Big Data
Volume 8, Number 4, 2020

DOI: 10.1089/big.2019.0123

ORIGINAL ARTICLE

MonkeyKing: Adaptive Parameter Tuning on Big Data Platforms with Deep Reinforcement Learning

Haizhou Du,[1,2,*] Ping Han,[2] Qiao Xiang,[3] and Sheng Huang[2]

Abstract

Choosing the right parameter configurations for recurring jobs running on big data analytics platforms is difficult because there can be hundreds of possible parameter configurations to pick from. Even the selection of parameter configurations is based on different types of applications and user requirements. The difference between the best configuration and the worst configuration can have a performance impact of more than 10 times. However, parameters of big data platforms are not independent, which makes it a challenge to automatically identify the optimal configuration for a broad spectrum of applications. To alleviate these problems, we proposed MonkeyKing, a system that leverages past experience and collects new information to adjust parameter configurations of big data platforms. It can recommend key parameters, which have strong impact on performance according to job types, and then combine deep reinforcement learning (DRL) to optimize key parameters to improve job performance. We choose the current popular deep Q-network (DQN) structure and its four improved algorithms, including DQN, Double DQN, Dueling DQN, and the combined Double DQN and Dueling DQN, and finally found that the combined Double DQN and Dueling DQN has a better effect. Our experiments and evaluations on Spark show that performance can be improved by $\sim$25% under best conditions.

Keywords: big data platforms; performance optimization; parameter tuning; deep reinforcement learning

Introduction

In recent years, traditional computing models in the era of big data have gradually failed to meet performance and efficiency requirements, resulting in some excellent big data processing platforms, such as Hadoop,[1] Spark,[2] and Storm.[3] The execution engines of big data platforms have evolved into efficient and complex systems with multiple configurable parameters, and the impact of parameters may vary from applications or clusters. Besides, users can adjust parameters according to specific application requirements.

Related experiments have shown that choosing the right parameter configurations can greatly improve the performance of an application, and improper selection may significantly degrade the performance, and increase the average operating cost by 2–3 times, and in the worst case by 12 times.[4] The default parameter configurations cannot meet the performance requirements of big data platform users, so researchers have done a lot of work to find the optimal parameter configurations for applications and clusters. Obviously, it is impossible to check the impact of different values of all tunable parameters on performance, as their combination is nondeterministic polynomial time-hard. In the past, we used expert experience and manual operations to adjust parameters to improve performance, but it was expensive and time-consuming.

Currently, there are two main ways to tune the configuration parameters of big data analytics platforms. First, parameters can be manually adjusted by trial and error.[5] Although it is intuitive and effective, the inefficiency and time-consuming due to large parameter space and complex interactions between parameters cannot be ignored.

Second, some researchers have proposed a cost-based performance modeling method to tune parameters of Hadoop platform.[6] However, the underlying implementation mechanisms vary widely among platforms, so we cannot use this method directly on other platforms. At

[1]*College of Electronics and Information Engineering, Tongji University, Shanghai, China.*
[2]*School of Computer Science and Technology, Shanghai University of Electric Power, Shanghai, China.*
[3]*Department of Computer Science, Yale University, New Haven, Connecticut, USA.*

*Address correspondence to: *Haizhou Du, College of Electronics and Information Engineering, Tongji University, Shanghai 201804, China*, E-mail: duhaizhou@shiep.edu.cn

the same time, cost-based modeling is a white-box approach that requires an in-depth understanding of the internal structure of systems. And using a cost-based model to capture system complexity due to the inclusion of the software stack and hardware stack[7] is very difficult. Therefore, it is a challenging task to adjust a wide range of big data analytics platform applications by automatically searching all possible parameter configurations.

For the complexity of big data platforms, complex business configurations, cumbersome manual configuration, and error-prone problems all become the vital issues.[8] Different users have different requirements for parameter configuration, which makes the above problems more prominent. Our solution is to be able to predict the efficiency of subsequent system execution through machine learning algorithms and finally convert it into a regression problem. Traditional machine learning can be used to regress and optimize parameters. However, there are some problems. For example, algorithms rely heavily on the amount and quality of training data. In big data platforms, it is difficult to generate enough sample data for training because sample acquisition comes at the expense of time and cost.

In this article, we study reinforcement learning (RL) techniques because RL is a subfield of machine learning related to decision-making and action control.[9] RL is a way to simulate interactive learning between agent and environment. It only receives some good or bad feedback every time it takes action. Through these accumulated feedbacks, it can adjust and optimize actions and finally iteratively generate the optimal strategy. In recent years, with the advancement of deep neural network (DNN) technology, the combination of deep learning (DL) and deep reinforcement learning (DRL) has achieved good results in many real and complex environments, such as DeepMind's Atari results[10,11] and AlphaGo.[12] Inspired by these results, we study deep Q-network (DQN) algorithm and propose using a DRL technique that combines Double DQN[13] and Dueling DQN[14] to dynamically train optimal parameter configuration to improve job performance.

The contributions of our work are summarized as follows:

- For the explosion problem of parameter combination, we can select the key parameters for different jobs that have an impact on performance by using a feature selection technique called LASSO.
- For the parameter optimization problem, we leverage DRL techniques to dynamically tune parameters and verify their versatility through experiments.
- For the problem of recurring jobs, we have designed a historical information base to save job running information, so as to effectively recommend suitable parameters for different workloads.

The rest of the article is organized as follows. Background and motivation are shown in the Background and Motivation section. Preliminary will be listed in the Preliminary section. The specific design ideas of MonkeyKing will be proposed in the MonkeyKing Design section. The implement will be given in the Implementation section. Experimental methodology and results are given in the Experiments and Evaluation section. Discussion is shown in the Discussion section. The related work is discussed in the Related Work section. Finally, the conclusions are summarized in the Conclusion section.

Background and Motivation

In this section, we show the importance and challenges of choosing the best parameter configurations. We also present the feasibility of DRL in parameter tuning.

Big data platforms have received more and more attention, but they have hundreds of configurable parameters, and the setting of parameters is complicated, which hinders the popularity and application of big data platforms to a certain extent. In general, the performance-related parameters of big data platforms can be divided into several different types, such as runtime environment, shuffling behavior, compression and serialization, memory management, execution behavior, and network. There are also complex interactions between different types of parameters. A single tuning technology is not suitable for solving such problems, and a huge search space makes the use of trial and error unrealistic. In addition, jobs running on big data platforms are also rich in types such as WordCount, Sort, PageRank, machine learning tasks, and image processing tasks.[15] The dependency analysis of parameters for each job type is also extremely complex.

Parameter optimization is an important branch of performance optimization for big data platforms. However, developing high-performance computing using big data platforms is not straightforward. If parameter configurations are not properly set, the job may be executed for a long time, and the benefits of big data platforms as fast calculation engines cannot be fully demonstrated. For resource parameters, if set too low,

the cluster resources may not be fully utilized and the operation will be very slow. If set too large, the queue does not have enough resources to provide, which can cause various exceptions.[16] In general, choosing the right resource parameters for big data analytics platform has the following challenges.

The explosion of parameters search space

Big data platforms have hundreds of configurable parameters so far, and each parameter has at least two tunable values, which causes the parameter space to explode exponentially, and its impact cannot be examined in detail. The large parameter space and the complex interactions between parameters make it impossible to manually adjust parameters through trial and error.

The selection of key parameters

The total parameters' number of big data platforms has reached hundreds of thousands, but not all parameters are suitable for tuning, and only a small number of parameters affect performance. We cannot directly determine which parameters have an impact on performance. It is unwise to randomly select parameters or adjust all parameters. To this end, we have specifically proposed a method of selecting key parameters.

