a collection of

parameter set and workload, we evaluated all machine learning algorithms in ***MFL*** and selected the best algorithm that minimizes the error between the outcome of F and the outcome of ***MFL***.

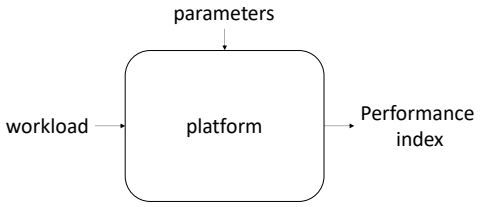

Fig. 4. Performance model of the platform system.

The system process is described in the following steps:

1. Collecting and managing the platform data
   Platform data such as parameters, workload, and performance index can be obtained from the log or plug-in modules. We collected the data from distributed nodes into a unified data store. By doing so, the collected data quality can be measured by querying with missing value, outlier, and irregular cardinality filters.
2. Building and testing the performance model
   By unifying the data storage, we selected the target performance index and parameters as the training features based on their quality. Furthermore, we built the performance model using different machine learning algorithms with cross-validation [31] to ease the overfitting issue [32].
3. Generating and verifying the parameters
   Given the performance model, we used optimization technology (such as metaheuristics) that can sufficiently provide good parameters to solve the platform performance optimization problem.

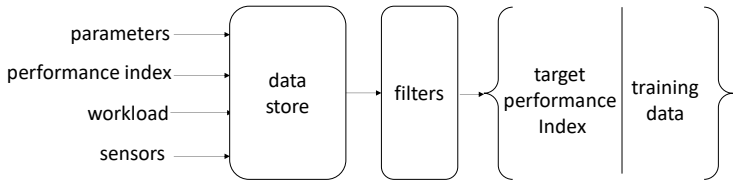

Fig. 5. Data flow for data management.

3.2 Data Management Design

In the data management design stage (Fig. 5), there are two principal issues: (1) What type of data can be collected? and (2) How can the data quality be improved? Using extract-transform-load (ETL) tools, it is easy to aggregate data from distributed nodes into a unified data store for data collection. However, certain raw data may not be sufficient for representing the system's performance, thereby possibly leading to sampling bias [33]. To overcome this limitation, system engineers deploy more sensors for

gathering new features using the platform's plug-in/add-on functions. Note that more sensors can not only be used to collect more data but also to create an impact on the system's performance. For instance, for a cluster's performance, if the experimental raw data in Table 1 only contains features such as maximum memory allocation (*i.e.*, all cluster physical memory bytes), it is not sufficient to represent the system performance for a fine-grained perspective. For the fine-grained perspective, system engineers deploy sensors for gathering new features such as run-time process memory usage (*i.e.*, all map-reduce framework physical memory bytes and all map-reduce framework virtual memory bytes). Fig. 6 shows the lines for the three different features as a log axis scatter diagram. We determined that maximum memory allocation (*i.e.*, all cluster physical memory bytes) is a stable line without variation and features of run-time process memory usage (*i.e.*, all map-reduce framework physical memory bytes and all map-reduce framework virtual memory bytes) are more diverse compared to the actual run-time memory usage.

Table 1. Features of the cluster memory.

All map reduce framework Physical memory bytes	All map reduce framework Virtual memory bytes	All cluster physical memory bytes
305303552	1397387264	305488396288
326262784	1382666240	305488396288
329830400	1382506496	305488396288
347770880	1382518784	305488396288
348684288	1383247872	305488396288
349093888	1408602112	305488396288
349839360	1383165952	305488396288
353980416	1405599744	305488396288

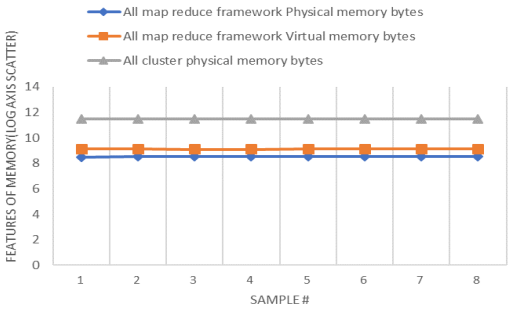

Fig. 6. Different features for cluster memory.

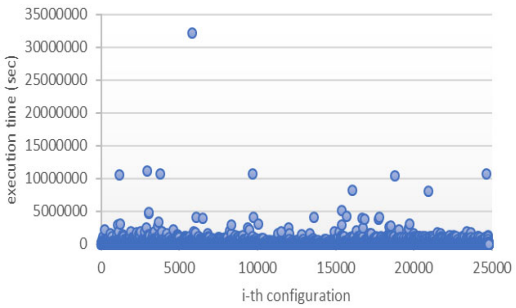

Fig. 7. Execution time for outliner.

For improving data quality, we considered the logical cases of invalid data such as missing value, outlier, and irregular cardinality, and designed some filters for those cases when the data are used for analytics.

• Outliner Case

As shown in Fig. 7, the outliner case takes > 30,000,000s to complete the task. If we use the record for an experiment, it could affect the statistic properties of other normal samples. For such cases, the deletion filter is often employed.

• Missing Value Case

As shown in Fig. 7, if certain missing value cases occurred, we often design filters such as mean substitution, regression, or clustering to impute the value.

Furthermore, domain knowledge such as system specifications and physical rules must be considered for cases involving an in-depth analysis. For instance, if the data for maximum memory allocation is missing, it could be found from another history log. Using logical and domain filters, data can be exported as the target performance index and training features for performance model building.

3.3 Performance Model Design

Fig. 8 shows the performance model design workflow, including learning algorithm selection, feature selection, and tuning of algorithm parameters.

