SILHOUETTE: Efficient Cloud Configuration Exploration for Large-Scale Analytics

Yanjiao Chen, *Senior Member, IEEE*, Long Lin, *Student Member, IEEE*, Baochun Li, *Fellow, IEEE*, Qian Wang, *Senior Member, IEEE*, and Qian Zhang, *Fellow, IEEE*

Abstract—Choosing the best cloud configuration for large-scale data analytics jobs deployed in the cloud can substantially improve their performance and reduce costs. However, current cloud providers offer a wide variety of instance types and customized cluster sizes, making it both time-consuming and costly to pinpoint the optimal cloud configuration. This article presents the design, implementation, and evaluation of SILHOUETTE, a cloud configuration selection framework based on performance models for various large-scale analytics jobs with minimal training overhead. The essence of SILHOUETTE is to build performance prediction models with carefully selected small-scale experiments on small subsets of input data to estimate the performance with entire input data on larger cluster sizes. To reduce the training time and cost, SILHOUETTE incorporates new statistical techniques to select those experiments that yield the best possible information for performance prediction. Moreover, we develop a novel model transformer to convert a prediction model built on one instance type to a different instance type with only one extra experiment, which significantly reduces the training overhead. We evaluate SILHOUETTE with an extensive array of large-scale data analytics jobs on Amazon EC2. Our experimental results have shown convincing evidence that SILHOUETTE is effective in optimizing cloud configuration while saving both training time and costs compared with existing solutions.

Index Terms—Cloud configuration, large-scale data analytics, performance prediction, training overhead

1 INTRODUCTION

The last decade has seen rapid growth in large-scale data analytics jobs that are more complex than ever, such as natural language processing [1], deep learning for image processing [2], and genomic data analysis [3]. Compared with traditional workloads, these analytics jobs are typically data-intensive and computationally demanding, often requiring much longer completion times and inducing much higher costs. To execute these large-scale data analytics jobs, it has been routine to resort to the cloud to leverage its abundant computing capacity. However, it is challenging to determine the *best cloud configuration* given a specific job, and a poor choice may incur significantly higher costs to the user.

Determining the right cloud configurations is critically crucial for modern cloud service providers, such as Amazon EC2 [4], Google Compute Engine [5], and Microsoft Azure [6]. They provide users with a total of over 100 instance types with different resource configurations (i.e., CPU, memory, storage, and networking capacity) to cater to the behavior and demand of different jobs. While most cloud service providers only allow users to choose from the pool of available instance types, Google enables users to create virtual machines with customizable configurations (in terms of vCPUs and memory), making it even more challenging to choose the right cloud configurations.

The migration towards serverless cloud architectures – such as AWS Lambda [7], Google Cloud Functions [8], and Microsoft Azure Functions [9] – allows users to run their jobs as serverless functions without launching instances with pre-specified configurations. However, serverless architectures may require applications to refactor their code, and serverless cloud providers may not be interested in minimizing job completion times and incurred costs.

The choice of cloud configurations, i.e., the types and the number of instances, directly affects a job's completion time and monetary cost. A properly chosen cloud configuration can achieve the same performance goal at a much lower cost. Potential savings are even more significant for large-scale data analytics jobs due to their much longer completion times. Unfortunately, it is difficult to accurately and efficiently optimize cloud configuration given highly diversified resource requirements from different jobs. Besides, the search space is potentially huge, especially given many combinations of instance types and cluster sizes. An exhaustive search for the best cloud configuration is neither practical nor scalable in such a vast search space.

CherryPick [10] is proposed to reduce the search space of all possible cloud configurations. Given a specific job, a cost minimization model is established. A few cloud configurations are run and their execution times are fed into the

- *Yanjiao Chen is with the College of Electrical Engineering, Zhejiang University, Hangzhou, Zhejiang 310007, China. E-mail: chenyanjiao@zju.edu.cn.*
- *Long Lin is with the School of Computer Science, Wuhan University, Wuhan, Hubei 430072, China. E-mail: william_lin@whu.edu.cn.*
- *Baochun Li is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada. E-mail: bli@ece.toronto.edu.*
- *Qian Wang is with the School of Cyber Science and Engineering, Wuhan University, Wuhan, Hubei 430072, China. E-mail: qianwang@whu.edu.cn.*
- *Qian Zhang is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China. E-mail: qianzh@cse.ust.hk.*

Manuscript received 19 June 2019; revised 29 Nov. 2020; accepted 1 Feb. 2021. Date of publication 9 Feb. 2021; date of current version 23 Feb. 2021. (Corresponding author: Yanjiao Chen.) Recommended for acceptance by S. K Prasad. Digital Object Identifier no. 10.1109/TPDS.2021.3058165

optimization model, based on which the next cloud configuration to run is selected according to Bayesian optimization. The iterative process helps distinguish the near-optimal configuration from the rest with a few runs. However, CherryPick can only meet the goal of cost minimization but not other objectives, such as minimizing the job completion time with a cost budget. Alternatively, *model-based* mechanisms build prediction models to estimate the execution time under different cloud configurations for different jobs [11], [12], [13]. The prediction models are more expressive and can be used to choose cloud configurations that serve different purposes, e.g., cheapest or fastest. The most recent attempts include Ernest [11], PARIS [12] and Arrow [14]. However, Ernest only optimizes the number of instances given a specific instance type but does not consider different types of instances. PARIS and Arrow focus on deriving the best instance type across multiple public clouds but not optimizing the number of instances needed.

In this paper, we present the algorithm design and implementation of SILHOUETTE, a new algorithmic framework proposed to address the problem of cloud configuration selection for large-scale data analytics jobs. Specifically, given the type and the workload (size of the input data) of a specific data analytics job, SILHOUETTE predicts the execution time under different cloud configurations, based on which the type of instances and the number of instances can be optimized according to the client-defined policy. SILHOUETTE helps users achieve a balance between cost and completion time in cloud computing environments. It can achieve a high prediction accuracy with a low search overhead.

In SILHOUETTE, we first collect the training data to build the base runtime prediction model for a specific instance type. To reduce the number of experiments needed for training data collection, we propose new statistical techniques to sort out the most informative experiments for performance estimation so that we can achieve high accuracy with only a small number of tests. We then establish the prediction model based on a comprehensive analysis of both computation and communication patterns in typical data analytics jobs. The computation time generally increases quasi-linearly with the size of input data, and the communication time usually depends on the cluster size and the execution framework of the job. We leverage mutual information to shape the prediction model in a way that best reflects the computation and communication pattern of the given job.

To further reduce training overhead, rather than repeating the experiments on every instance type, we propose to transform the prediction model of one instance type to another with only one extra experiment. Then, SILHOUETTE integrates prediction models for all instance types to choose the best cloud configuration that fulfills a user-specified goal, e.g., minimizing cost with a completion time deadline or minimizing the completion time with a cost budget.

We have implemented SILHOUETTE on Amazon EC2 and evaluate its performance using six representative large-scale data analytics jobs, including TeraSort and WordCount in Hadoop, three machine learning tasks in SparkML, and TPC-DS in Spark SQL. Compared to existing solutions including Ernest [11], CherryPick [10] and coordinate descent search, SILHOUETTE can further reduce the training time and the training cost by up to 66 percent. Our experimental results have also verified the robustness of SILHOUETTE for various jobs, cloud configurations, and input dataset sizes. In summary, our work makes the following contributions:

- We propose SILHOUETTE, a cloud configuration selection system for large-scale data analytics jobs with high accuracy and low overhead.
- We build performance prediction models by studying the computation and communication patterns of large-scale data analytics jobs in a cluster computing environment with a minimal training overhead.
- We conduct extensive experiments to confirm the accuracy, cost-effectiveness, and robustness of SILHOUETTE.

The rest of the paper is structured as follows. We first present the principles that guide the design of SILHOUETTE in Section 2. Then, we give an overview of SILHOUETTE and introduce specific design decisions in Section 3. We perform extensive evaluations of SILHOUETTE using six representative large-scale data analytics jobs in Section 4. We review related works in Section 5. We discuss operational challenges and provide a road map for future works in Section 6. Finally, we summarize the work in Section 7.

2 BACKGROUND AND MOTIVATION

In this section, we discuss the challenges of finding the best cloud configurations for large-scale data analytics jobs.

Complex Cost Models. Cloud platforms provide various ways for users to pay for cloud resources. For example, there are four ways to pay for Amazon EC2 instances: on-demand, reserved instances, spot instances, and dedicated hosts. In this paper, we focus on the most commonly-used on-demand instances, which charge users by the instance type and the execution time. Users can configure the cloud with different types and numbers of instances for different jobs. In general, an instance with more CPU, memory, and storage has a higher unit price per time unit. For users, choosing a more powerful configuration (regarding the number and the type of instances) may accelerate the computation but raise the unit price per time unit. In contrast, using a less powerful configuration saves costs per time unit but may lead to longer execution times.