The diversity of applications

The demand for applications for the amount of resources on big data platforms varies by their diversity. In other words, parameters that affect performance of the current application do not necessarily apply to other applications at the same time. Therefore, before resource parameter allocation, we also need to separately analyze the impact of parameters on different types of applications.

The reuse of historical information

Repetitive analysis jobs are typically performed on big data platforms,[17] so the reuse of historical information is particularly important. If important information can be extracted from historical information, the efficiency of subsequent analysis will be greatly improved and the cost will be reduced.

In previous research and work, most researchers only focused on parameter optimization. Of course, parameter optimization is necessary and important, but the work of parameter selection cannot be ignored. For instance, as one of the most popular big data platforms, Spark has over 180 parameters, more than 150 of which are configurable parameters, it is not easy to pick out the parameters that have strong impact on cluster performance. We used to rely on expert experience and other scholars' recommendations, which are effective but not very accurate. For big data platforms, different cluster environments and workloads have different sensitivity to parameters. Even with expert recommendation, it takes time and cost. Therefore, we need a simple and efficient algorithm that can recommend the required parameters in a short time.

DRL is an end-to-end perception and control system with strong versatility. Its learning process can be described as: (1) at each moment, *agent* interacts with *environment* to obtain a high-dimensional observation and uses DL to perceive observation to obtain a specific state feature table; (2) evaluating the value function of each *action* based on the expected reward, and mapping the current *state* to the corresponding action through a certain strategy; (3) *Environment* reacts to this *action* and gets the next *state*. By continuously cycling the above process, the optimal strategy for achieving the goal can be finally obtained. The DRL principle framework is shown in Figure 1.

With the continuous development of RL technology, many areas have successfully applied RL and made good progress. Mnih et al.[18] used recent advances in training DNNs to develop a novel artificial agent, termed a DQN, that can learn successful policies directly from high-dimensional sensory inputs using end-to-end RL. AuTO[19] is an end-to-end automated traffic optimization system based on RL that collects network information, learns from past decisions, and performs operations to achieve operator-defined

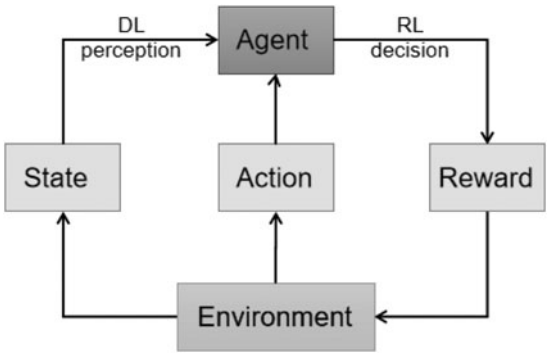

FIG. 1. DRL principle framework. DRL, deep reinforcement learning.

goals. Hansen[20] presented a novel definition of the RL state, actions and reward function that allows a DQN to learn to control an optimization hyperparameter. Ma et al. proposed ANANKE,[21] a scheduling system addressing these challenges. It extends the state-of-the-art in portfolio scheduling for data centers with a reinforcement-learning technique and proposes various scheduling policies for managing complex workflows. Yan et al.[22] developed CAPES based on Lustre file system, a model-free DRL unsupervised parameter adjustment system driven by DNN, which aims to find the optimal value of tunable parameters in computer systems. Yan et al.[22] used DQN to establish a parameter tuning system, which effectively proves the contribution of DQN to parameter optimization, but they mainly focus on the hyperparameter optimization of machine learning algorithms.

Based on the current research status of parameter optimization and DRL technology, in our strategy, DQN-based algorithms are used to automate parameter tuning to improve job performance.

Preliminary

In this section, we mainly introduce the working principle of MapReduce for big data platforms and several principles of DRL technologies.

MapReduce

MapReduce is a parallel scalable computing model with good fault tolerance, mainly for batch processing of massive offline data. MapReduce consists of a JobTracker and a TaskTracker. JobTracker is responsible for resource management and job control, and TaskTracker is responsible for running tasks.

In Map phase, the input file is split into splits, and each split is used as input to a map task. The intermediate result of input data processed by the map stage is written to the memory buffer and determines which partitioner the data is written to. When the written data reaches the threshold of memory buffer, a thread is started to write the data in the memory to the disk. During the data writing process, the MapReduce framework sorts the keys. In Reduce phase, when all map tasks are completed, each map task forms a final file, and the file is divided by region. Before reduce task starts, it will start a thread to get the map result data to the corresponding reduce task and continuously merge the data to prepare for the data input of reduce. After all the map tasks are completed, the reduce task is started, and the output is finally stored in Hadoop Distributed File System.

Deep Q network

For the problem of large state set size, just like our parameter tuning, DQN is a good solution. The basic idea of DQN comes from Q-Learning. But the difference with Q-Learning is that its Q value is calculated by a neural network called Q network. The optimal action-value function $Q^*(s', a')$ in multiple experiments is:

$$Q(s, a) = E_{s' \sim \varepsilon}[r + \gamma \max Q^*(s', a') | s, a]. \quad (1)$$

Multiple Q values can be obtained by multiple experiments in state s. When the number of experiments tends to infinity, this expected value tends to the true $Q(s, a)$. In DQN, each Q value is estimated through the network:

$$Q(s, a; \theta) \approx Q^*(s, a). \quad (2)$$

The input of DQN is the state vector corresponding to our states, and the output is the action value function Q of all actions in this state. The main skill used by DQN is experience replay, which saves the rewards and status updates obtained by each interaction with the environment for the subsequent update of the target Q value. There is an error in the target Q value obtained by experience replay and the Q value calculated through the Q network, and a loss function L_i can be introduced to minimize the error. It can be expressed as:

$$L_i(\theta_i) = E_{s, a \sim \rho(.)}[(y_i - Q(s, a; \theta))^2], \quad (3)$$

where y_i is expressed as shown in Eq. (5). When calculating the value of y_i, the parameter θ_{i-1} is used after the last network update.

$$y_i = r + \gamma \ \max_a Q(s', a' | \theta_{i-1}). \quad (4)$$

Double deep Q network

The target Q value of DQN is directly obtained by greedy algorithm. Although the maximum value can quickly make the Q value close to the possible optimization target, it is easy to cause over estimation, and the final algorithm model has a large bias. To solve this problem, Double DQN achieves the problem of eliminating overestimation by decoupling the selection of the target Q value action and the calculation of the target Q value. The structure of Double DQN is shown in Figure 2.