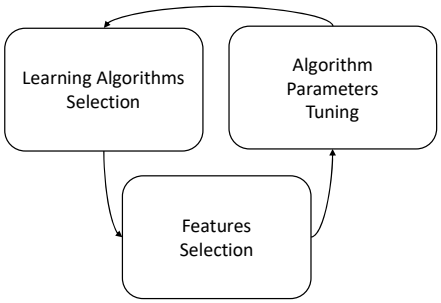

Fig. 8. Performance model design workflow.

• Learning Algorithm Selection

A learning algorithm that considers an approximation function to a system model requires different complexity, hyperparameters, and theories for the analyst's selection. At this stage, we provide suggestions for different purposes.

➢ Problem view

A system problem can be a state decision (such as normal or abnormal) or system performance prediction (such as throughput, latency, and power consumption). Such a problem can be abstractly viewed as classification, clustering, regression, or dimensionality reduction; therefore, it can be solved using the proposed algorithms.

➢ Explainable for human understanding

The other important factor in selecting a candidate learning algorithm is its explainability or ability to be easily understood by a human being. Some algorithms resemble black boxes with good performance (such as support vector machines (SVMs) or deep learning) and use hyperparameters (such as soft margin for SVMs and number of layers and number of epochs for deep learning). Other algorithms, such as regression, decision trees, *k*-nearest neighbors, *k*-means, and sequence pattern mining, are easy to understand and possess readily explainable properties.

➢ Algorithm theory view

In machine learning, the No-Free-Lunch (NFL) theorem indicates that, if there are no prior assumptions, all algorithms perform equally. However, data scientists' attempts to determine a suitable learning algorithm can lead to extraneous background information such as assumptions and related work. This background information can suggest many learning algorithms, *e.g.*, rule-based, entropy-based, probability-based, error-based, metaheuristic, or neural network (NN) algorithms.

• Feature Selection

For platform parameter tuning, feature selection (*i.e.*, variable or attribute selection) is an important part of the analytical process. Feature selection involves selecting the most dominant features from the raw feature set for the learning algorithm to ensure better performance, cost effectiveness, and understandability [33], as well as to remove redundant and irrelevant features. In general, there are three types of feature selection algorithms: filter, wrapper, and embedded.

Without using machine learning algorithms, filter methods use a statistical measure (such as information gain, Chi-squared, or correlation coefficients) to obtain the feature score with the independent variable assumption. Note that features are selected to be retained or removed based on a score ranking.

Unlike filter methods, wrapper methods incorporate a machine learning algorithm and use search techniques (such as best-first search, random hill-climbing, or forward and backward passes) in identifying the best feature combination set with a given predictive model. Unlike filter and wrapper approaches, embedded methods combine the learning algorithm and feature selection to perform feature selection while executing learning algorithms. The typical embedded methods are tree-based algorithms (such as CART, C4.5, and random forest) and regularization models with feature weights (such as Lasso and Elastic Net).

• Model Tuning

After the evaluation matrix (such as RMSE, AE, and ROC) is selected to represent how a model performs on training and testing data, the next key step is model tuning to improve the model and represent the original data without any bias. There are two typical approaches for model tuning: parameter-related and parameter-free. Parameters of the model (hyperparameters) such as learning rate, constraints, and weights must be tuned to minimize the predefined evaluation matrix on given training data using a grid, random, or gradient-based search and Bayesian-based or evolutionary optimization. To reduce model performance variability for overfitting or underfitting, we use regularization to produce reasonable solutions to ill-posed problems [34]. In addition to the original target function, regularization adds weight constraints for different norms (such as the L1- or L2-norm) to obtain better results. The L1-norm can retain important features but is unstable [35], while the L2-norm is more stable but not robust [36]. Dropout is a type of variant weight level regularization for NNs. It drops certain nodes and the related weights for these nodes in the training phase, resulting in the training of a subnetwork.

Furthermore, parameter-free approaches focus on data. Note that cross-validation focuses on the data itself and adjusts the data partition for resampling, resulting in an effective approach when data collection is not high in terms of volume. Using the right problem of domain knowledge, data arguments use operations or transformations on data

to create new data representations without losing its essence. For instance, objects in image and video data must be identical when subjected to the rotation, resizing, and clip operations. Early stoppage is used to observe the error rate of training and testing data to select a suitable stopping point that helps avoid overfitting. The stopping condition is the beginning of increasing testing error when the training error starts to decrease [37].

After tuning, the model can be used for configuration optimization and the workflow can be re-executed if new data are available with new system components.

3.4 Configuration Optimization

The key concept of optimization is to identify suitable and available solutions of objective functions under specified constraints in predefined domains such as science, engineering, and management. Traditionally, mathematicians and scientists treat these types of problems as convex problems and use linear programming and gradient-based methods to solve them. However, real-world problems include many non-convex optimization problems that mix different perspectives such as continuous and discrete, linear and nonlinear, and local and global. Determining whether an optimization problem is convex is an NP-hard problem. The strategy for solving non-convex problems is to relax the non-convex problem to a convex problem and leverage randomized techniques (such as stochastic optimization) for every sub-convex problem.

When considering optimization problems from a configurational perspective, configuration optimization involves selecting the best configuration in connection while considering constraints (such as workload properties, configuration parameters, and resource limitation) affecting the system model. In the previous step, we obtained the system model, $\boldsymbol{MFL_{opt}(W, P)}$, which can be used to select training data from a collection of data to represent the system behavior S, as shown in Eq. (7).