Fig. 1 shows the runtime and the total cost for a SparkML [15] classification algorithm (the details of the job are described in Section 4) with one instance of 8 different types. The cheapest instance `m4.large` naturally yields the longest completion time, yet the total cost is not necessarily the lowest. Fig. 1 shows that `r3.2xlarge` has both longer completion time and a higher total cost than `c4.2xlarge` even though the former has more memory. This observation indicates that choosing the *right* cloud configuration is quite complicated and challenging, even more so for large-scale analytics jobs whose execution times are much longer and consume more resources.

Non-Linear Relationship. The key to finding the best cloud configuration lies in how the complex relationship between the execution time and the cloud configuration, including the instance type and the number of instances, can be discovered and profiled. The cloud provider offers various

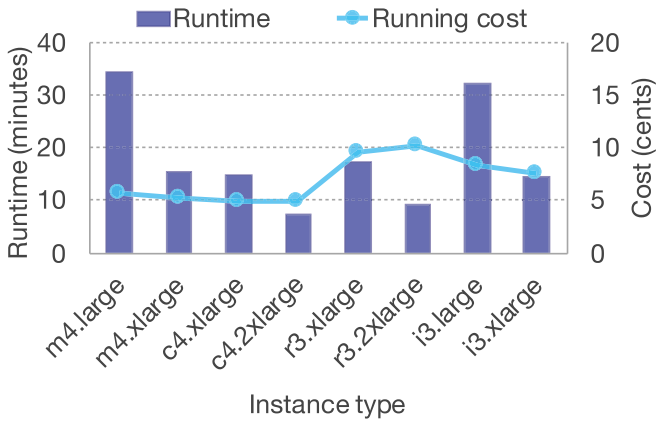

Fig. 1. Execution times and total costs of a machine learning job with EC2, across various instance types.

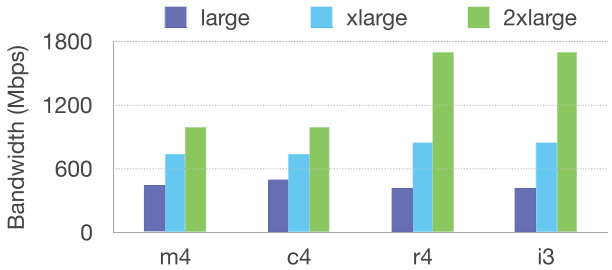

Fig. 2. Maximum bandwidth capacities across different instance families.

instances for different purposes, e.g., compute/memory/storage-optimized or general-purpose instances. As shown in Fig. 2, the available maximum bandwidth capacities are different across instance types. For `m4` and `c4` families of instances, we find that the instance type with fewer cores has more bandwidth available per core, which will directly affect the end-to-end completion times of different jobs.

Large-scale analytics jobs are computationally demanding and sensitive to the number of available cores, memory, and storage. Their completion times usually have a non-linear relationship to the quantity of resources in the cloud configuration. Consider three large-scale analytics jobs: (a) `CTR`, which predicts click-through-rate for the display advertising challenge (Dac) dataset[1] [16] using an implementation of SparkML, (b) `TeraSort` [17], a benchmark sorting job for big data analytics on Hadoop [18] and (c) `TPC-DS`, a decision support benchmark [19]. Fig. 3a illustrates the completion times of `CTR` and `TeraSort` using different numbers of instances of the same type, i.e., `m4.large`. It is natural that both applications' completion times are shorter with more instances, but the rate of decrease is slower with more available resources. In addition, Fig. 3b shows significant runtime gaps when `TPC-DS` uses different instance types, mainly due to different core numbers and CPU/RAM ratios (e.g., `c4.2xlarge` delivers the best performance thanks to more cores and a higher CPU/RAM ratio).

Job Diversity. There is a variety of large-scale analytics jobs with different resource requirements, e.g., computationally demanding or memory demanding, which should be considered when making cloud configuration choices. Even with the same computing capacity, the total costs will be different under different cloud configurations for different jobs.

Fig. 3c shows the total costs of `CTR` and `TeraSort` with a fixed number of cores, but different types and numbers of instances from the `m4`-family, e.g., four `m4.2xlarge` instances each with 8 cores or two `m4.4xlarge` instances each with 16 cores. We can observe that for `TeraSort`, the 4-node `m4.2xlarge` cluster has the lowest total cost, while `CTR` benefits the most from single-node `m4.4xlarge` cluster. There is usually a tradeoff between the runtime and the budget for `TPC-DS` as shown in Figs. 3b and 3d. Quantifying such tradeoff is also a crucial design requirement in SILHOUETTE. Moreover, the best cloud configuration depends on the application configuration, such as the number of map and reduce tasks. Our work on choosing the best *cloud* configuration is complementary to existing works on identifying the best *application* configurations [20], [21], i.e., SILHOUETTE can work with any application configuration.

3 SILHOUETTE: DESIGN DETAILS

In this section, we present the detailed design of SILHOUETTE, a data-driven approach for optimizing cloud configuration, with the objective of using minimal training data to achieve a balanced tradeoff between performance and overhead.

3.1 Overview

The design rationale of SILHOUETTE is a three-step process: (1) gather information from small-scale selected training experiments on one instance type; (2) build a base prediction model based on the computation and communication patterns of the job, then transform the base prediction model to accommodate all other candidate instance types; (3) adopt a configuration selector to optimize cloud configuration.

Given a specific job and workload (size of the input data), SILHOUETTE enables the user to optimize the cloud configuration by predicting the performance and cost of candidate cloud configurations. Fig. 4 shows the architecture of SILHOUETTE, consisting of three major steps.

- *Training data collector*. The training data collector first determines which training experiments to conduct. The training experiments run the analytics job with small subsets of the input data on a limited number of instances of a specific type. We develop a selection approach to reduce the number of experiments to run while obtaining enough training data for establishing the execution time prediction model. The training data collector then runs the job on selected experiments to obtain the execution time samples.
- *Model constructor*. The model constructor models the relationship between the execution time and the cloud configuration in two steps. The model builder first builds a base prediction model fitted using the training data of one instance type obtained from the training data collector. Then the model transformer transforms the base prediction model to prediction models for other instance types.

1. We use a subset of the Dac dataset to make the timescale of `CTR` comparable to that of `Terasort`.

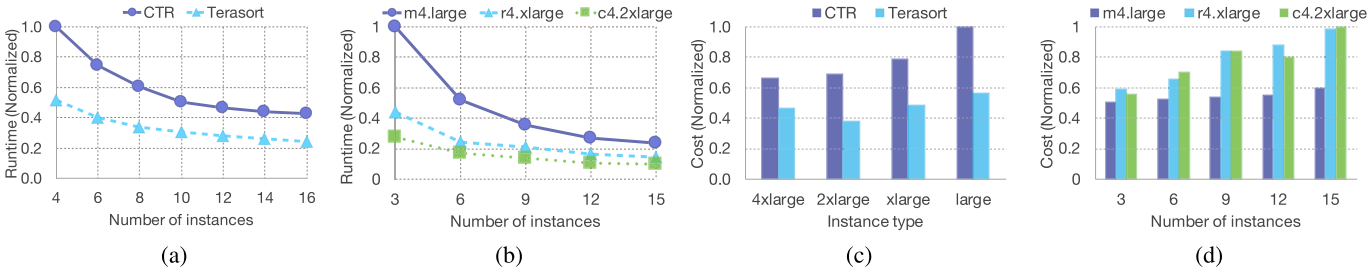

Fig. 3. (a) Execution times of `CTR` and `Terasort` with different numbers of `m4.large` instances. (b) Execution times of `TPC-DS` with different numbers of `m4.large, r4.xlarge and c4.2xlarge` instances. (c) Costs of `CTR` and `Terasort` with varying resources of `m4` family instances. (d) Costs of `TPC-DS` with different numbers of `m4.large, r4.xlarge and c4.2xlarge` instances.

- *Selector builder*. With prediction models for all candidate instance types in the execution time predictor pool, the selector builder can obtain the estimated execution time and cost for a specific job with a specific configuration. Then, according to the user-defined policy, the selector builder optimizes the cloud configuration using grid enumeration. In this way, the user can obtain the most-preferred cloud configuration that yields the shortest execution time or minimum cost.

3.2 Training Data Collector

The training data collector conducts experiments on a specific instance type with only small fractions of the input data, which helps predict the performance of job execution on the entire input data. The training data collection consists of experiment selection and experiment execution.

Experiment Selection. There are two important parameters to determine for experiment selection: (1) *scale*, i.e., the fraction of input data; and (2) the *number of instances* used in the job execution. Suppose the scale is constrained in $1\% \sim 10\%$ of the entire input data, and the number of instances is limited to be smaller than 10, then there are 100 experiment settings, i.e., (scale, number) pairs, which will induce a high overhead. Therefore, we propose statistical techniques to run fewer experiments while maintaining high prediction accuracy. The key idea is to select the experiments that yield the most information for predicting the performance of cloud configurations.