In double deep Q-network, it is no longer directly looking for the maximum Q value in each action in the target Q network, but first finding the action

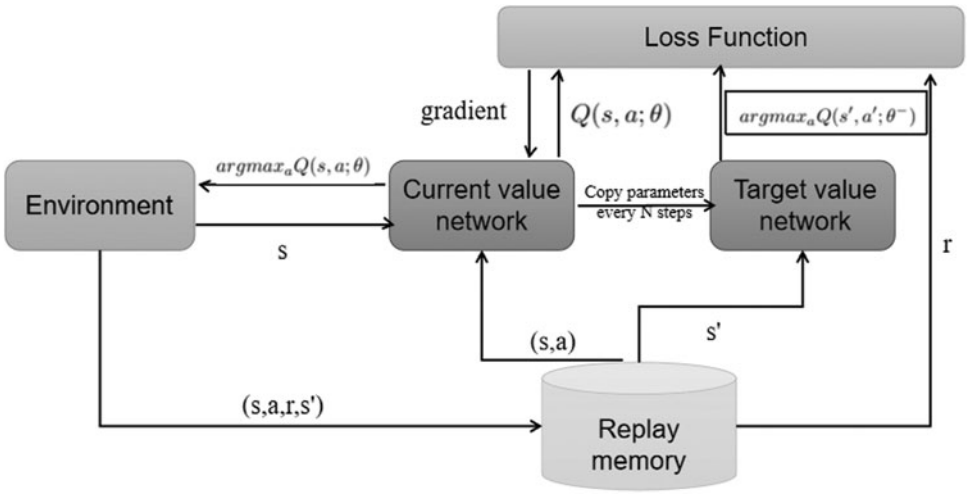

FIG. 2. The structure of Double DQN. DQN, deep Q-network.

corresponding to the maximum Q value in the current Q network.

$$a_{\max} = \arg\max_a Q(s', a'; \theta_{i-1}). \quad (5)$$

Then use this selected action $a_{\max}$ to calculate the target Q value in the target network. The final target Q value is expressed as:

$$y_i = r + \gamma Q(s', \arg\max_a Q(s', a'; \theta_i); \theta_{i-1}). \quad (6)$$

Dueling deep Q network

With the help of Double DQN, we solved the problem of overestimation of the DQN algorithm, and the convergence of DQN can also be improved by using Dueling DQN.

In Figure 3, Dueling DQN attempts to optimize algorithm by optimizing the structure of neural network. It considers dividing the Q network into two parts. The first part is only related to state s and has nothing to do with action a to be used. This part is called value function and is written as $V(s;\theta,\alpha)$. The second part is related to state s and action a. This part is called advantage function and is recorded as $A(s,a;\theta,\beta)$. Then, the final value function can be re-expressed as:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\alpha) + A(s,a;\theta,\beta), \quad (7)$$

where θ is the network parameter of public part, α is the network parameter of the unique part of value function, and β is the network parameter of the unique part of advantage function.

MonkeyKing Design

In this section, a parameter optimization system named MonkeyKing is proposed for modeling the performance as it is a key component in an auto-tuning system. Next, we will elaborate and analyze objective function, overall design, the two important modules and so on.

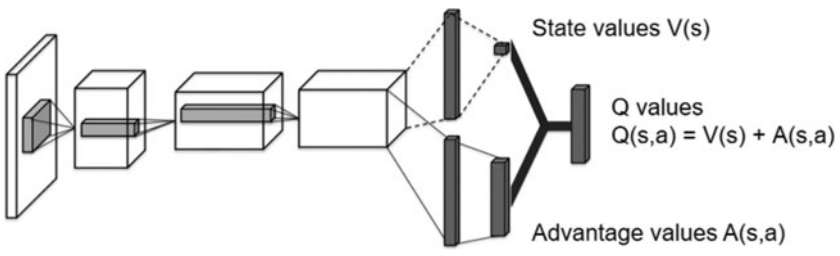

FIG. 3. The structure of Dueling DQN.

Objective function

We analyzed various performance indicators for performance and noted that job completion time (JCT) can directly and effectively reflect whether performance has improved. The perfect thing is that we can get JCT directly from the log information generated by the submitted job. From this, we can dedicate the goal to minimize the JCT that satisfies the constraints under a given set of configuration parameters. Considering the general situation that users want performance improvement while ensuring CPU utilization, we have added CPU utilization constraints.

We are committed to optimizing performance by adjusting big data platforms' configuration parameters. First, we need to define the objective function. The objective function can be described by a formula as:

$$\begin{aligned} &\max \ P = F(c, u) \\ &s.t. \quad u \geq u_{\min} \end{aligned}, \tag{8}$$

where P denotes the performance of big data platforms, c represents the valid value of a given set of configuration parameters, u is the CPU utilization of jobs, F indicates a function of P about c and u, $u_{\min}$ in the constraint indicates the lowest CPU utilization, which is a variable that users can define in advance according to different requirements. Our goal is to find the optimal solution that satisfies the objective function, which is the optimal configuration parameter value that satisfies the constraint.

Overall design

Referring to Figure 4, MonkeyKing consists mainly of three modules: Parameter Selection Module, Parameter Tuning Module, and historical information base. MonkeyKing is a combination of both parameter selection and parameter optimization. We first select the key parameters that have a great influence on performance from the original parameter set through Parameter Selection Module and then deliver them to Parameter Tuning Module. When a workload is running, it will generate corresponding parameter configuration information and job log. The environment based on DQN algorithm interacts with parameter configuration and log information in the cluster to obtain corresponding actions, states, and rewards. At the same time, the calculation results of these two modules will be saved in the historical information base.

Parameter selection module

First, we pay attention to the parameter selection section because this work is done before parameter optimization. We cannot and do not have to study all the parameters because this is a dimensional disaster problem, and only a subset of parameters actually affects

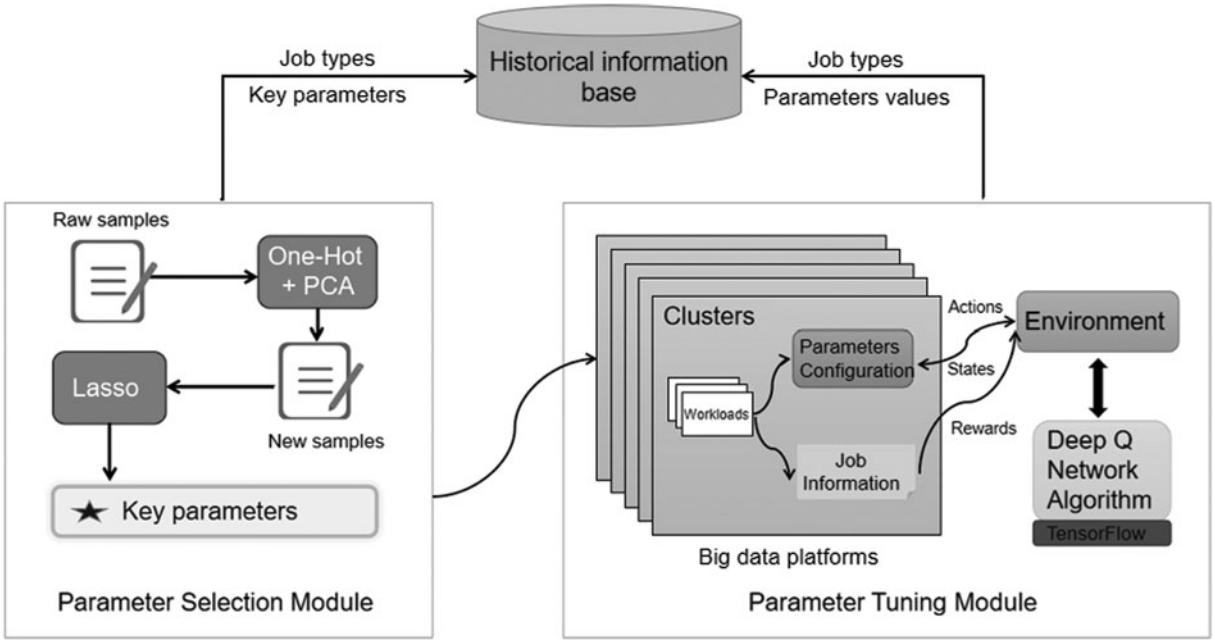

FIG. 4. The overview of MonkeyKing.

performance. Our goal is to determine this minimal subset that has a strong impact on the performance of big data platforms with hundreds of tunable parameters for diversity job types and job sizes. Filtering out parameters that are not related to performance or have a weak impact on performance, leaving important parameters not only reduces the difficulty of learning tasks but also allows subsequent tuning learning processes to build models on only a small number of parameters. This is very helpful in mitigating dimensional disasters.