For a given model F with fixed W, the parameter set could be discrete and nonlinear (such as memory size and max task), and their combination could be discrete. Therefore, the configuration optimization aim is to identify a feasible P with discrete and nonlinear constraints. Traditionally, these types of problems are NP-hard [38]. Metaheuristics [39] are techniques for obtaining a global optimization strategy with discrete and nonlinear properties. For complex learning procedures, they are efficient search approaches for exploring near-optimal solutions in the search space and adaptive strategies from simple local search procedures. They can be classified into two principal categories: single solution-based (such as trajectory methods, simulated annealing, Tabu search, GRASP methods, iterated local search, and guided local search) and population-based methods. Population-based metaheuristics can be used to generate a set (*i.e.*, a population) of solutions rather than a single solution. These approaches are inspired by natural phenomena such as Darwinian evolution and the social interaction of living creatures. For optimization of the configuration tuner, we used PSO, which is based on swarm intelligence, including multiple simple entities for executing uncomplicated tasks and interacting locally with one another and with the environment.

4. SYSTEM IMPLEMENTATION

BigExplorer was developed on a 2.7 GHz i7 CPU with 32 GB RAM machine and

Ubuntu 16.04 64-bit desktop edition and was written in Python 2.7. We employed the machine learning library scikit-learn 1.8 and the evolutionary optimization package pyswarm 0.6 [40]. Hadoop 2.6.0 was deployed with a 10-node cluster size with each node having 12 cores (2.4 GHz/core) and 128 GB RAM. Fig. 9 shows the architecture of BigExplorer with its three modules.

- Data collection module
  This module collects data from a big data platform (such as Hadoop), cleans the data (such as missing values or data formats), and stores the data as plain text in the file system.
- Performance model module
  To build the performance prediction model, this module provides potential features for data scientists to select, use machine-learning algorithms with the training data, as well as store the model for the next stage.
- Configuration optimization module
  This module leverages the optimization algorithm to generate parameters and then use them as inputs toward the performance simulator and obtain various simulation results. The parameter validator was used to select candidate parameters as the new configuration and validate the simulation results.

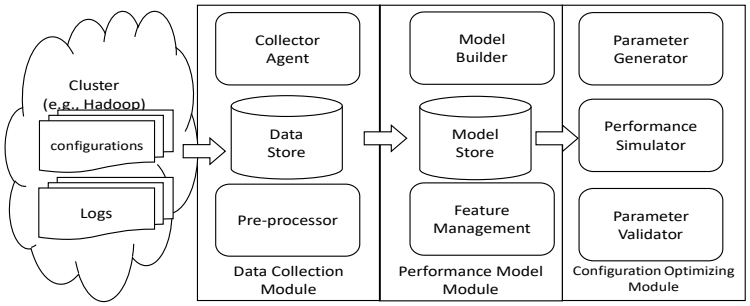

Fig. 9. BigExplorer architecture.

4.1 Data Collection Module

The data collection module has three components. Collector Agent collects logs from nodes in the cluster. For a Hadoop cluster environment, the agent collects the MapReduce workload log (*i.e.*, job history), system log (*i.e.*, YARN container execution history), and cluster configuration (*i.e.*, config.xml). Both logs and configurations are stored in the data store with the workload information (such as application type and completion time). The pre-processor rearranges the files on the data store, processes them using different filters (such as missing value, min–max, and invalid execution time filters) and saves the results in a plain text format.

4.2 Performance Model Module

The performance model module has three components. The feature management

component uses a random forest (RF) algorithm to provide potential features for data scientists. Based on the selected features, the model builder component divides the collected data into training (70%) and testing (30%) sets for the machine learning algorithms to build the performance model. The machine learning algorithms considered are as follows:

• Random Forest

RF [41], one of the algorithms widely used in big data competitions [27], is an ensemble learning method that generates many independent decision trees for classification of tasks and fits them to assorted sub-samples of the data (*i.e.*, bootstrap aggregating or bagging). To avoid the effects of over-fitting, it uses the averaging generalization error (such as out-of-bag error) to improve predictive accuracy. For regression problems, it outputs the average prediction of the trees rather than that of the class, as shown in Eq. (8) [42].

$$\hat{f}_{RF}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x) \tag{8}$$

• Gradient Boosting Machines (GBM)

Gradient boosting machines (GBM) [43, 44], another algorithm that is extensively used in big data competitions [27], is an ensemble learning method that is based on bagging technique. Unlike RF, gradient tree boosting sequentially builds sub-trees by optimizing the loss function. To solve a regression problem, it fits a regression subtree to the negative gradient of the given loss function, as shown in Eq. (9) [42, 44].

$$\tilde{\theta} = \arg\min_{\theta} \sum_{i=1}^{N} (-g_{\Im} - T(x_i;\theta))^2 \tag{9}$$

• Support vector regression (SVR)

SVMs [45] are used to solve various real-world problems (such as text and hypertext categorization, image segmentation, and hand-written characters) by building hyperplanes according to a functional margin in high- or infinite-dimensional space. The best separation of the margin is the largest area that is closest to the training data surface of any class data.

SVR [46] employs the SVM approach to regression problems. To solve a regression problem, we minimize the bound on infeasible constraints and a generalized error that considers a regularization term rather than only a minimizing function of the errors on the training set as shown in Eq. (10) [46]. Moreover, we used a sequential minimal optimization algorithm to solve the SVM optimization problem [47]. By constructing a Lagrange function from the object function, we can reformulate the available solution with a kernel function, as shown in Eq. (11). In this study, we use a radial basis function (RBF) as the default kernel with SVR.

$$\min_{w,b} \frac{1}{2} \| \omega \|^2 + C\sum_{i=1}^{m} l_{\varepsilon}(f(x_i) - y_i) \tag{10}$$

where l_{ε} denotes insensitive loss.

$$f(x) = \sum_{i=1}^{m} (\hat{\alpha}_i - \alpha_i)\kappa(x, x_i) + b \tag{11}$$

where $\kappa(x, x_i)$ denotes a kernel function.