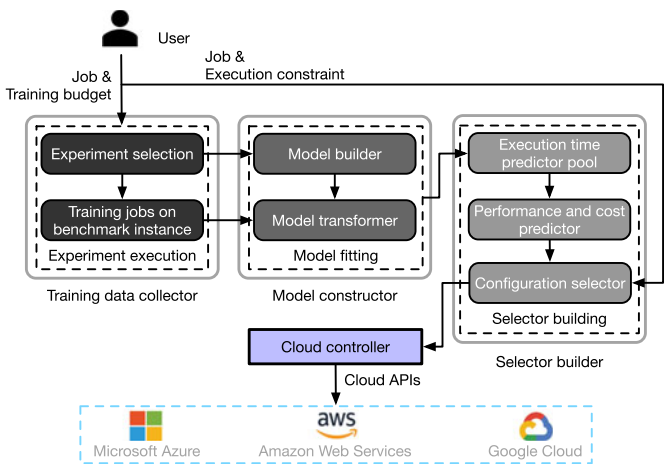

Fig. 4. Architecture of SILHOUETTE.

We use $E_i = (x_i, y_i)$ to denote an experiment setting, where x_i is the number of instances and y_i is the scale. Let M denote the total number of experiment settings obtained by enumerating all possible scales and numbers of instances. Then, we obtain the K-dimensional feature vector $F_i = (f_1(x_i, y_i), f_2(x_i, y_i), \ldots, f_k(x_i, y_i))$, in which each entry corresponds to one term in the prediction model. The prediction model is built to predict the execution time $T_{base}(x_i, y_i)$ needed to run experiment E_i. More specifically, we choose a subset of all experiment settings to run to obtain the training data for establishing the prediction model that forecasts the execution time of the remaining experiment settings. We use linear fitting to obtain the prediction model. Linear fitting is chosen due to its simplicity and good performance, as shown in the experiments.

$$T(x_i, y_i) = \sum_{k=1}^{K} w_k \beta_k f_k(x_i, y_i), \tag{1}$$

where w_k is the fitted coefficient, $\beta_k \in \{0, 1\}$ indicates whether feature f_k is chosen or not. $\beta_k = 1$ means the feature is chosen, and $\beta_k = 0$ means the feature is not chosen. The form of the feature function $f_k(x, y), k \in [1, K]$ are constructed based on the analysis of the execution time of large-scale analytics jobs.

Large-scale analytics jobs are numerically intensive and inherently iterative. More specifically, large-scale analytics jobs often run in a succession of supersteps (i.e., iterations) until a termination condition is satisfied. Each superstep is mainly composed of two phases: 1) concurrent computation and 2) communication. The computation time is related to the workload and the cloud configuration [11], and there is only a few common communication patterns in large-scale analytics jobs [22], [23], [24]. Therefore, we can dissect the execution time $T(x_i, y_i)$ under experiment setting E_i into the computation time and the communication time. Our main objective is to customize the prediction function for a given job by characterizing the computation and the communication patterns of the job and design feature functions $f_k(x, y)$ that can yield a suitable fitting. In summary, we construct the feature vector as

$$F_i = (y_i/x_i, \sqrt{y_i}/x_i, 1, x_i, \log x_i, x_i^2), \tag{2}$$

in which the first three elements correspond to typical computation time patterns, and the rest elements correspond to typical communication time patterns. A detailed analysis of different computation and communication patterns of typical

large-scale analytics jobs can be found in [11]. Note that users can choose certain elements in F_i by setting β_k as 0 or 1, and can add more elements in F_i based on specific analytics jobs.

Experiment selection aims to obtain as much information as possible with as few experiments as possible. Therefore, it is desirable to choose experiment settings that can yield divergent information. According to the optimal design theory [25], the coefficients in Equation (1) can be estimated by experimental designs that optimize certain statistical criteria. We adopt *D-optimality*, a popular criterion that maximizes the determinant of the information matrix, i.e., the weighted sum of the covariance matrix $F_i^T F_i$. In this way, we can obtain the highest differential Shannon information, a measure of average surprisal of the random variables of coefficients $w_k, k \in [1, K]$.

$$\max_{\alpha_i, i \in [1,M]} \left| \sum_{i=1}^{M} \alpha_i F_i^T F_i \right|, \tag{3}$$

$$\text{subject to } 0 \leq \alpha_i \leq 1, i \in [1, M], \tag{4}$$

$$\sum_{i=1}^{M} \alpha_i y_i / x_i \leq B, \tag{5}$$

where α_i represents the probability that experiment setting E_i is selected. The larger the α_i, the more likely that E_i is selected. The total cost of the selected experiments is constrained by the budget constraint B. More specifically, y_i/x_i is the cost for running experiment E_i according to the cloud pricing model (i.e., the cost is associated with the workload and the number of instances). Equation (3) is a convex optimization problem, and we leverage CVXPY,[2] a Python-embedded solver for convex optimization problems. The output of the above optimization problem is $\alpha_i, i \in [1, M]$, i.e., the probability that E_i is selected. We sort α_i and choose the first m experiment settings with the highest probability α_i. The value of m is determined by the user or the administrator considering a tradeoff of training overhead and the performance. In our experiments, we choose $m = 10$ out of a total of $M = 100$ candidate experiment settings.

Experiment Execution. Given a selected experimental setting $E_i = (x_i, y_i)$, we need to decide which data samples from the entire input dataset should be included to form the experiment dataset to satisfy the specified scale y_i. We use random sampling to select data samples from the entire input dataset, as random sampling can avoid getting stuck in isolated regions of the dataset. We first randomly pick a starting seed sample from the input dataset. Then, at each sampling step, an outgoing sample is picked uniformly at random. The process continues until the number of selected samples satisfies the scale requirement. Having obtained the small datasets and deployed the specified number of instances, we run the job with the selected experiment settings to get the execution time samples as training data to build the prediction models.

2. https://www.cvxpy.org/

3.3 Model Constructor

The model constructor consists of the *model builder* and the *model transformer*. The model builder establishes the base prediction model, given the collected execution time training data for a specific instance type from the training data collector. On the other hand, the model transformer derives the prediction models for all other instance types according to the base prediction model.

Model Builder. Given all the candidate feature functions in F_i, users can select the ones that are most predictive of the execution time T_i. We also provide a way of feature selection to help trim F_i. We use mutual information [26], [27], [28] as the criterion to sift good predictors. Given the collected m training execution time data samples under different numbers of instances and different data scales, we first compute the K-dimensional feature vector $F_i = (f_1(x_i, y_i), f_2(x_i, y_i), \ldots, f_k(x_i, y_i))$ for each experiment setting. Then, we calculate the mutual information between each feature function $f_k(x_i, y_i), i \in [1, m]$ and the execution time $E_i, i \in [1, m]$, and choose the f_k whose mutual information with the execution time is higher than a threshold.

The base prediction model we are fitting based on the m training execution time samples tries to learn values of w_k in the following equation.

$$T_{base}(x, y) = \sum_{k=1}^{K} \beta_k w_k f_k(x, y), \tag{6}$$

$$\beta_k \in \{0, 1\}, \tag{7}$$

$$w_k \geq 0. \tag{8}$$

where β_k is determined by the user or the mutual information. We use a non-negative least squares (NNLS) solver to find the model that best fits the training data, which can avoid over-fitting and corner cases, e.g., the execution time becomes negative as the number of instances increases infinitely.

Model Transformer. The cloud provider usually provides various families of instances with different combinations of CPU, memory, storage, and networking capacity to meet different jobs' needs, e.g., general-purpose and compute/memory/storage-optimized. Through extensive experiments, we find that given a specific job and a fixed dataset scale, the execution time of two different instance types has a certain relationship when scaling the cluster size. These observations motivate us to design a transformer in SILHOUETTE, which helps convert prediction models between different instance types according to simple mapping. Therefore, we do not need to run experiments on every instance type to acquire the training data to build prediction models, which greatly reduces the training time and training costs.

One way to construct such transformers is to find the relationship between execution time and instance hardware parameters, e.g., bandwidth, and then derive the mapper based on hardware parameters for different instance types. Nevertheless, such a method is complicated to design and implement due to the vast number of instance types and hardware parameters. In this paper, we adopt a simple yet effective mapping approach. We use the transformer Φ to represent a mapping from the base prediction model built

for one instance type to the target prediction model for another instance type $\Phi : T_{base}(x, y) \rightarrow T_{target}(x, y)$. Through extensive experiments to compare execution times on different instance types for the same job and the same dataset scale, we find that the category of feature functions in prediction functions are similar, i.e., the values of β_k for $T_{target}(x, y)$ can inherit from $T_{base}(x, y)$. In other words, if f_k is included in $T_{base}(x, y)$, it is most likely that $T_{target}(x, y)$ should also contain f_k. This is mainly because the computation and the communication patterns of the job remain unchanged under the same job and the same number of instances. However, the weight of each feature function w_k will be different under different instance types, and we can focus on the mapping of weights from the base prediction model to the target prediction model. Besides, only the weight of the feature functions that contain x should be changed since y remains unchanged among different prediction models.