The parameter selection module mainly includes parameter data processing and feature selection using LASSO. In this module, we skillfully combine some existing high-efficiency technologies, such as One-Hot coding technology, principal components analysis (PCA) dimensionality reduction technology, and LASSO feature selection technology. The details of this work are shown in the Parameter Selection Module in Figure 1.

Parameter tuning module

The parameter optimization problem of big data platforms is obviously a parameter combination problem, which results in a large parameter search space. For this reason, we apply DRL techniques, including four DQN frameworks. Previous work has been performed to apply DRL techniques to hyperparameter optimization, thus improving the efficiency of algorithms. However, we cannot directly apply this approach to big data platforms because there are hundreds of parameters in big data platforms that are inconsistent in type.

DRL formulation of parameter tuning

Markov decision process. Each parameter of Spark has a certain range of variation, so state space can be set as a set of all valid values of key parameters. Action space includes actions that can adjust key parameters, and reward is performance improvement after executing Spark job. It is represented by JCT in our application scenario. *Agent* is in the *environment*. At time t, a certain state s_t takes action a_t to reach the next state s_{t+1}, and at the same time obtains corresponding reward r_t. *Agent* can take action in s_{t+1}, get new state and reward.

State space. The state space contains all the states in the parameter tuning scenario. For parameters, we can use their valid values to represent *state*. Since we also consider different workloads, we should also include information such as job type and job size. It can be expressed as: (para1, para2, …, paraN, Jtype, Jsize).

Action space. The action space is a combination of all the actions we can take. The processed parameter values are all numeric types. For each numeric parameter, we can take three adjustment actions: increase, decrease, and unchanged. So we use the action space containing these three actions to describe the transition between *states*.

Rewards. Rewards are feedback to the agent on how good its actions are. The reward can be obtained after the completion of a job. At time step t, the value of *reward* depends on the ratio of the completion time (T') of job under a set of new configuration parameters obtained by taking a certain action and the JCT (T) under the default configuration parameters, ratio $= \frac{T'}{T}$. Therefore, at time step t, *reward* can be expressed as:

$$r_t = \begin{cases} 1 & (\text{ratio} < 1 \quad \text{and} \quad u \geq u_{\min}) \\ -1 & (\text{ratio} > 1 \quad \text{and} \quad u \geq u_{\min}) \\ 0 & (\text{ratio} = 1 \quad \text{or} \quad u < u_{\min}) \end{cases}. \tag{9}$$

Historical information base

We found that to get the running information of jobs, it is often necessary to run jobs repeatedly, but it is time-consuming. If we can extract the running information of jobs and save it to a database, we can query the database when needed, which can improve efficiency and usability. For the problem of parameter tuning, we need properties such as job type, parameter information, and completion time. Therefore, we have designed a historical information base to store the key parameters and parameter weights (the degree of impact on performance) calculated by the parameter selection module and the optimal values of the key parameters calculated by the parameter optimization module. The detailed design of the database is shown in Table 1. After each job runs, the relevant job type,

Table 1. Design of historical information base

ID	*JobName*	*executor.memory*	*Weight1*	*executor.cores*	*Weight2*	…	*ParaN*	*WeightN*
1	WordCount	4g	2.437	3	3.558	…	Optimum/default value	Weight value
2	Sort	4g	2.977	2	3.034	…	Optimum/default value	Weight value
3	PageRank	4g	2.341	1	3.284	…	Optimum/default value	Weight value
4	Kmeans	4g	2.602	1	2.988	…	Optimum/default value	Weight value
…	…	…	…	…	…	…	…	…

parameter name, parameter value, and weight are saved in the database. N is the number of parameters. For key parameters, the database stores the optimized values, and the non-key parameters store the default values. Once values changes, the original information of the database is overwritten with new values.

The above content introduces the main design ideas of MonkeyKing. In the following part, we will conduct experiments based on the system on Spark platform.

Implementation

In this section, we mainly introduce the specific module design and implementation in the previous section.

Data preprocessing

The step of parameter data preprocessing needs to be completed before the feature selection, and the purpose of data preprocessing is to normalize the parameter data. This is necessary because parameter values are not always contiguous and not all digital data but contains many categorical values. In our proposed solution, first convert all functions to "virtual" variables using One-Hot encoding[23] and then normalize the data.

One-Hot encoding is proposed because most of the algorithms are calculated based on the metrics in the vector space, to make the nonpartial relationship variable values have no partial order and the distances to the dots are equal.[24] Using One-Hot coding, the value of the discrete feature is extended to the European space. A certain value of the discrete feature corresponds to a certain point in the European space, which makes the distance calculation between the features more reasonable. After the One-Hot encoding of the discrete features, the features of each dimension can be regarded as continuous features.

Although One-Hot coding solves the problem that classifiers cannot handle attribute data well, it expands the feature to some extent. When the number of categories increases, the feature space becomes very large. In this case, PCA[25] can generally be used to reduce the dimensions. And the combination of One Hot encoding and PCA is also very useful in practice. PCA is one of the most widely used data reduction algorithms. Its main idea is to map the n-dimensional features to the k-dimension. This k-dimensional is a new orthogonal feature, also called the principal component, which is a k-dimensional feature reconstructed on the basis of the original n-dimensional features.

The job of PCA is to sequentially find a set of mutually orthogonal coordinate axes from the original space. The selection of new coordinate axes is closely related to the data itself. Wherein, the first new coordinate axis selection is the direction with the largest variance in the original data, and the second new coordinate axis selection is the plane orthogonal to the first coordinate axis, so that the variance is the largest, and the third axis is the first one. The variance of the plane orthogonal to the two axes is the largest. By analogy, n coordinate axes can be obtained. In this way, most of the variances are included in the first k axes, and the subsequent axes contain almost zero variance. The remaining axes can then be ignored, leaving only the first k axes with most of the variance. In fact, this is equivalent to retaining only the dimensional features that contain most of the variance, while ignoring the feature dimensions containing the variance of almost zero, to achieve dimensionality reduction of the data features.

After using PCA for dimensionality reduction, the data dimensions will reduce from n to k, which is $n \gg k$. Then, we normalize the data using the Z-score method, which subtracts the mean and dividing by the standard deviation. Since the original data are more variegated, the accuracy is reduced, and the process of standardization can effectively improve the accuracy.

Parameter selection with LASSO

After adopting the PCA techniques of the previous section, we will consider how to exact the most key parameters from the preliminary set of features.

As we known, linear regression is a statistical method that can be used to determine the strength of the relationship between one or more dependent variables and each of the independent variables. The relationship between the independent variable and the dependent variable can be modeled based on the weight of a linear predictor function estimated from the data. The most common method of fitting a linear regression model is the ordinary least squares (OLS), which estimates the regression weight by minimizing the residual squared error.[26] We could use OLS to determine important parameters, but it has two significant drawbacks in high(er) dimensional settings. First, OLS estimate has low deviation but high variance, and the high variance reduces the prediction and variable selection accuracy of the model. Second, OLS estimates that it is difficult to interpret the number of features because it never removes irrelevant features.

To avoid high variance and irrelevant features disturbance problems, we use a regularized version of

the least squares method called LASSO, which reduces the effects of unrelated variables in linear regression models by penalizing models with large weights. Compared with other regularization and feature selection methods, LASSO's main advantage lies in its interpretability, stability, and computational efficiency.[27] There is also practical and theoretical work supporting its effectiveness as a consistent feature selection algorithm.[28]

LASSO works by adding the $L1$ penalty (the constant λ multiplied by the sum of the absolute weights of the loss functions). Each nonzero weight contributes to the penalty, so it can force other weights to zero by effectively narrowing some weights. We consider features with zero weight as irrelevant features, LASSO can automatically distinguish features with nonzero weights (related features) and features with zero weight (unrelated features). In the nonzero weight feature, we get a ranking of the extent to which features affect performance according to the size of weights. The number of features with nonzero weights depends on the strength of their penalty, which can be controlled by adjusting the value of λ. LASSO reduces the variance and creates a more stable model by reducing the value of the smaller weight to zero, thus improving the prediction accuracy of the OLS estimate.