During the performance model building stage, we used the model with threefold cross-validation. After performing cross-validation, the models trained with different algorithms were stored in the model store in the PMML format.

4.3 Configuration Optimizing Module

The parameter generator produces parameters based on the given constraints and uses the parameters as input to the performance simulator. The performance simulator is based on the performance model and obtains the simulated results with the given input while the parameter validator uses certain top-ranking configurations to validate the simulation results.

In this module, we used PSO as our optimization algorithm. PSO is a type of swarm intelligence technique that optimizes a domain problem by attempting to improve a population of candidate solutions in an interactive manner using the given environmental measure. PSO moves the particles in the search space according to the particle's position X and velocity V, and the movement of each particle can be influenced by its local best-known position P_{id} and the best-known position P_{gd} in the global search space. Note that the velocity and position are updated according to Eqs. (12) and (13), respectively.

$$V_{id}(t+1) = V_{id}(t) + C_1\varphi_1(P_{id}(t) - X_{id}(t)) + C_2\varphi_2(P_{ad}(t) - X_{id}(t)) \tag{12}$$

φ_1 and φ_2: random numbers with uniform distribution in [0,1]
C_1 and C_2: acceleration coefficients

$$X_{id}(t+1) = X_{id}(t) + V_{id}(t+1). \tag{13}$$

In Section 5.4, we present the design of the PSO experiment to compare its performance under different parameter settings because PSO does not guarantee a globally optimal solution. The purpose of the experiment is to identify the impact of different parameters of the particle number and the max-iteration on the performance of the system.

5. SYSTEM EVALUATION

In this section, we first explore the experimental data and list the features we selected to build our performance model. Using this model, we compare the simulation and validation results using different machine learning algorithms.

5.1 Explorer Data

We collected three application workloads (*i.e.*, TeraSort with 11,657 records, Word Count with 16,035 records, and Pig script with 11,675 records) from our cluster and used

them for our system evaluation. Fig. 10 shows the execution time of the three applications. From the box plot, we determined that Word Count takes more time with more variation than TeraSort. There are certain outliers because of misconfiguration, which resulted in certain failed jobs. To determine the baseline of the performance improvement, we measured the system background noise by repeating the workload with the same configuration. The deviation ranges from 0.04% to 3.06%.

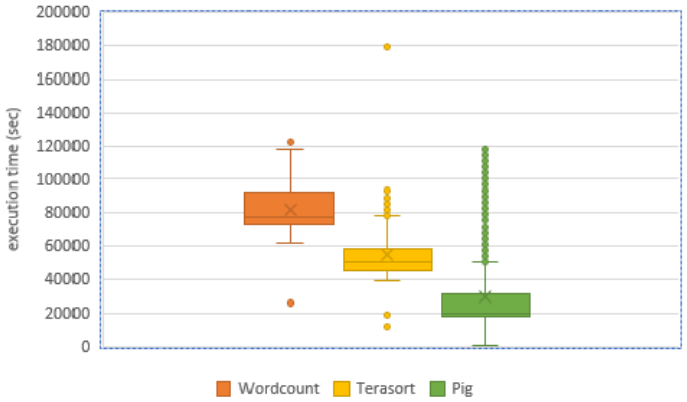

Fig. 10. Box plot of *WordCount*, *TeraSort*, and *Pig*.

5.2 Feature Selection

Originally, there were ~190 configuration parameters associated with the Hadoop system configuration. We used RF algorithm for ranking feature importance [48] and then divided them into three groups such as high (feature importance greater than 0.04), medium (feature importance between 0.04 and 0.01), and low (feature importance of < 0.01). With the three groups, we selected the high group for feature engineering, 24 of which (15 relate to Word Count and TeraSort and nine relate to Pig) are shown in Table 2. The feature types include Boolean (MapOutputCompress, OutputCompress, PigCombination, and PigFileCompression), number (ShuffleMergePer, ReduceCopyNum, and PigBytePerReducer), and categorical (PigCodec). Certain feature constraints are related to software (JVMReuse for Java virtual machine design) or hardware (SortMB for memory size).

5.3 Model Performance Results

We used the mean absolute percentage error (MAPE) [49], as shown in Eq. (14), to evaluate the accuracy of the model. In the abovementioned formula, n denotes the total number of experimental cases, A_i denotes the actual value, and F_i denotes the forecast value.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{A_i - F_i}{A_i}\right| \tag{14}$$

Table 2. Parameter abbreviation and feature importance of word count, terasort, and pig.