First, given a selected experiment setting $E_i = (x_i, y_i)$ that has run on the base instance type, we have already known its execution time as $T_{base}(x_i, y_i)$. Second, we run the experiment E_i on the target instance type to get the execution time $T_{target}(x_i, y_i)$. Hence, we can calculate the ratio of the two execution times as $\gamma^{E_i} = T_{target}(x_i, y_i)/T_{base}(x_i, y_i)$. Finally, we reconstruct the weight w_k of each term containing x, and the model transformer derives the prediction model for the target instance type as

$$T_{target}(x, y) = \sum_{k=1}^{K} \beta_k w'_k f_k(x, y), \tag{9}$$

where $w'_k = w_k$ if feature $f_k(x, y)$ is not a function of x, and $w'_k = w_k / f_k(\gamma^{E_i}, 1)$ if feature $f_k(x, y)$ is a function of x. For example, the target weight for the feature y/x is $w_k \times \gamma^{E_i}$, and the target weight for the feature $y/\sqrt{x}$ is $w_k \times \sqrt{\gamma^{E_i}}$.

We can also improve the mapping accuracy with n experiments where $1 \leq n < m$. More specially, we select the first n experiment settings with the highest α_i as obtained from Equation (3). Note that the values of α_i are the same for different instance types. We obtain the ratio of execution time for experiment E_i with probability α_i as γ^{E_i}. The fitted weight of the prediction model for the target instance type is

$$w'_k = w_k / f_k(\overline{\gamma}, 1), \tag{10}$$

where the weighted mean $\overline{\gamma}$ of γ^{E_i} is

$$\overline{\gamma} = \frac{\sum_{i=1}^{n} \alpha_i \gamma^{E_i}}{\sum_{i=1}^{n} \alpha_i}.$$

3.4 Selector Builder

After model construction, the predictor for the given job and a certain instance type is a 3-tuple $(job, instance\ \tau, model\ T_\tau\ (x, y))$. All predictors will be put into a predictor pool, within which there is a variety of predictors for different jobs and instance types.

Given a job, all the related execution time prediction models can be integrated into a single execution time predictor $T(\boldsymbol{x}, y)$, in which $\boldsymbol{x}$ is the cloud configuration vector consisting of both the type τ and the number of instances x as (τ, x). For a given input dataset of the job, we aim at finding the most preferred cloud configuration that satisfies specific execution time and cost constraints. Let $P(\boldsymbol{x})$ be the unit price per time unit of the cloud configuration $\boldsymbol{x}$, i.e., the unit price of the instance type τ times the number of instances. The cloud configuration optimization problem can be formulated as

$$\boldsymbol{x}^* = S(T(\boldsymbol{x}, y), C(\boldsymbol{x}, y), R(\boldsymbol{x}, y)), \tag{11}$$

where

$$C(\boldsymbol{x}, y) = P(\boldsymbol{x}) \times T(\boldsymbol{x}, y), \tag{12}$$

$$0 \leq y \leq 1. \tag{13}$$

in which the total cost $C(\boldsymbol{x})$ required for finishing the job with a cloud configuration is the unit price multiplies the execution time, $R(\boldsymbol{x}, y)$ is the constraints enforced by the user, e.g., the maximum tolerated execution time or the maximum tolerated cost, and the selector $S(*)$ is determined by the user to optimize the cloud configuration $\boldsymbol{x}^*$ that achieves the desirable performance-cost tradeoff.

SILHOUETTE allows users to implement a high-level policy of cloud configuration selection. A policy can be to choose a cloud configuration for a certain job that: (a) induces a minimum cost with a specified execution time constraint, or (b) requires the shortest execution time in the worst case. SILHOUETTE can also be used to check whether a job can be finished on time with a specific cloud configuration. An example of policy (a) is to minimize the total cost while ensuring that the execution time is within a deadline. We can translate this policy as the following optimization problem

$$\min_{(\tau, x)} \ C(\tau, x) = P(\tau, x) \times T(\tau, x, 1), \tag{14}$$

$$\text{subject to } T(\tau, x, 1) \leq \overline{T}, \tag{15}$$

$$T(\tau, x, 1) = T_\tau(x, 1) = \sum_{k=1}^{K} \beta_k^\tau w_k^\tau f_k(x, 1). \tag{16}$$

Given the above objective function and constraints, it is possible to conduct an exhaustive search or adopt heuristic algorithms. $T(\tau, x, 1)$ first decreases, then increases with x. This is because initially, with more instances, the computation time drops quickly. However, as the number of instances becomes too large, the communication time increases dramatically. Therefore, we can find the maximum $x_{\max}$ that satisfies $T(\tau, x, 1) \leq \overline{T}$. We can then enumerate all possible experiment settings with $x \in [1, x_{\max}]$ for the given jobs. Most notably, this means that SILHOUETTE can check whether each candidate satisfies the constraints and select the cheapest one without any experiment execution.

Given the chosen cloud configuration, we deploy the cluster in the cloud environment with existing images through our cloud controller using the cloud's APIs. Furthermore, the cloud controller is also used as a training data collector. Once a job is finished with a specific dataset scale y_i and cluster size x_i, the execution time t^{E_i} can be added as the training data or the data for rebuilding prediction models with model builder or model transformer illustrated in Section 3.3.

TABLE 1
Configurations for Different Instance Types

Instance	vCPU	Memory (GiB)	Storage (GB)	Dedicated EBS Bandwidth (Mbps)	Network Performance (Gbps)	Price
m4.large	2	8	EBS-only	450	moderate	$0.1 per hour
c5.large	2	4	EBS-only	up to 4,750	up to 10	$0.085 per hour
r4.large	2	15.25	EBS-only	n/a	up to 10	$0.133 per hour
i3.large	2	15.25	1 x 475 NVMe SSD	n/a	up to 10	$0.156 per hour

4 Evaluation

This section evaluates the performance of Silhouette by answering three key questions: 1) Does the prediction model built by Silhouette achieve high accuracy for heterogeneous large-scale analytics jobs? 2) Does the experiments for collecting the training data in Silhouette introduce a low overhead? 3) Is Silhouette robust for a variety of analytics jobs and different sizes of input data?

4.1 Experimental Setup

Large-Scale Data Analytics Jobs. We evaluate Silhouette with five large-scale analytics jobs, namely, (1) Classification, (2) Regression, (3) Clustering, (4) TeraSort, and (5) WordCount, on two data-parallel processing engines, Spark [29] and Hadoop [18]. The details of data analytics jobs are as follows.

- The 3 machine learning jobs, i.e., classification, regression, and clustering, are part of SparkML [15]. For classification, we use samples with 44,000 features from rcv1 [30], a collection of text categorization benchmark jobs. For regression and clustering, we use 1 million synthetic samples with 44,000 features. We set up the TPC-DS [19] benchmark with SparkSQL [31]. The TPC-DS has a set of standard decision support queries based on those used by industry users. We evaluate Silhouette on TPC-DS with 99 queries, where the TPC-DS benchmark has relevant data for various tables.
- For Hadoop, we use TeraSort [17], a common benchmarking application for large-scale data analytics that requires a balance between I/O bandwidth and CPU speed. TeraSort sorts a set of randomly-generated records. We run TeraSort on Hadoop with 200 million samples, which is large enough to exercise on the cloud. We also use WordCount [32], which computes the occurrence frequency of each word in English literature. WordCount leverages identity map and identity reduce functions as the MapReduce framework to perform the counting. We run WordCount to count words in 55 million entries from Wikipedia articles.

Cloud Configuration. To test the effectiveness of Silhouette for different types of instances, we choose m4.large (general purpose), c5.large (compute-optimized), r4.large (memory-optimized), and i3.large (storage-optimized) from Amazon EC2. All instances are located in the US West (Oregon). Each instance runs the Linux system with default hardware (vCPU, memory, storage and network performance as in https://aws.amazon.com/ec2/instance-types/). Table 1 lists the hardware configurations and prices for each type of instances.

For training the data collector, we choose m4.large as the base instance type since it has the lowest unit price. For experiment selection, we consider that the number of instances x_i is within [1,10], the fraction of input data is within [1%, 10%] with 1 percent apart. Thus, there is a total of $M = 100$ candidate experiment settings. For simplicity, we set $\beta_k = 1, k \in [1, K]$ for all jobs. The budget B is set as 1. After solving problem (3) to get the probabilities $\alpha_i, i \in [1, 100]$, we select $m = 10$ experiment settings with the highest α_i to run to collect the training data.

For the model constructor, we fit the training data using the non-negative least squares solver to establish the base prediction model. To transform the base prediction model to prediction models for other instance types, we run three extra experiments for model transformation.

For the selector builder, we use the on-demand pricing of Amazon EC2 published in June 2018 as our cost model, as shown in Table 1. The policy we adopt is to minimize the total cost subject to the deadline constraint, the same as in Equation (14).

Our experiments are run with Apache Spark 2.2 and Hadoop 2.8. Our predicted execution times are compared against at least ten actual runs of execution times, and our figures show the mean value with error bars indicating the standard deviation.

Our experiments are selected according to those in CherryPick [10] to demonstrate the effectiveness of Silhouette. However, we did not run real-world applications that are more complicated. We have made our code open-source and expect that more researchers can test Silhouette with real-world applications in the future.