As with the usual regression scenarios, we construct a set of independent variables (X) and a dependent variable (Y) based on historical data obtained from previous job logs. In our application scenario, X is the parameter of the Spark platform, and Y is the corresponding JCT. First, a high penalty is set in the LASSO algorithm, and the weights of all features are zero. Then reduce the penalty in small increments, gradually distinguish between nonzero weights and zero weights, and sort according to the values.

To avoid the disaster of dimensional explosion and to DL methods, we reduce the number of parameters as much as possible. We analyzed and summarized all configurable parameters, filtered some parameters that could not be adjusted. First, we use the PCA to reduce some dimensions. Then, we take LASSO algorithm into the remaining parameter candidate set. The two stages can not only reduce the overall dimension calculation but also obtain more accurate key parameters. Finally, we selected the smallest candidate parameter set that affected performance. In our Spark application scenario, we selected 55 adjustable parameters from all spark platform parameters, which were reduced almost half dimensions after PCA algorithm. And after the LASSO algorithm, a total of 14 key parameters were selected as the parameters that seriously affected the JCT of Spark platform. We can conclude that the PCA step improves the performance and the LASSO technology is very sufficient to select key parameters relevant to JCT in cases.

Experiments and Evaluation

In this section, we verify the effectiveness of MonkeyKing on a 10-node Spark cluster, including one master node and nine slave nodes. Each node has also the same software stack: Ubuntu 14.04.3, Spark 2.2.0, Hadoop 2.7.2, Hibench 7.0, Java 1.8.0, and Scala 2.11.4. Two kinds of hardware configurations existed in the cluster. In the parameter tuning module, we used several different DRL algorithms, including DQN, Double DQN, Dueling DQN introduced in the system design section, and the combination of Double and Dueling DQN. Next, we will introduce benchmark, evaluation metrics, experimental results and analysis, and so on.

Benchmark

HiBench is a big data benchmark suite that helps assess speed, throughput, and system resource utilization of big data frameworks.[29] The frameworks it supports are: HadoopBench, SparkBench, StormBench, FlinkBench, and GearpumpBench. HiBench has a total of 19 test directions, which can be roughly divided into 6 test categories: micro, ml (machine learning), sql, graph, websearch, and streaming. We select four kinds of benchmarks from the HiBench benchmark suite, including WordCount, Sort, PageRank, and Kmeans. These workloads are easy to understand and represent a true Spark application with a wide range of applications.

Competing methods

Because of the resilient distributed dataset and localization technology of the Spark platform, there are fewer existing works to tune Spark parameters than Hadoop platform. They adopt different underlying execution mechanisms. For example, Starfish[30] is specially designed for Hadoop platform and optimized by combining with the system framework, not only parameters tuning system. For fairness, we compared MonkeyKing with two methods, including C5.0 decision tree,[31] and exist rule of thumb native Spark recommendations from industry leaders.[32] These methods are chosen because they are the state-of-the-art in the relevant areas, that is,

C5.0 is very efficient multiclassification method of traditional machine learning.

The default parameters value of native Spark is designed to afford all kinds of applications, which cannot be used to compare different applications. To have a fair comparison to MonkeyKing, we select the better parameters value for different type workloads based on gounaris2018methodology. This adaption significantly enhances the performance of native Spark. For C5.0 comparison, first, sampling was conducted on the parameter space, and 50 parameter lists were generated for each application to be used for testing. Each of parameter list was tested three times. However, the biggest problem is that in the process of collecting training data, which parameter values need to be further explored. Since the parameter space is huge, sparse sampling is selected at this time. In particular, it is necessary to search the space of different parameter subsets in detail, and to collect data evenly and randomly within the range of different.

All methods are implemented on Spark2.2.0 platform: DQN, Double DQN, Dueling DQN, and MonkeyKing are implemented using tensorflow,[33] C5.0 is taken from its authors, and parameters setting of native Spark is taken from the study of Gounaris and Torres.[32]

Evaluation metrics

In the experiment, we chose computational performance as an evaluation metric. The performance aspect is mainly reflected by comparing JCT. If JCT is greatly reduced, then MonkeyKing is proved to be available. It is judged whether MonkeyKing is valid by comparing the parameters obtained by the comparison algorithm with the default determination of parameters configuration in terms of performance improvement. For comparison of several different algorithms, we chose convergence as a criterion.

Experimental results and analysis

Parameter selection for spark. Spark currently has more than 150 configurable parameters.[34] We studied Spark's 175 configurable parameters, analyzed and summarized the 10 categories listed in Table 2, and excluded a total of 120 unneeded parameters based on the classification results. In addition, based on the adjustment recommendations of other researchers, we finally selected a candidate subset of 55 parameters.

Then, we collect the obtained 55 parameter-related parameter data as raw data samples in the historical data information and input them to the parameter selection module for different job types. After data preprocessing and feature selection, we finally get the weight of each feature, that is, the degree of impact on performance of Spark jobs. Due to space limitations, we only show the first 14 parameters as representative in Table 3.

From the results in Table 3 and Spark tuning guide recommendations,[2] we compiled 10 parameters that have the strongest impact on Spark performance as experimental objects.

Convergence test. To demonstrate the convergence of MonkeyKing, we run 10 times for 4-type application by the same workloads with the same data size on

Table 2. Classification of Spark configuration parameters

Category	*Example*	*Total*
Related to safety or recommended to be used with caution (once modified, it can cause irreparable damage.)	acls.enable	20
Related to waiting time or number before the next operation	rpc.numRetries	19
Related to application name, file name, file list, file information, save path	app.name	19
Recommended default values	memory.fraction	17
For low version or other deployment mode	shuffle.memoryFraction	9
Port, IP address, hostname, URL	ui.port	8
Related to dynamic allocation	dynamicAllocation.enabled	7
Related to whether the web UI, progress bar, and console are displayed	ui.enabled	4
Related to algorithm	io.encryption.keygen.algorithm	2
Others	files.overwrite	15
Sum	/	120

Table 3. The effect of parameters on different workloads

Parameter	*WordCount*	*Sort*	*PageRank*	*Kmeans*
driver.cores	1.432	3.107	2.008	1.417
driver.memory	3.256	2.842	3.119	2.705
executor.cores	3.558	3.034	3.284	2.988
executor.memory	2.437	2.977	2.341	2.602
default.parallelism	2.156	2.235	1.704	2.430
serializer	2.607	2.429	1.834	2.017
rdd.compress	1.093	0.998	1.024	0.968
shuffle.compress	3.647	0.826	2.034	1.287
shuffle.spill.compress	1.894	3.227	2.736	2.304
reducer.maxSizeInflight	2.808	2.360	2.489	2.542
io.compression.codec	2.001	2.048	2.164	2.019
files.useFetchCache	1.872	1.530	1.309	1.488
shuffle.file.buffer	1.526	1.249	1.007	1.103
broadcast.compress	1.122	1.046	0.857	1.114

MonkeyKing. The different applications are as follows. The sort application is 320 MB data size. The WordCount application is 3.2 GB. The PageRank application is 3.6 GB. The K-means application is 3.6 GB. Figure 5 shows the JCT for different applications by MonkeyKing in a single heterogeneous cluster. The results demonstrate that MonkeyKing achieve better convergence trend for different applications in heterogeneous environments.