Abbreviation	Parameter in Hadoop Configuration	Feature Importance
SplitSize	mapreduce.input.fileinputformat.split.minsize	0.053
HttpThread	mapreduce.tasktracker.http.threads	0.042
JVMReuse	mapreduce.job.jvm.numtasks	0.058
ShuffleMergePer	mapreduce.reduce.shuffle.merge.percent	0.061
ReduceSlowstart	mapreduce.job.reduce.slowstart.completedmaps	0.063
InMemMergeThreshold	mapreduce.reduce.merge.inmem.threshold	0.065
SortPer	mapreduce.map.sort.spill.percent	0.056
ShuffleInputPer	mapreduce.reduce.shuffle.input.buffer.percent	0.059
ReduceCopyNum	mapreduce.reduce.shuffle.parallelcopies	0.082
SortMB	mapreduce.task.io.sort.mb	0.063
MapOutputCompress	mapreduce.map.output.compress	0.084
MapTasksMax	mapreduce.tasktracker.map.tasks.maximum	0.044
RduceTasksMax	mapreduce.tasktracker.reduce.tasks.maximum	0.051
SortFactor	mapreduce.task.io.sort.factor	0.058
OutputCompress	mapreduce.output.fileoutputformat.compress	0.047
PigCacheMemusage	pig.cachedbag.memusage	0.048
PigMapPartAgg	pig.exec.mapPartAgg	0.056
PigMinReduction	pig.exec.mapPartAgg.minReduction	0.063
PigBytePerReducer	pig.exec.reducers.bytes.per.reducer	0.051
PigReducersMax	pig.exec.reducers.max	0.046
PigReduceMemusage	pig.skewedjoin.reduce.memusage	0.045
PigCombination	pig.splitCombination	0.056
PigFileCompression	pig.tmpfilecompression	0.062
PigCodec	pig.tmpfilecompression.codec	0.042

Table 3. MAPE of different models.

Model Name	MAPE
GBM	0.29
RF	0.37
SVR	0.27

As shown in Table 3, the GBM and SVR models perform well at the same level and yield better MAPE results than the result obtained using the RF model.

However, based on the time of model building and cross-validation, we used RF as the basic unit and compared the time (*i.e.*, model building and cross-validation) with that of the other two models. In Tables 4 and 5, the GMB and SVR models are approximately one and five orders of magnitude, respectively, significant as RF in terms of time. In general, the SVR model produced the best results, but with extremely high training costs.

Table 4. Ratios of model building times.

Model Name	Time Ratio
GBM	7.38
RF	1
SVR	66,0731.42

Table 5. Ratios of cross-validation times.

Model Name	Time Ratio
GBM	8.87
RF	1
SVR	212604.73

5.4 Simulation Results

From the performance simulation conducted using the generated parameters, we obtained the simulation results using PSO with 300 particles and five iterations for various models. For straightforward comparison, we used RF as the basic unit and compared the prediction results (*i.e.*, execution time) between the models built from distinct machine learning algorithms. From Table 6, it is obvious that SVR produces better prediction results than RF and GBM and improves the TeraSort performance over that of Word Count. However, for Pig workloads, GBM shows better prediction results than the other two models.

Table 6. Ratio of prediction of execution times.

Model \ App.	TeraSort	WordCount	Pig
GBM	1.02	1.30	0.98
RF	1.00	1.00	1.00
SVR	0.75	0.89	1.07

Table 7. Ratio of prediction of execution times with different particles iterations settings.

			App								
			TeraSort			WordCount			Pig		
		i \ *p*	200	300	400	200	300	400	200	300	400
Models	GBM	3	2.76	1.75	1.43	3.73	2.57	1.83	2.92	2.16	1.11
		5	1.57	1.00	1.00	1.90	1.00	1.00	1.94	1.00	0.99
		10	1.06	1.00	1.00	1.00	1.00	1.00	1.41	0.99	0.98
	RF	3	3.24	1.83	1.40	2.93	1.98	1.27	3.13	1.94	1.51
		5	1.75	1.00	1.00	1.85	1.00	1.00	2.11	1.00	1.00
		10	1.01	1.00	1.00	1.21	1.00	1.00	1.40	1.00	1.00
	SVR	3	3.57	1.85	1.25	2.86	1.46	1.31	3.53	1.80	1.25
		5	1.53	1.00	0.96	1.65	1.00	0.99	1.93	1.00	1.00
		10	0.99	0.92	0.89	0.99	0.99	0.98	1.26	1.00	1.00

Table 7 shows (1) 200, 300, and 400 particles and (2) 3, 5, and 10 iterations for each model and application to observe the impact of different parameters of the particle number and the max-iteration on the performance. For a straightforward comparison, we used RF with 300 particles and 5 iterations as the basic unit to compare the prediction results (*i.e.*, execution time) between models built from distinct machine learning algorithms. From the result obtained for each model and app experiment, we observed performance improvement from 2.76 with 200 particles and 3 iterations to the basic unit to 3.57 with 300 particles and 5 iterations. With an increase in the number of particles and

iterations, there are six leading cases (*i.e.*, TeraSort with GBM and RF, WordCount with GBM and RF, and Pig with RF and SVR) going to stable solutions without improvement. Three cases (*i.e.*, TeraSort with SVR, WordCount for SVR, and Pig for GBM) showed room for improvement and TeraSort with SVR with 400 particles and 10 iterations even led to an 11% improvement.

5.5 Simulation Validation

To validate the simulation results, we used the normalized workload execution time, which was calculated as shown in Eq. (15), to compare the actual execution time with the performance of the rule of the thumb models that were tuned with respect to the Hadoop references [50-52] by our platform operation team, including five to seven system experts having at least 7 years of experience, as shown in Table A1 of Appendix.

$$\text{Normalized workload execution time} = \frac{\text{Execution time of the specific model}}{\text{Execution time of Rule of Thumb}} \tag{15}$$

For comparison, we used the rule of thumb as the basic unit (Table 8 and Fig. 11). For TeraSort and Word Count workloads, the improvement rankings are as follows: SVR, RF, and GBM. The configuration of the TeraSort application produced better improvement than the Word Count application. However, for Pig workloads, GBM produced the best results among the three models. Compared with the simulator results (Table 6), the validation ranking results are in the same order (*i.e.*, SVR, RF, and GBM) for TeraSort and Word Count. However, the results obtained using the Pig technique with RF and SVR are the opposite of the simulator and validation results.

Table 8. Validation results.