Our code is made available at https://github.com/williamlinl/Silhouette.

4.2 Effectiveness

We compare Silhouette with Ernest and CherryPick to show its efficiency and effectiveness. First, we compare Silhouette and Ernest in terms of prediction accuracy and training overhead. Then, we compare Silhouette and CherryPick in terms of training overhead. CherryPick does not establish an execution time prediction model, but only a cost minimization model. Therefore, the comparison with CherryPick does not include the execution time prediction accuracy. The implementation of Ernest and CherryPick is according to the open-source projects at https://github.com/amplab/ernest and https://github.com/harvard-cns/cherrypick/tree/master/spearmint/examples/cherrypick, respectively.

Comparison With Ernest. In Fig. 5, we compare the performance of Silhouette, Ernest and coordinate descent search,

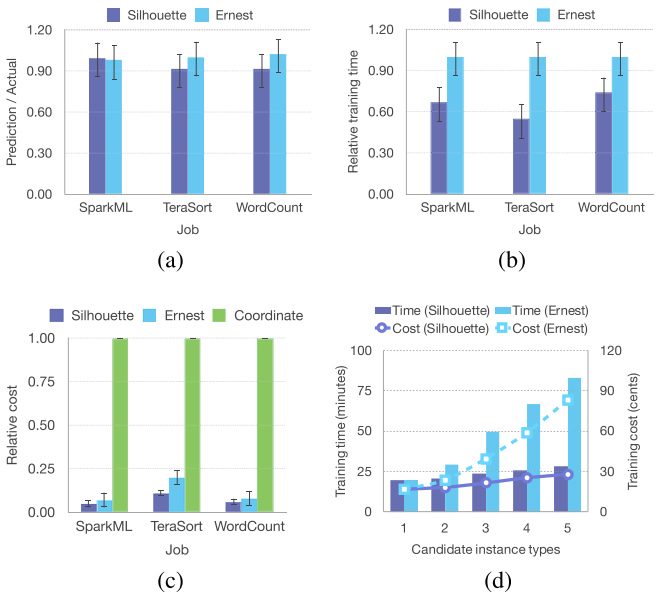

Fig. 5. Performance comparison of SILHOUETTE with the baseline Ernest (a) prediction accuracy, (b) training time, (c) overhead, and (d) scalability.

in which coordinate descent search seeks for the optimal configuration from a randomly chosen starting point (instance type, cluster size) and searches one dimension at a time: first along the dimension of RAM/CPU ratio, then the cluster size. Fig. 5a shows that the ratios of the predicted execution time to the actual execution time of SILHOUETTE for SparkML, TeraSort and WordCount are 0.994, 0.914 and 0.916 (the prediction accuracies are 99.4, 91.4 and 91.6 percent), and those of Ernest for SparkML, TeraSort and WordCount are 0.979, 1.001, 1.024 (the prediction accuracies are 97.9 percent, 99.9 and 97.6 percent). The prediction accuracy of SparkML is comparable with that of Ernest. The training time and the training cost of SILHOUETTE are much shorter than those of Ernest. As shown in Figs. 5b and 5c, SILHOUETTE can save up to 25 percent runtime and 30 percent cost compared with Ernest, and using coordinate descent search requires much more time and cost compared with SILHOUETTE. We can see from Fig. 5d that the training time and the training cost of SILHOUETTE raise much slower than those of Ernest when there are more candidate instance types. When there are 5 candidate instance types, the training time of SILHOUETTE and Ernest is 25 minutes and 83 minutes, respectively, which shows that SILHOUETTE can slash the training overhead by as much as 66 percent.

Comparison With CherryPick. We compare SILHOUETTE against CherryPick using TPC-DS. Unlike a traditional

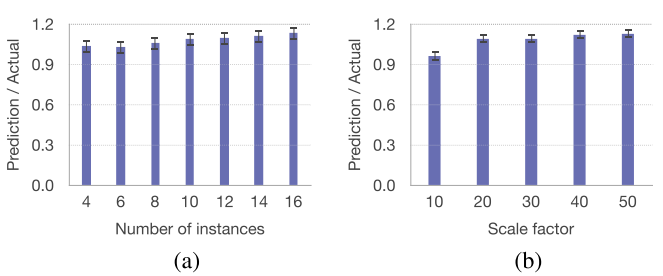

Fig. 6. Prediction accuracy using SILHOUETTE for TPC-DS with (a) different cluster sizes (scale factor = 20) and (b) different scale factors (cluster size = 10).

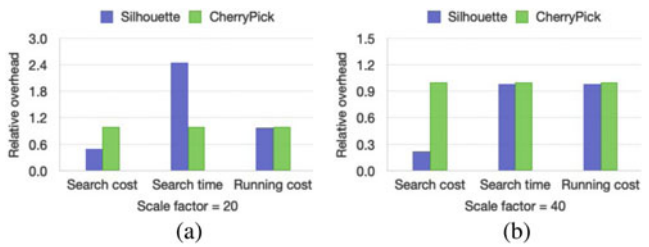

Fig. 7. Comparing SILHOUETTE with CherryPick with different scale factors (TPC-DS).

dataset with a constant size, TPC-DS can run with different raw data sizes produced by the data generator based on a set of discrete scaling points (scale factors). Fig. 6 presents the prediction accuracy of SILHOUETTE versus cluster size and scale factor. It is shown that SILHOUETTE can achieve a high prediction accuracy (the lowest prediction accuracy is 97.5 percent) and is stable with scaling dataset size.

To implement CherryPick, we use `large`, `xlarge`, and `2xlarge` instance types within each family as listed in Table 1, and change the total number of cores from 32 to 112, thus we have a total of 66 configurations [10].

As we use a 20 scale factor database for the job, both SILHOUETTE and CherryPick pick the near-optimal configuration that minimizes the running cost. For search time, as shown in Fig. 7a, SILHOUETTE spends a longer time on model training. However, CherryPick has a higher search cost because it uses larger cluster sizes and the whole dataset during search, which is more expensive. In comparison, SILHOUETTE always uses smaller and cheaper clusters. Although CherryPick identifies the near-optimal configuration for a specific scale factor, it is sensitive to the variation of input workload. As shown in Fig. 7b, when we change the scale factor to 40, CherryPick needs to rerun to build a new model for the new job while SILHOUETTE can use the previous model to select a new cloud configuration without re-train. In this way, SILHOUETTE saves search time and cost. Since input workloads usually vary in practice, SILHOUETTE can always maintain a satisfactory performance while CherryPick should be rerun to adapt to different workloads. Furthermore, CherryPick may not be suitable for meeting other objectives, such as minimizing job completion time with a cost budget, but SILHOUETTE can deal with different goals.

4.3 Prediction Accuracy

We evaluate the prediction accuracy of the five jobs where the base and transformed execution time prediction models are built for `m4.large` and `c5.large`, respectively. In this experiment, we configure SILHOUETTE to use $1 \sim 8$ instances and $1\% \sim 8\%$ of the input data for training data collection. We then predict the execution time for cases where the SparkML algorithms use the entire input data on 4, 8, and 16 instances, and the Hadoop applications use the entire input data on 16 and 32 instances. To show how close the prediction is to the actual job completion time, we use the ratio of the predicted job completion time to the actual job completion time as the evaluation metric.

Figs. 8a, 8b, 8c, 8d, 8e, 8f and 8g demonstrate the prediction accuracy for jobs in SparkML. Both the base prediction

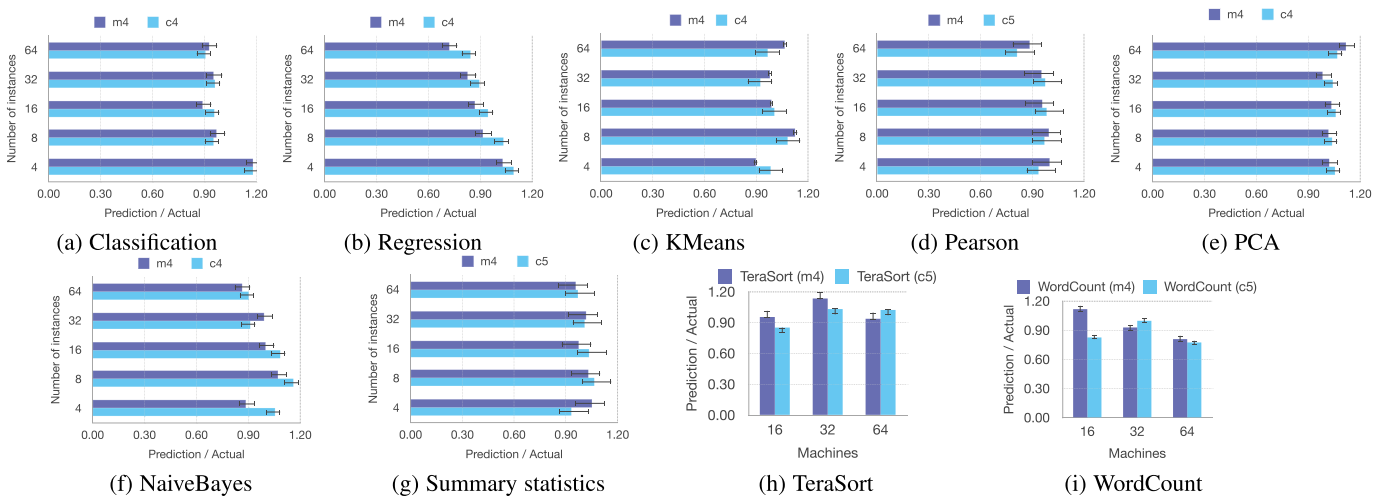

Fig. 8. Prediction accuracy using SILHOUETTE for (a) Classification. (b) Regression. (c) KMeans. (d) Pearson. (e) PCA. (f) Naive Bayes. (g) Summary statistics. (h) TeraSort. (i) WordCount.

model for `m4.large` and the transformed prediction model for `c5.large` achieve a high accuracy, which confirms the effectiveness of the model transformer in SILHOUETTE. More specifically, all predicted execution times by SILHOUETTE are within 19 percent deviation from the actual execution time. We can observe a few exceptions with a slightly higher error, which is due to the performance variations of instances in the cloud environment, which leads to discrepancies in the execution time between the test and the actual runs.