Effectiveness test with different DRL algorithms. In the choice of specific algorithms, we are mainly based on DQN because it has developed into a very mature technology and contains various DQN variants. We finally chose several popular frameworks: DQN, Double DQN, Dueling DQN, and the combination of Double and Dueling DQN. The initial sample data size used in the experiment was 200. The minimum CPU utilization $u_{\min}$ we determined was 10% based on the range of CPU utilization for the actual operation of the four jobs. After using LASSO to calculate the key parameters, we separately explored the effects of training with four different DQN architectures for the optimization of these parameters. Our findings show that the combination of Double and Dueling DQN has more outstanding performance, as shown in Figure 6. DQN has the most training time, while the combination of Double DQN and Dueling DQN has the least training time.

This is because Double DQN solves the problem of overestimation in DQN, and Dueling DQN improves the network structure, so this method converges faster than the other three. The experimental results of MonkeyKing are represented by this combination algorithm.

We leverage MonkeyKing to calculate the optimal configuration of the above parameters. Then, we represent the change in performance by comparing the JCT under the parameter configuration recommended by our approach with the JCT under the default parameter configuration. Affected by limited cluster resources, we may not be able to achieve the best value recommended by the industry, we can only find a set of optimal configurations based on existing resources.

Impact analysis of different key parameters. MonkeyKing can prove that parameter tuning is important and necessary. This can be seen in the single parameter tuning results. We tuned 10 parameters to determine their impact on job performance, respectively. Figure 7 compares various JCTs achieved by MonkeyKing with 10

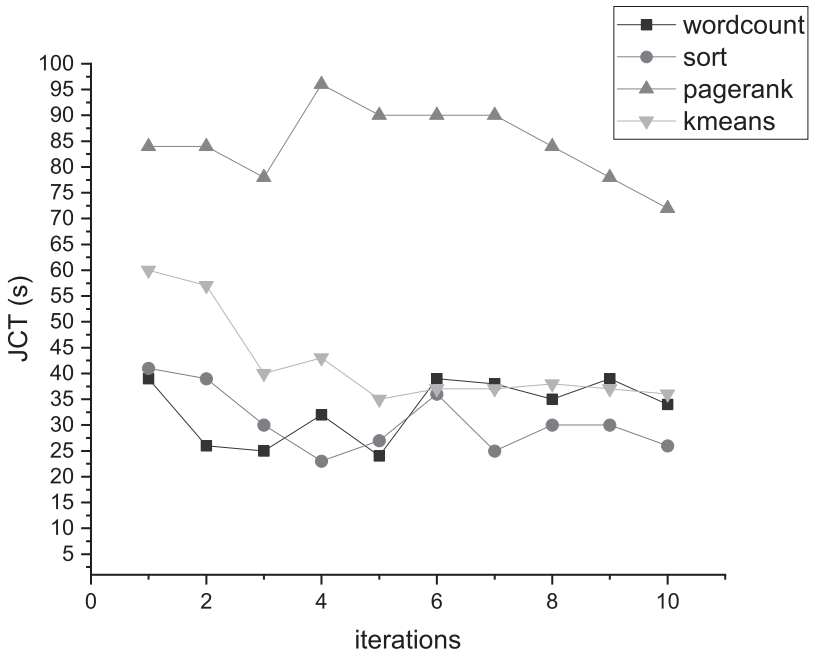

FIG. 5. Learning curves of different workloads.

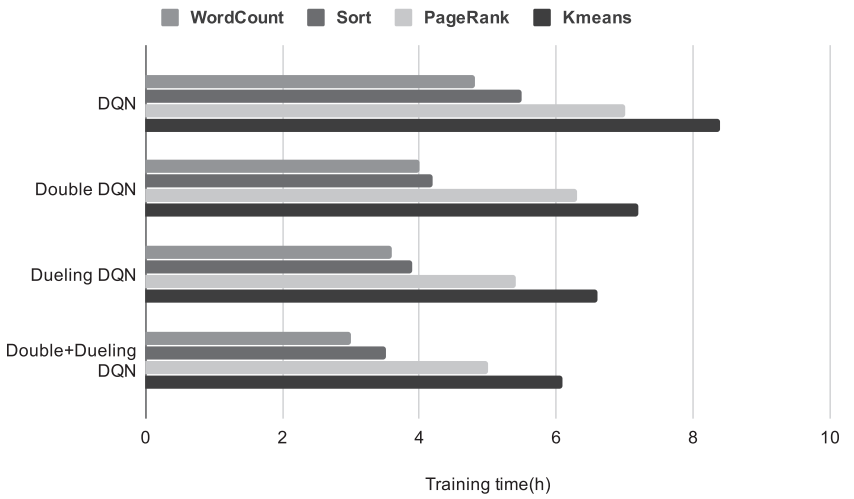

FIG. 6. Comparison of four DQN algorithms.

parameters, respectively. The Figure 7a-d shows the performance of Wordcount, Sort, PageRank, and K-means, respectively. Based on Figure 7, we further analyzed performance improvement of different parameters impacting, respectively. Figure 8 shows that parameter tuning is useful. We can see that the impact of these 10 parameters on performance is roughly in the range of 6%–36% from results. Besides, the impact of the same parameter on different types of jobs is different. The results demonstrate MonkeyKing achieve shortest training time than other DRL approaches in heterogeneous environments. This is due to the fact that the MonkeyKing adopts the double DQN and dueling DQN composition framework to speed up training time.

Effectiveness test with different approaches. We mainly study parameter combination tuning because JCT is affected by multiple parameters at the same time. Experiments show that MonkeyKing can effectively adjust the combination of parameters and can significantly reduce JCT and improve performance. The performance results of four different types of job are shown in Figure 9. Figure 9a-d shows the performance of Wordcount, Sort, PageRank, and K-means, respectively. In this part of the experiment, we used the small job size of 1.2 GB. Spark clusters are affected by various factors, so there is no guarantee that the completion time of each job is exactly the same. For this reason, we also run all comparison experiments more than 10 times and label all standard deviation in the results figures. We also analysis the standard errors for all competing approaches with different applications. And Figure 10 shows that the MonkeyKing almost achieves all the best standard errors in different applications.

We can draw a conclusion that WordCount performance increased by an average of 24.8%, Sort performance increased by an average of 19.7%, PageRank performance increased by an average of 18.5%, and Kmeans performance increased by an average of 21.7%. The results demonstrate that MonkeyKing achieve better performance improvement than other approaches in heterogeneous environments. We also find that C5.0 achieves slight performance improvement compared with the native approach. This is due to the fact that the main contribution of MonkeyKing relies on the DRL and adaptive configuration in heterogeneous environments.