App. / Model	Normalized workload execution time		
	TeraSort	WordCount	Pig
GBM	0.57	1.21	0.78
RF	0.50	0.97	0.85
SVR	0.49	0.72	0.80
Rule of thumb	1	1	1

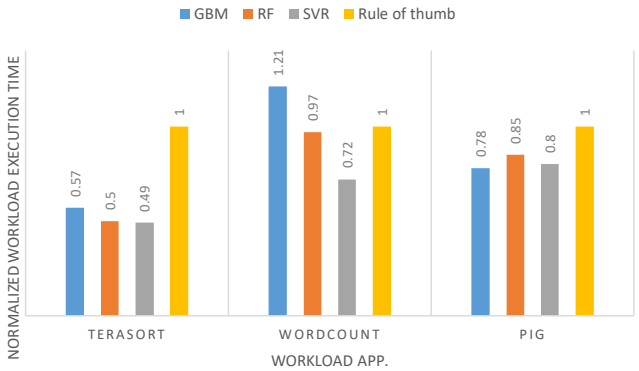

Fig. 11. Normalized workload execution times of different models and workloads.

6. RELATED WORK

Traditionally, there are two main fields of system performance optimization [12]: one is based on implementation (*i.e.*, source code or architecture design) to enhance the subsystems [53], and the other is based on the configuration of the application (*i.e.*, parameters), the hardware and the software platforms without incurring changes at the implementation level. For parameter configuration tuning, heuristic [21, 54] and machine learning-based methods [20, 32, 55] have been proposed to handle different constraints and assumptions. For virtualization environments, Chen *et al.* [56] used deep learning to predict the Hadoop configuration with analytical workloads in a cloud platform built using OpenStack. For homogeneous workloads, Jamshidi *et al.* [57] enhanced transfer learning to build performance modeling by applying a linear transformation with efficient sampling.

To show that SVR has high accuracy and efficient computation ability, Yigitbasi *et al.* [20] used five Hadoop parameters with multiple linear regression, parameter interactions and quadratic effects, artificial neural networks, model trees, and SVR. We compared our system with an auto-tuning system developed by Yigitbasi *et al.* using the same benchmark (*i.e.*, WordCount and TeraSort) and the results are shown in Table 9 and Fig. 12. Our tuned configuration achieved better improvement than the rule-of-thumb configuration and auto-tuning system. In particular, our configuration tuner achieved 28%-51% improvement over the rule-of-thumb configuration as calculated by the following formula.

$$\text{Improvement} = \left(\frac{\text{Normalized workload execution time of BigExplorer}}{\text{Normalized workload execution time of Rule of Thumb}} - 1 \right) \times 100\%. \tag{16}$$

Table 9. Normalized workload execution times of other work.

Configuration \ App	Normalized workload execution time	
	TeraSort	WordCount
BigExplorer	0.49	0.72
Auto-tuning System	0.6	0.95
Rule of Thumb	1	1

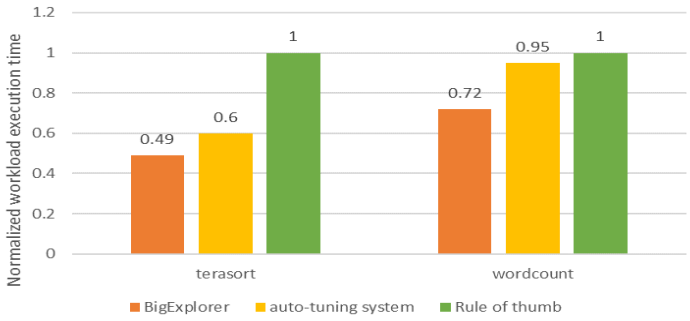

Fig. 12. Normalized workload execution times for BigExplore and other work.

7. CONCLUSIONS

Because systems have become more complicated on big data platforms, it is essential to develop means of self-improvement for such systems, *i.e.*, designing a learning component for different purposes such as system performance or job scheduling. To ensure learning by design, we proposed a configuration layer as a learning component on a widely-used big data platform (Hadoop) and built the configuration tuner to collect data, pre-process data, and obtain feedbacks on golden configurations. Based on the processed data, we used a semi-automatic feature engineering technique to provide features for data engineers and built a performance model using three different machine learning algorithms (RF, GBM, and SVR).

Our work focused on configuration parameter tuning using machine learning-based methods. Currently, there are two existing related research work to this study. Rizvandi *et al.* designed a performance predictor technique using linear regression models [17]. The model contained two parameters (*i.e.*, map tasks and reduce tasks) as features; moreover, it can be used to predict the tested application execution time with an average error of < 5%. Yigitbasi *et al.* proposed an end-to-end machine learning-based auto-tuning flow that uses five parameters selected by domain experts, and their work shows that machine learning techniques can perform better than domain experts (*i.e.*, rule-of-thumb configuration). We believe that with complicated system design, a learning component must be considered necessary for self-improvement rather than relying completely on human involvement.

APPENDIX: Table A1. Parameter default and rule-of-thumb values.

	Default value	Rule of Thumb
SplitSize	0	0
HttpThread	40	40
JVMReuse	1	1
ShuffleMergePer	0.66	0.66
ReduceSlowstart	0.05	0.8
InMemMergeThreshold	1000	1000
SortPer	0.80	0.80
ShuffleInputPer	0.70	0.70
ReduceCopyNum	5	10
SortMB	100	256
MapOutputCompress	False	True
MapTasksMax	2	2
ReduceTasksMax	2	2
SortFactor	10	64
OutputCompress	False	False
PigCacheMemusage	0.2	0.5
PigMapPartAgg	True	True
PigMinReduction	10	2
PigBytePerReducer	1000000000	1024000000
PigReducersMax	999	500
PigReduceMemusage	0.5	0.64
PigCombination	True	True
PigFileCompression	False	False
PigCodec	GZ	GZ

ACKNOWLEDGEMWNTS

This work was supported in part by the Taiwan Information Security Center (TWISC), Academia Sinica, the Ministry of Science and Technology, and the Ministry of Economic Affairs, Taiwan under the grant 107-2218-E-009-044, 107-2221-E-009-030-MY3, and 107-EC-17-A-02-S5-007 and is a partial result of Project No J36784200 conducted by Industrial Technology Research Institute and Ministry of Economic Affairs(Contract No. 108-EC-17-A-21-1516).