Figs. 8d and 8e shows the prediction accuracy for TeraSort and WordCount. Different from SpartML jobs that directly print the results to the terminal, these two jobs read input and write output to the Hadoop distributed file system (HDFS) with a heavy shuffle. SILHOUETTE can capture such I/O overheads in the base prediction model. It is shown that both the base prediction model for `m4.large` and the transformed prediction model for `c5.large` can achieve an accuracy higher than 77 percent. SILHOUETTE manages to maintain predictive power in the transformed prediction model.

Fig. 8 shows that the prediction accuracy is higher for a medium number of machines, and is lower for a large or a small number of machines, possibly due to the following reasons. When we select experiments for obtaining training samples, we usually choose experiments with small numbers of machines in order to reduce training costs, thus the prediction power for a large number of machines is naturally limited. When the number of machines is too small, e.g., 4, the prediction accuracy is also low, possibly due to a high performance variation at low computation capabilities.

SILHOUETTE experiments on small fractions of the input data to train the prediction model to estimate the completion time on the entire input data. We use the same number of experiment settings to build the base and transformed prediction models and evaluate their prediction accuracy with a varied size of entire input data. Fig. 9 illustrates that when we use $0.5\times$, $1\times$, $1.5\times$, $2\times$, $2.5\times$ and $3\times$ dataset sizes, the prediction error is always below 15 percent, which shows that the predictive power of the models trained by SILHOUETTE with small subsets of the dataset can maintain as the input dataset scales.

4.4 Training Overhead

SILHOUETTE aims at finding the best cloud configuration with a low overhead. Therefore, we compare the completion time of running the job on the entire input data (both actual and predicted runtime) with the time spent in obtaining the training data for building the base prediction model. Note that the training time consists of two parts: the time to solve the optimization problem (3) and the time to run selected experiment settings. The former consumes several milliseconds using the python-embedded solver CVXPY, while the latter runs in hours for large-scale data analytics jobs. Therefore, the predominant component of the training overhead comes from running selected experiment settings.

Fig. 10a demonstrates that the training time of SILHOUETTE is below 20 percent of the total completion time for all applications except TeraSort. The dataset of TeraSort is larger than those of other applications, and unlike others, TeraSort read/write disks more frequently, thus requiring more training time. Despite this, large-scale data analytics jobs often recur, while the training is a one-time deal. If TeraSort is rerun for 40 times on the entire dataset, the training time is lower than 1 percent of the total actual execution time. The short training time confirms that SILHOUETTE can efficiently handle large-scale analytics jobs.

SILHOUETTE can also enable users to balance the tradeoff between the training cost and the accuracy of the trained prediction model. Fig. 10b shows the cost of running the experiments to obtain the training data and the RMSE of the

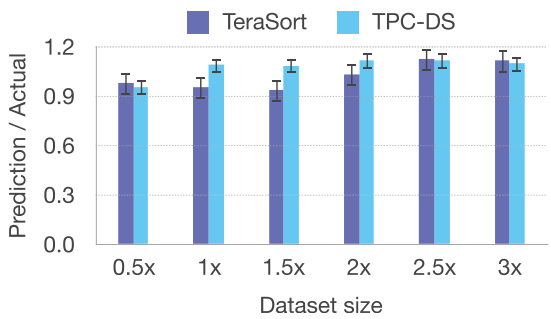

Fig. 9. Prediction accuracy of SILHOUETTE with different dataset sizes.

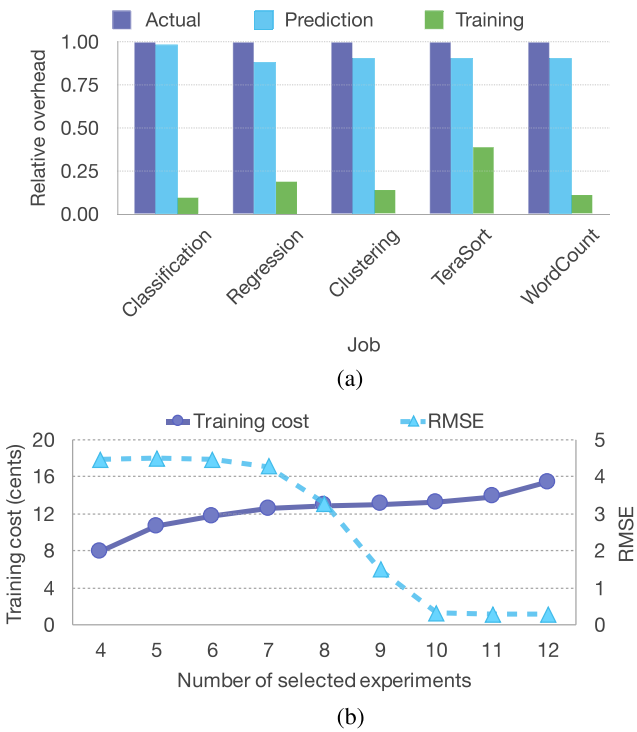

Fig. 10. (a) execution time versus training time of SILHOUETTE and (b) Training cost versus prediction error across different numbers of selected experiment settings.

trained prediction model when we select different numbers of experiment settings. We can observe that the cost increases first slowly then sharply with the number of selected experiment settings while the RMSE first decreases then becomes stable beyond a certain number of experiments. It is shown that choosing 10 experiment settings is a wise choice, which achieves a high prediction accuracy and induces an acceptable cost.

4.5 Applications

We apply SILHOUETTE to optimize cloud configuration for a particular job WordCount. We run experiments on `m4.large` instances to build a base prediction model and rerun one experiment for each of the three other instance types, namely `c5.large`, `r4.large` and `i3.large`. We integrate all prediction models for the selector builder to select the best cloud configuration. We assume that the selector aims at minimizing the total cost given a deadline, and we consider cloud configurations with different numbers of instances from the above four instance types to find the cheapest one. Table 2 shows the actual execution time and the total cost of running the job with the entire dataset using the best cloud configuration regarding each instance type. We can observe

TABLE 2
Optimizing Cloud Configuration

Instance	Cost (Normalized)	Runtime (Normalized)
m4	0.75	1.00
c5	0.51	0.81
r4	0.88	0.88
i3	1.00	0.86

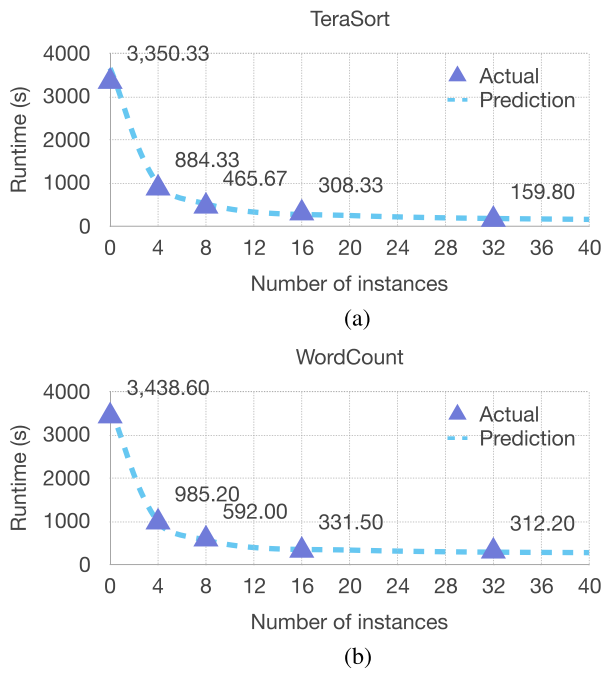

Fig. 11. Prediction and actual execution time with different cluster sizes of `m4.large` instances for (a) TeraSort. (b) WordCount.

that the execution time of the compute-optimized instance type `c5.large` is relatively the same as that of the storage-optimized instance type `i3.large`, yet the cost of the former is 1.93 times lower than that of the latter. This indicates that SILHOUETTE identifies WordCount as computation-intensive, thus choosing the instance type with better CPUs and stronger computing capacity.