Scalability of MonkeyKing. We also studied the relationship between job size and parameter tuning.[35] It can be inferred from the results that the larger the job size, the more obvious the effect of MonkeyKing performance improvement compared with the C5.0 and native Spark. Figure 11 shows the performance changes of the four benchmarks for different job sizes. In summary, with the increase of job size, the effect of performance improvement is more obvious. Figure 11a shows the optimization effect of WordCount; WordCount has 22.7% performance gain when the job size is 1 GB and 26.4% performance gain when the job size is 6 GB. Figure 11b

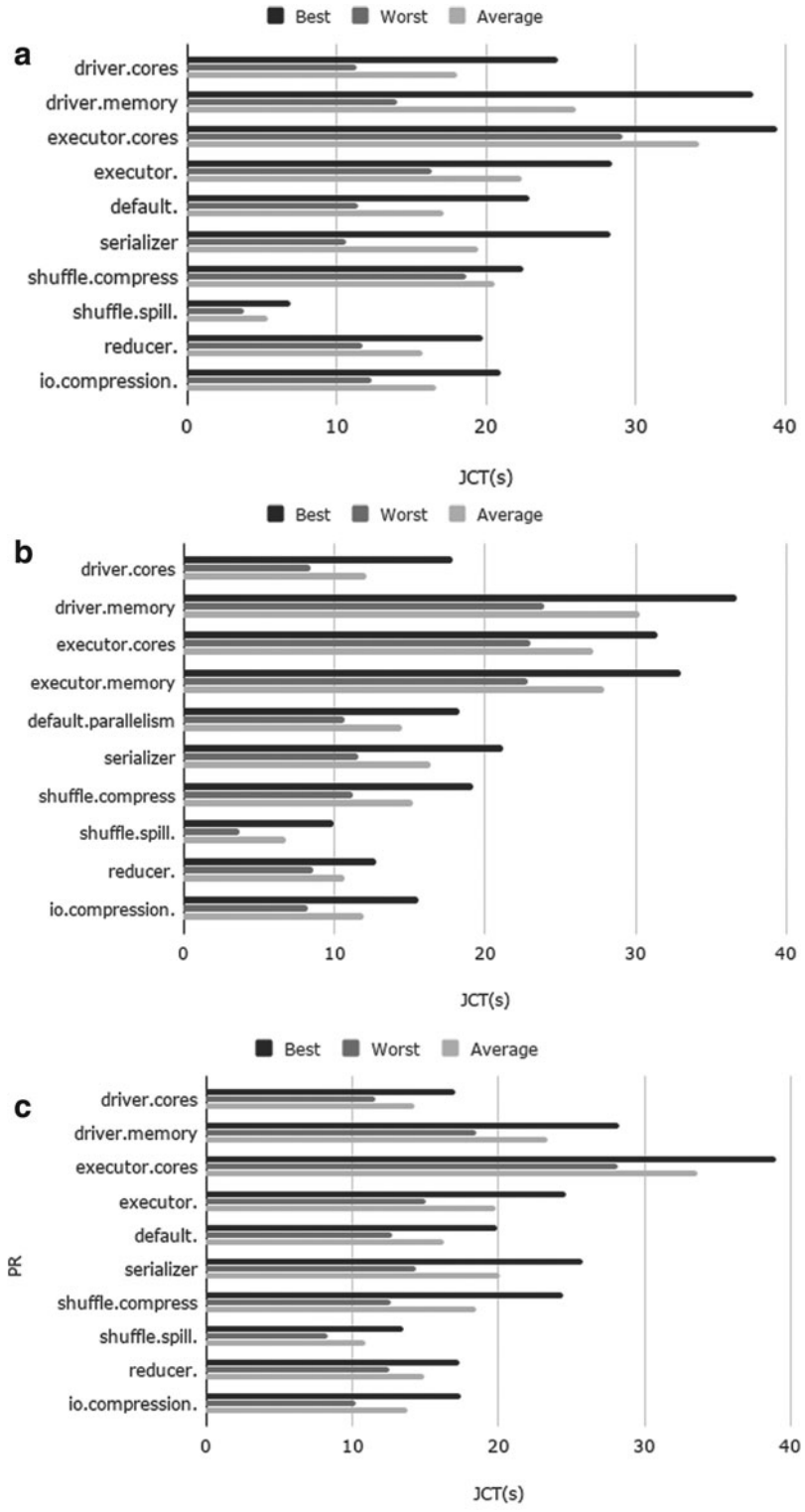

FIG. 7. **(a–d)** Performance improvement of single parameter tuning with different applications.

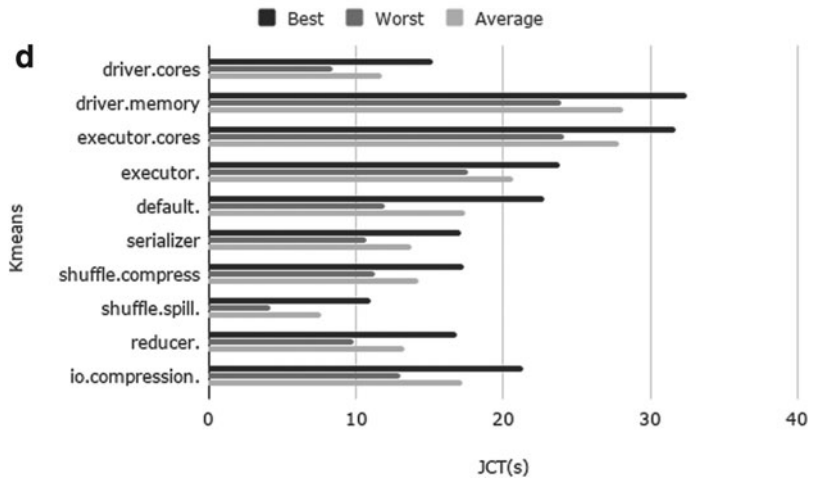

Fig. 7. (Continued).

represents the optimization effect of Sort benchmark. We can see that Sort has 20.6% performance gain when the job size is 1 GB and 25.5% performance gain when the job size is 6 GB compared with the native Spark. Figure 11c represents the optimization effect of PageRank, which illustrates that PageRank has 21.2% performance gain when the job size is 1 GB and 24.4% performance gain when the job size is 6 GB. Figure 11d shows the optimization effect of Kmeans. Kmeans has 20.7% performance gain when the job size is 1 GB and 22.9% performance gain when the job size is 6 GB. The C5.0 method also works in

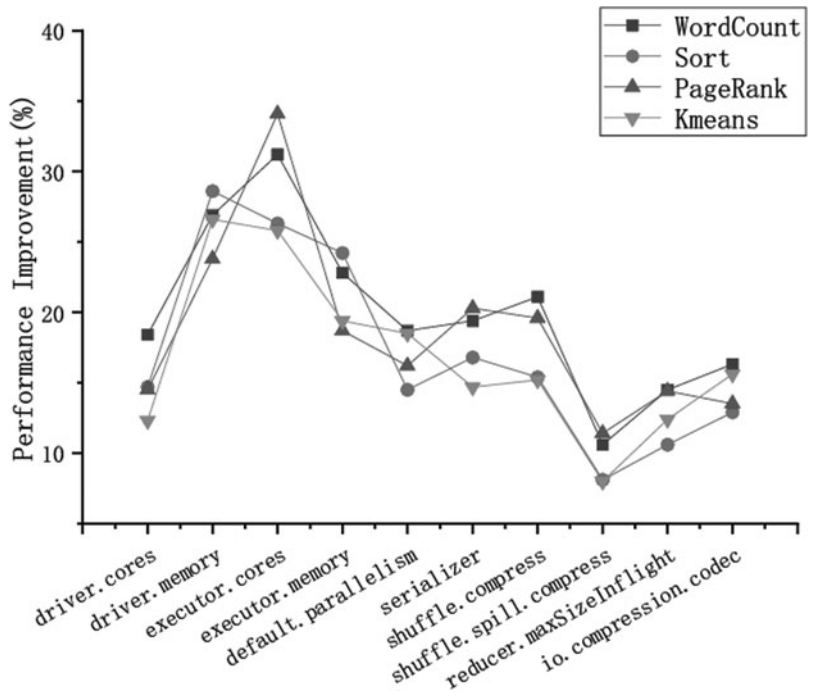

FIG. 8. Performance improvement of single parameter tuning.