REFERENCES

1. D. Ormerod, "AlphaGo defeats Lee Sedol 4-1," in *Google DeepMind Challenge Match*, Go Game Guru, 2016, pp. 3-16.
2. D. Pogue, "Super Siri," *Scientific American*, Vol. 313, 2015, p. 31.
3. Apache Spark, "Lightning-fast cluster computing," http://spark.apache.org/.
4. Apache Flink, "Scalable batch and stream data processing," https://flink.apache.org/.
5. A. Candel, V. Parmar, E. LeDell, and A. Arora, "Deep learning with H2O," *H2O.ai*, 2015.
6. A. S. Rabkin, "Using program analysis to reduce misconfiguration in open source systems software," Ph.D. Thesis, Department of Computer Science, University of California, Berkeley, 2012.
7. J. Gao, "Machine learning applications for data center optimization," Google White Paper, 2014.
8. Facebook, "Sustainable data centers," https://sustainability.fb.com/innovation-for-our-world/sustainable-data-centers/.
9. W. Yang, X. Liu, L. Zhang, and L. T. Yang, "Big data real-time processing based on storm," in *Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2013, pp. 1784-1787.
10. M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, 2013, pp. 423-438.
11. E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a mongodb and hadoop platform for scientific data analysis," in *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, 2013, pp. 13-20.
12. H. H. Liu, *Software Performance and Scalability: A Quantitative Approach*, Vol. 7. John Wiley and Sons, CA, 2011.
13. K. Wang, N. Liu, I. Sadooghi, X. Yang, X. Zhou, T. Li, and I. Raicu, "Overcoming hadoop scaling limitations through distributed task execution," in *Proceedings of IEEE International Conference on Cluster Computing*, 2015, pp. 236-245.
14. V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, and B. Saha, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual ACM Symposium on Cloud Computing*, 2013, pp. 1-16.
15. B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in

Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, 2011, p. 22.
16. Y. Chen, F. Raab, and R. Katz, "From tpc-c to big data benchmarks: A functional workload model," in *Specifying Big Data Benchmarks*, 2014, Springer, pp. 28-43.
17. S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *New Frontiers in Information and Software as Services*, 2011, Springer, pp. 209-228.
18. B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010, pp. 143-154.
19. A. Yasin, Y. Ben-Asher, and A. Mendelson, "Deep-dive analysis of the data analytics workload in cloudsuite," in *Proceedings of IEEE International Symposium on Workload Characterization*, 2014, pp. 202-211.
20. N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema, "Towards machine learning-based auto-tuning of mapreduce," in *Proceedings of IEEE 21st International Symposium on Modelling*, *Analysis and Simulation of Computer and Telecommunication Systems*, 2013, pp. 11-20.
21. S. B. Joshi, "Apache hadoop performance-tuning methodologies and best practices," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, 2012, pp. 241-242.
22. D. P. Bertsekas, "Nonlinear programming," *Athena Scientific Belmont*, 1999, p. 334.
23. L. Grippo, F. Lampariello, and S. Lucidi, "A nonmonotone line search technique for Newton's method," *SIAM Journal on Numerical Analysis*, Vol. 23, 1986, pp. 707-716.
24. L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*, 2012, Springer, pp. 421-436.
25. K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, Vol. 16, John Wiley & Sons, NY, 2001.
26. D. E. Goldberg, *Genetic Algorithms*, Pearson Education, India, 2006.
27. S. Guindon and O. Gascuel, "A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood," *Systematic Biology*, Vol. 52, 2003, pp. 696-704.
28. J. Kennedy, *Particle Swarm Optimization*, *in Encyclopedia of Machine Learning*, Springer, Berlin, 2011, pp. 760-766.
29. W. Pedrycz and S.-M. Chen, *Social Networks: A Framework of Computational Intelligence*, Vol. 526, Springer, 2013.
30. L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, and C. Zheng, "Bigdatabench: A big data benchmark suite from internet services," in *Proceedings of IEEE 20th International Symposium on High Performance Computer Architecture*, 2014, pp. 488-499.
31. R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," *International Joint Conference on Artificial Intelligence*, 1995. Stanford, CA.
32. Wikipedia, "Overfitting," https://en.wikipedia.org/wiki/Overfitting.
33. I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, Vol. 3, 2003, pp. 1157-1182.
34. A. N. Tikhonov, "On the solution of ill-posed problems and the method of regularization," in *Doklady Akademii Nauk*, Russian Academy of Sciences, 1963.