When users have a fixed-time or fixed-cost budget, it can be tricky to figure out how many instances should be deployed for a job. We can use SILHOUETTE to optimize the number of instances given a specific instance type. We consider two jobs, TeraSort and WordCount. Using the prediction model trained with selected experiment samples, we can estimate the execution time across a wide range of cluster sizes, as shown in Fig. 11. We also show the actual execution time to validate the prediction results. It is shown that the predicted execution time closely approximates the actual execution time for TeraSort and WordCount across a wide range of cluster sizes of `m4.large` instances. Consider that a user has a fixed-time budget of 3,000s to run 10 times of TeraSort and 5 times of WordCount. According to Fig. 11 and taking error margin into account, SILHOUETTE can infer that launching 16 instances for TeraSort and 8 instances for WordCount is sufficient to meet their deadlines.

5 Related Work

Performance Prediction. There have been numerous efforts on predicting performance based on system properties and workload patterns to support SLOs or deadlines. Thus far, several approaches have concentrated on exploring the relationship between performance and system configurations for algorithms implemented on distributed platforms [33], [34], [35]. Distributed jobs can be modeled at a fine granularity to optimize configuration options [36], [37]. Various works predict interference among applications and consolidate VMs on

underlying physical machines [38], [39], [40]. In SILHOUETTE, we establish runtime prediction models based on training samples with small subsets of the dataset and the transformation between the runtime across different instance types.

Cloud Configuration Selection. Modern cloud providers have offered simple suggestions for selecting cloud configurations. Amazon EC2 [4], for example, recommends `c4` family of instances (compute-optimized) for machine learning jobs. However, there are many types of instances within a family, and it is unclear how to decide the exact type and number of instances. One method to choose the best configuration is to model application performance and then select the best cloud configuration [11], [41]. Another approach for choosing the best cloud configuration is to exhaustively search for the optimal or near-optimal cloud configuration. However, this method incurs a high overhead since there are nearly 100 instance types, and the cluster size is infinite. To reduce the search cost, previous works [10], [42], [43] have tried to build efficient frameworks based on Bayesian optimization. CherryPick [10] is proposed to reduce the searching time given a particular objective, but the fixed objective restricts its application since different users may prefer different performance-cost tradeoff. The frailty of Bayesian optimization is identified later in [14], and the authors proposed to use low-level performance information (e.g., memory usage) to instruct the search of Bayesian optimization.

Dynamic Resource Provisioning. Several dynamic resource provisioning approaches have been proposed to meet the application performance requirement. In [20], [21], customized cluster resource managers schedule distributed deep learning jobs to cater resource-intensive and time-consuming plans. Many online schedulers largely rely on historical traces to dynamically adjust resource allocations [44], [45], [46]. Nevertheless, an online adjustment may be disruptive to large-scale data analytics jobs, and the adjustment may not be fast enough to track the demand fluctuations. Also, prior large-scale scheduling systems [47], [48], [49] aimed at allocating resources adaptively based on the job progress and the data flow graph dependency. The security of executing data analytics jobs in the cloud environments has also been considered in existing works [50], [51]. SILHOUETTE focuses on long-term recurring large-scale analytics jobs and plans ahead to optimize the cloud configuration.

6 DISCUSSION

In this section, we discuss some of the issues and limitations of SILHOUETTE.

Model Transformation. SILHOUETTE transforms the base prediction model to the prediction model for other instance types by considering the ratio of execution times by the base instance type and another instance type under the same experiment settings. In this way, the prediction model of a new instance type can be established with a single or a few experiments. To improve the performance of model transformation, we may run more experiments or use more complicated mapping approaches. For example, we may consider that weight w'_k of the target instance type is a linear function of all or some of the weights of the base instance type, i.e., $w'_k = \sum_{\kappa=1}^{K} \lambda_{\kappa,k} w_\kappa$. Linear regression may be used to obtain the coefficients $\lambda_{\kappa,k}$ with more experiments than the simple solution in SILHOUETTE. A tradeoff of overhead and performance improvement needs to be considered to decide the appropriate mapping methods.

TABLE 3
Performance Variations

	Number of instances			
	4	8	16	32
`Classification`	6.25	3.28	11.91	4.01
`Clustering`	1.81	2.61	3.34	2.31
`TeraSort`	2.87	14.28	10.65	3.89
`WordCount`	3.56	7.18	3.94	1.64

Heterogeneous Cluster. Cloud service providers provide a wide selection of instance types, which can be jointly used in one cluster. With heterogeneous instance types, the cluster may achieve better performance with less resource provisioning. However, the number of potential cluster configurations increases exponentially with the types of instances, making it extremely challenging to pinpoint the most appropriate configuration. This is the very reason that existing works only focus on homogeneous instance clusters, which we follow in our design of SILHOUETTE. Heterogeneous clusters will be our future direction, and we will study how to optimize configuration consisting of different instance types efficiently.

Cloud Performance Variation. One primary concern about cloud computing solutions like Amazon EC2 is the performance variation, i.e., the completion time varies during repeated executions with the same cloud configuration. We investigate this issue by running jobs repeatedly on different sizes of instance clusters for 20 times. Table 3 shows that the standard deviation is always less than 15, and the variation is tiny for all cluster sizes. This indicates that SILHOUETTE is relatively robust under cloud performance variations. Furthermore, we may introduce container technologies and auto-scaling mechanisms to adjust the resource provisioning dynamically according to specific performance indicators, which improves the performance of SILHOUETTE.

Hardware and Application Customization. Following existing works CherryPick and Ernest, SILHOUETTE only considers the type and the number of instances, but not the specific hardware configurations of each instance, e.g., cache, network switches for communications. Furthermore, SILHOUETTE provides a general way of finding the best cloud configuration for different jobs without considering each job's specific characteristics. For example, some compute-intensive applications use accelerators in the cloud environment. In the future, we aim to improve SILHOUETTE by further considering hardware configurations of each instance and customize SILHOUETTE for different jobs.

Input Data Distribution. To train the execution time prediction model, we run selected experiment settings with small fractions of the input data. We randomly sample the input dataset to attain the designated workload of the selected experiment settings. However, some applications may require a larger sampling size in order to reach certain accuracy bounds. Our experiment selection process only aims to obtain as much information as possible with as few experiments as possible but does not consider the

input data size requirement of specific data analytics jobs. The input dataset may contain adversarial samples, which leads to data skewness and affects the performance of data analytics jobs. To account for input data distribution when selecting experiment settings and optimizing cloud configuration is a future direction that is worth exploring.

7 Conclusion

We present Silhouette to help users find the proper cloud configuration, which dramatically trims down the cost for recurring large-scale data analytics jobs. Silhouette builds runtime prediction models based on a few experiments on small fractions of input data with limited cluster sizes. Instead of running experiments on all types of instances as in existing solutions, Silhouette can intelligently transform the prediction model of one instance type to another with only one extra experiment. Our extensive experiments on Amazon EC2 with commonly-used benchmark data analytics jobs confirm that Silhouette can identify appropriate cloud configurations with up to 66 percent less overhead compared with existing solutions.

Acknowledgments

The work of Yanjiao Chen was supported in part by the National Natural Science Foundation of China under Grant 61972296, and in part by Wuhan Advanced Application Project under Grant 2019010701011419. The work of Qian Wang was supported in part by the National Key Research and Development Program of China under Grant 2020AAA0107700, in part the NSFC under Grant 61822207 and Grant U20B2049, and in part by the Fundamental Research Funds for the Central Universities under Grant 2042019kf0210. The work of Qian Zhang was supported in part by the RGC under Contract CERG 16204418, 16203719, R8015, and in part by the Guangdong Natural Science Foundation under Grant 2017A030312008.

References

[1] V. Tablan, I. Roberts, H. Cunningham, and K. Bontcheva, "Gatecloud. net: A platform for large-scale, open-source text processing on the cloud," *Philos. Trans. Royal Soc. A: Math. Phys. Eng. Sci.*, vol. 371, no. 1983, 2013, Art. no. 20120071.

[2] Y. Yan and L. Huang, "Large-scale image processing research cloud," in *Proc. 5th Int. Conf. Cloud Comput. GRIDs Virtualization*, 2014, pp. 88–93.

[3] B. Langmead and A. Nellore, "Cloud computing for genomic data analysis and collaboration," *Nat. Rev. Genetics*, vol. 19, no. 4, 2018, Art. no. 208.

[4] Amazon EC2, Accessed: Feb. 17, 2021. [Online]. Available: https://aws.amazon.com/ec2/

[5] Google Compute Engine, Accessed: Feb. 17, 2021. [Online]. Available: https://cloud.google.com/compute/

[6] Microsoft Azure, Accessed: Feb. 17, 2021. [Online]. Available: https://azure.microsoft.com/en-us/

[7] Amazon Lambda, Accessed: Feb. 17, 2021. [Online]. Available: https://aws.amazon.com/lambda/

[8] Google Cloud Functions, Accessed: Feb. 17, 2021. [Online]. Available: https://cloud.google.com/functions/

[9] Microsoft Azure Functions, Accessed: Feb. 17, 2021. [Online]. Available: https://azure.microsoft.com/en-us/services/functions/

[10] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "CherryPick: Adaptively unearthing the best cloud configurations for big data analytics," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2017, pp. 469–482.

[11] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2016, pp. 363–378.

[12] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the best VM across multiple public clouds: A data-driven performance modeling approach," in *Proc. ACM Symp. Cloud Comput.*, 2017, pp. 452–465.

[13] K. Ousterhout, C. Canel, S. Ratnasamy, and S. Shenker, "Monotasks: Architecting for performance clarity in data analytics frameworks," in *Proc. ACM Symp. Operating Syst. Princ.*, 2017, pp. 184–200.

[14] C.-J. Hsu, V. Nair, V. W. Freeh, and T. Menzies, "Arrow: Low-level augmented bayesian optimization for finding the best cloud VM," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 660–670.

[15] X. Meng *et al.*, "MLlib: Machine learning in apache spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, 2016.

[16] "Kaggle display advertising challenge dataset," Accessed: Feb. 17, 2021. [Online]. Available: https://www.kaggle.com/c/criteo-display-ad-challenge/data

[17] O. O'Malley, "Terabyte sort on apache hadoop," *Yahoo*, pp. 1–3, May 2008. [Online]. Available: http://sortbenchmark.org/Yahoo-Hadoop.pdf

[18] T. White, *Hadoop: The Definitive Guide*. Newton, MA, USA: O'Reilly Media, 2012.

[19] M. Poess, B. Smith, L. Kollar, and P. Larson, "TPC-DS, taking decision support benchmarking to the next level," in *Proc. ACM Int. Conf. Manage. Data*, 2002, pp. 582–587.

[20] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proc. ACM Eur. Conf. Comput. Syst.*, 2018, Art. no. 3.

[21] J. Gu *et al.*, "Tiresias: A GPU cluster manager for distributed deep learning," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2019, pp. 485–500.

[22] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 98–109, 2011.

[23] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. ACM Workshop Hot Topics Netw.*, 2012, pp. 31–36.

[24] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 443–454, 2014.

[25] S. Silvey, *Optimal Design: An Introduction to the Theory for Parameter Estimation*. Berlin, Germany: Springer, 2013.

[26] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Phys. Rev. E*, vol. 69, no. 6, 2004, Art. no. 066138.

[27] B. C. Ross, "Mutual information between discrete and continuous data sets," *PloS One*, vol. 9, no. 2, 2014, Art. no. e87357.

[28] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, pp. 1226–1238, Aug. 2005.

[29] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. USENIX Workshop Hot Topics Cloud Comput.*, 2010, Art. no. 95.

[30] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "RCV1: A new benchmark collection for text categorization research," *J. Mach. Learn. Res.*, vol. 5, no. Apr, pp. 361–397, 2004.

[31] M. Armbrust *et al.*, "Spark SQL: Relational data processing in spark," in *Proc. ACM Int. Conf. Manage. Data*, 2015, pp. 1383–1394.

[32] Apache Hadoop MapReduce, Accessed: Feb. 17, 2021. [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[33] A. D. Popescu, A. Balmin, V. Ercegovac, and A. Ailamaki, "PREDIcT: Towards predicting the runtime of large scale iterative analytics," *Proc. VLDB Endowment*, vol. 6, no. 14, pp. 1678–1689, 2013.

[34] S. A. Jyothi *et al.*, "Morpheus: Towards automated SLOs for enterprise clusters," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 117–134.

[35] K. Rajan, D. Kakadia, C. Curino, and S. Krishnan, "PerfOrator: Eloquent performance models for resource optimization," in *Proc. ACM Symp. Cloud Comput.*, 2016, pp. 415–427.

[36] J. Shi, J. Zou, J. Lu, Z. Cao, S. Li, and C. Wang, "MRTuner: A toolkit to enable holistic optimization for MapReduce jobs," *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1319–1330, 2014.

[37] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *Proc. ACM Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 1355–1364.
[38] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini, "DejaVu: Accelerating resource allocation in virtualized environments," in *Proc. ACM Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2012, pp. 423–436.
[39] A. K. Maji, S. Mitra, and S. Bagchi, "ICE: An integrated configuration engine for interference mitigation in cloud services," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2015, pp. 91–100.
[40] X. Chen, L. Rupprecht, R. Osman, P. Pietzuch, W. Knottenbelt, and F. Franciosi, "CloudScope: Diagnosing and managing performance interference in multi-tenant clouds," in *Proc. IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2015, pp. 164–173.
[41] Y. Jiang, L. R. Sivalingam, S. Nath, and R. Govindan, "WebPerf: Evaluating what-if scenarios for cloud-hosted web applications," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 258–271.
[42] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 528–536.
[43] V. Dalibard, M. Schaarschmidt, and E. Yoneki, "BOAT: Building auto-tuners with structured Bayesian optimization," in *Proc. ACM Int. Conf. World Wide Web*, 2017, pp. 479–488.
[44] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 127–144, 2014.
[45] H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "SLAQ: Quality-driven scheduling for distributed machine learning," in *Proc. ACM Symp. Cloud Comput.*, 2017, pp. 390–404.
[46] Y. Chen, L. Lin, and B. Li, "Razor: Scaling backend capacity for mobile applications," *IEEE Trans. Mobile Comput.*, vol. 19, no. 7, pp. 1702–1714, Jul. 2020.
[47] E. Boutin *et al.*, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2014, pp. 285–300.
[48] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 81–97.
[49] I. Gog, M. Schwarzkopf, A. Gleave, R. N. Watson, and S. Hand, "Firmament: Fast, centralized cluster scheduling at scale," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 99–115.
[50] Y. Chen, X. Gong, Q. Wang, X. Di, and H. Huang, "Backdoor attacks and defenses for deep neural networks in outsourced cloud environments," *IEEE Netw.*, vol. 34, no. 5, pp. 141–147, Sep./Oct. 2020.
[51] Q. Wang, M. He, M. Du, S. S. M. Chow, R. W. F. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 3, pp. 496–510, May/Jun. 2018.

Yanjiao Chen (Senior Member, IEEE) received the BE degree in electronic engineering from Tsinghua University, in 2010, and the PhD degree in computer science and engineering from the Hong Kong University of Science and Technology, in 2015. She is currently a bairen researcher with the College of Electrical Engineering, Zhejiang University, China. Her research interests include network security, mobile sensing, and spectrum management.

Long Lin (Student Member, IEEE) received the BE degree in computer science from Central South University, China, in 2016. He is currently working toward the master's degree with the School of Computer Science, Wuhan University, China. His research interest includes cloud computing.

Baochun Li (Fellow, IEEE) received the BEngr degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995, and the MS and PhD degrees from the Department of Computer Science, University of Illinois at UrbanaChampaign, Urbana, in 1997 and 2000. Since 2000, he has been with the Department of Electrical and Computer Engineering, University of Toronto, where he is currently a professor. His research interests include large-scale distributed systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. He has co-authored more than 290 research papers, with a total of more than 13 000 citations, an H-index of 59 and an i10-index of 189, according to Google Scholar Citations. He was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000. He is a member of ACM.

Qian Wang (Senior Member, IEEE) received the PhD degree from the Illinois Institute of Technology, Chicago, IL. He is a professor with the School of Cyber Science and Engineering, Wuhan University. His research interests include AI security, data storage, search and computation outsourcing security and privacy, wireless systems security, big data security and privacy, and applied cryptography etc. He received National Science Fund for Excellent Young Scholars of China in 2018. He is also an expert under National "1000 Young Talents Program" of China. He is a recipient of the 2016 IEEE Asia-Pacific Outstanding Young Researcher Award. He is also a co-recipient of several best paper and best student paper awards from IEEE ICDCS'17, IEEE TrustCom'16, WAIM'14, and IEEE ICNP'11 etc. He serves as an associate editor for the *IEEE Transactions on Dependable and Secure Computing* (TDSC) and the *IEEE Transactions on Information Forensics and Security* (TIFS). He is a member of ACM.

Qian Zhang (Fellow, IEEE) received the BS, MS, and PhD degrees in computer science from Wuhan University, China, in 1994, 1996, and 1999, respectively. She joined Hong Kong University of Science and Technology in September 2005 where she is a full professor with the Department of Computer Science and Engineering. Before that, she was in Microsoft Research Asia, Beijing, from July 1999, where she was the research manager of the Wireless and Networking Group. She has published about 300 refereed papers in international leading journals and key conferences in the areas of wireless/Internet multimedia networking, wireless communications and networking, wireless sensor networks, and overlay networking. Her contribution to the mobility and spectrum management of wireless networks and mobile communications. She has received MIT TR100 (MIT Technology Review) Worlds Top Young Innovator Award. She also received the Best Asia Pacific (AP) Young Researcher Award elected by IEEE Communication access and management, as well as wireless sensor networks.