35. D. L. Donoho, "For most large underdetermined systems of linear equations the minimal," *Communications on Pure and Applied Mathematics*, Vol. 59, 2006, pp. 797-829.
36. Differences between the L1-norm and the L2-norm (Least Absolute Deviations and fLeast Squares), http://www.chioka.in/differences-between-the-l1-norm-and-the-l2-norm-least-absolute-deviations-and-least-squares/.
37. L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the Trade*, 1998, Springer. p. 55-69.
38. P. Belotti, T. Berthold, and K. Neves, "Algorithms for discrete nonlinear optimization in FICO Xpress," in *Proceedings of IEEE International Conference on Sensor Array and Multichannel Signal Processing Workshop*, 2016, pp. 1-5.
39. I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Information Sciences*, Vol. 237, 2013, pp. 82-117.
40. "Particle swarm optimization (PSO) with constraint support," http://pythonhosted.org/pyswarm/.
41. L. Breiman, "Random forests," *Machine Learning*, Vol. 45, 2001, pp. 5-32.
42. J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*, Vol. 1, 2001, Springer, NY.
43. T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," arXiv preprint arXiv:1603.02754, 2016.
44. J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of Statistics*, 2001, pp. 1189-1232.
45. B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the 5th ACM Annual Workshop on Computational Learning Theory*, 1992, pp. 144-152.
46. A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, Vol. 14, 2004, pp. 199-222.
47. J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Technical Report MSR-TR-98-14, Microsoft, 1998.
48. K. J. Archer and R. V. Kimes, "Empirical characterization of random forest variable importance measures," *Computational Statistics & Data Analysis*, Vol. 52, 2008, pp. 2249-2260.
49. E. Mahmoud, "Accuracy in forecasting: A survey," *Journal of Forecasting*, Vol. 3, 1984, pp. 139-159.
50. K. Tannir, *Optimizing Hadoop for MapReduce*, Packt Publishing Ltd., UK, 2014.
51. A. Holmes, *Hadoop in Practice*, Manning Publications Co., NY, 2012.
52. T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, Inc., MA, 2012.
53. H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics," in *Proceedings of Conference on Innovative Data Systems Research*, 2011, pp. 261-272.
54. X. Ding, Y. Liu, and D. Qian, "JellyFish: Online performance tuning with adaptive configuration and elastic container in Hadoop yarn," in *Proceedings of IEEE 21st International Conference Parallel and Distributed Systems*, 2015, pp. 831-836.
55. M. A. Rahman, J. Hossen, C. Venkataseshaiah, C. Ho, K. G. Tan, A. Sultana, and F. Hossain, "A survey of machine learning techniques for self-tuning Hadoop perfor-

mance," *International Journal of Electrical and Computer Engineering*, Vol. 8, 2018, p. 1854.
56. C. C. Chen, Y. T. Hasio, C. Y. Lin, S. Lu, H. T. Lu, and J. Chou, "Using deep learning to predict and optimize Hadoop data analytic service in a cloud platform," in *Proceedings of IEEE 3rd International Conference on Big Data Intelligence and Computing*, 2017, pp. 909-916.
57. P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, and Y. Agarwal, "Transfer learning for performance modeling of configurable systems: An exploratory analysis," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, pp. 497-508.

Chao-Chun Yeh (葉肇鈞) is a Senior Engineer in Computational Intelligence Technology Center at Industrial Technology Research Institute. He received his B.S. degree in Computer Science from National Chiao Tung University in 2000, M.S. degree in Computer Science from National Tsing Hua University in 2002. He is currently Ph.D. candidate at Institute of Computer Science and Engineering, National Chiao Tung University, Taiwan. His research interests include software security, software testing and systems engineering for machine learning and big data applications.

Han-Lin Lu (呂翰霖) received the B.S. degree in the Department of Transportation Technology and Management, and M.S. degree in Computer Science and Engineering from National Chiao Tung University, Taiwan in 2010, and 2012 respectively. He is currently pursuing the Ph.D. degree at the Institute of Science in Computer Science and Engineering of National Chiao Tung University. His research interests include software quality, network security, and software security.

Jiazheng Zhou (周嘉政) is a Senior Engineer in Computational Intelligence Technology Center, Industrial Technology Research Institute. He received his B.S. degree in Computer Science from National Chengchi University in 2002, M.S. degree and Ph.D. degree in Computer Science from National Tsing Hua University in 2004 and 2011. He was a Postdoc at National Tsing Hua University from 2011 to 2015. His research interests include machine learning, big data, cloud computing, parallel and distributed computing, storage systems, interconnection networks, and high-performance computing.

Sheng-An Chang (張聖安) is a Senior Engineer in Computational Intelligence Technology Center, Industrial Technology Research Institute. He received his B.S. and M.S. degree in Engineering Science from National Cheng Kung University in 2005 and 2007. His research interests include artificial intelligence, machine learning, cloud computing, distributed computing, optimization, and middleware.

Xuan-Yi Lin (林軒毅) is a Senior Engineer in Computational Intelligence Technology Center at Industrial Technology Research Institute. He received his Ph.D. degree in Computer Science from National Tsing Hua University in 2013. His research interests include cluster systems, cloud computing, many-core platforms, parallel and distributed computing, and high-performance computing. Recently, his research focuses on systems engineering for machine learning and big data applications.

Yi-Chiao Sun (孫逸樵) is an Engineer in Computational Intelligence Technology Center at Industrial Technology Research Institute. He completed his master's degree in Computer Science from National Chiao Tung University in 2000. His research interests include network security, operating systems, parallel and distributed computing, and high-performance computing. Recently, his research focuses on systems engineering for machine learning and big data applications.

Shih-Kun Huang (黃世昆) received his B.S. (1989), M.S. (1991), and Ph.D. (1996) in Computer Science and Information Engineering from National Chiao Tung University, and he was an Assistant Research Fellow at the Institute of Information Science, Academia Sinica, between 1996 and 2004. Currently, he is the Deputy Director of the Information Technology Service Center, and jointly with the Department of Computer Science, National Chiao Tung University. Dr. Huang's research integrates software engineering and programming languages to study cyber security and software attacks.