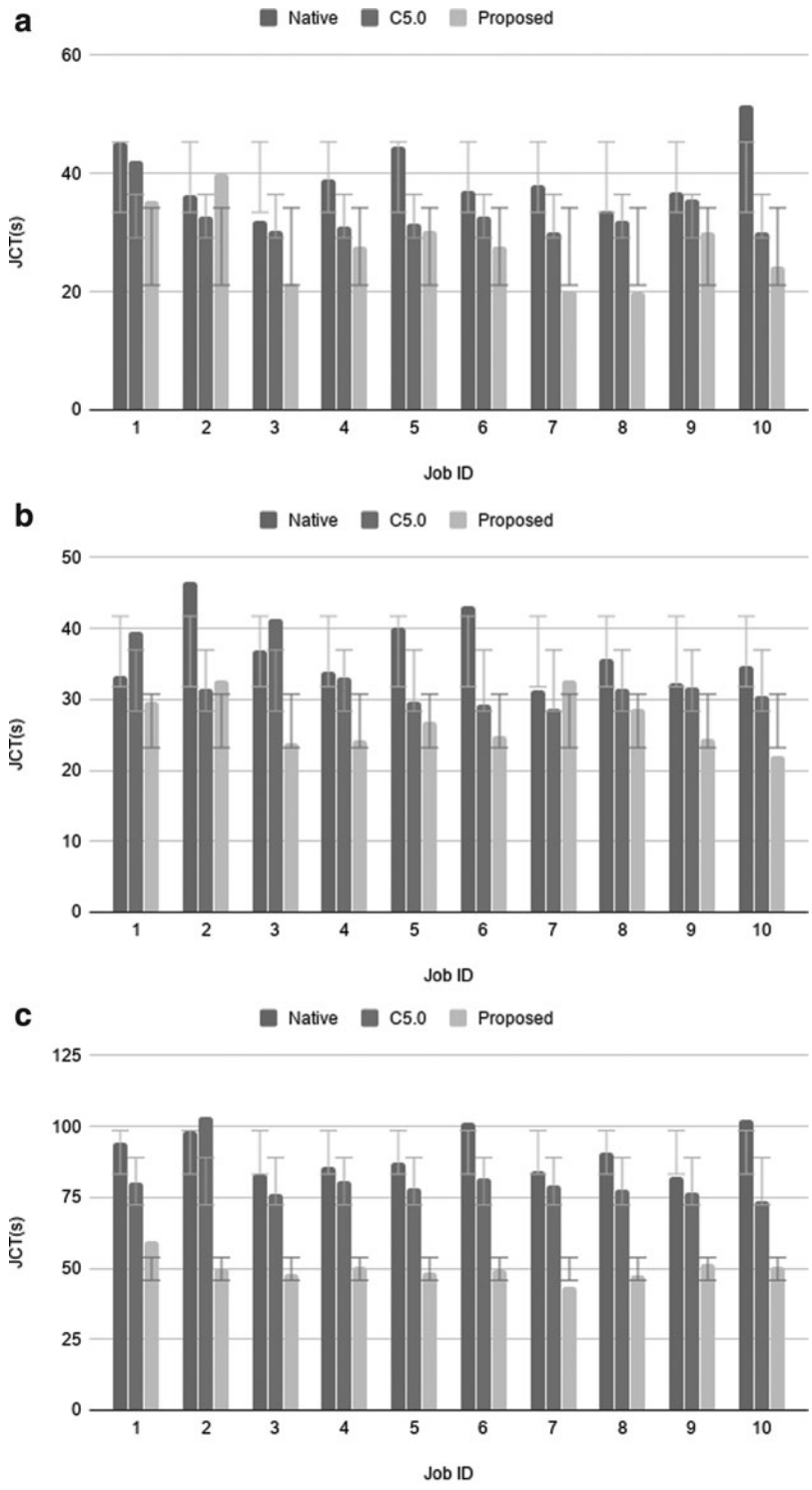

FIG. 9. **(a–d)** Performance improvement of parameter combination tuning.

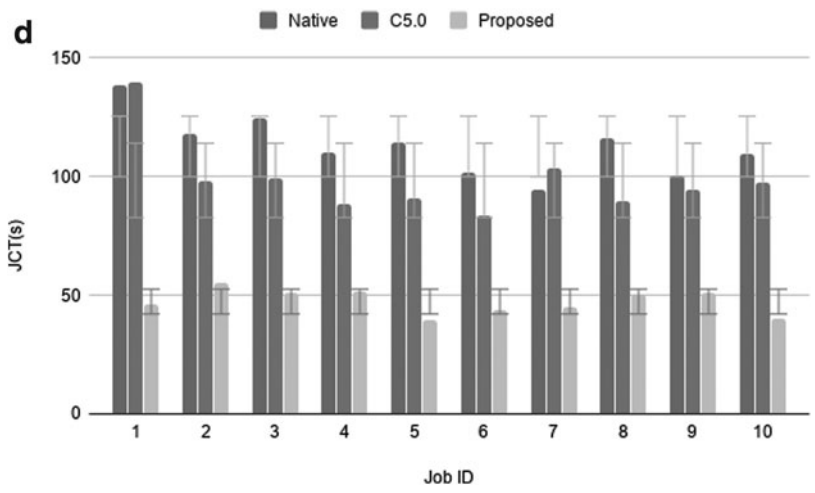

Fig. 9. (Continued).

terms of scalability, but not as well as what the MonkeyKing achieved.

Discussion

In this section, we mainly discuss the specific contributions of this article and the direction that is worth studying in the future.

Parameters tuning

Parameter tuning is a very important research task in many aspects. For example, big data analysis platform, hyperparameter tuning in AutoML, compiler parameter tuning, and Database system parameter tuning. In the current very hot research field of AutoML, some current research methods mainly use learning-based

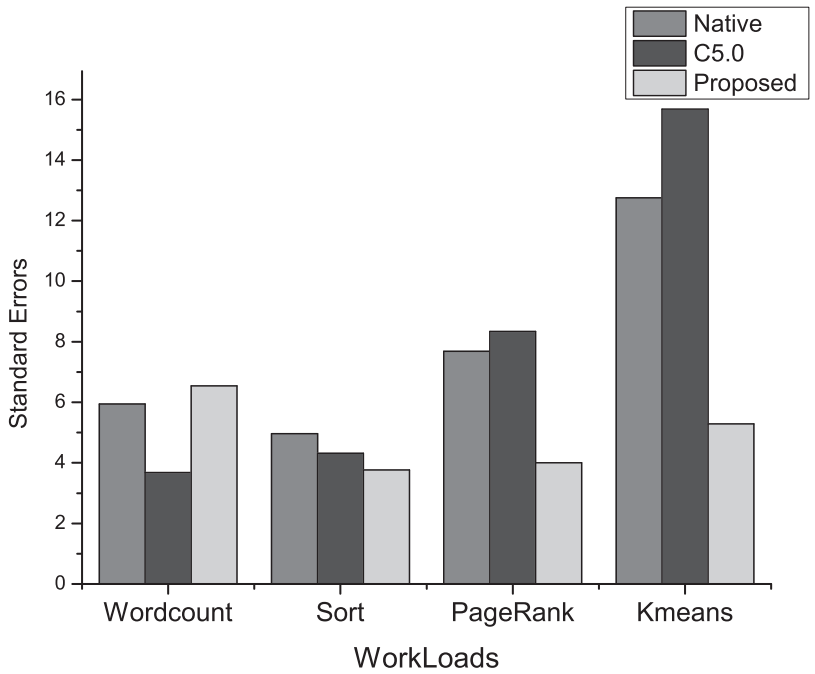

FIG. 10. Standard errors of competing approaches with different applications.

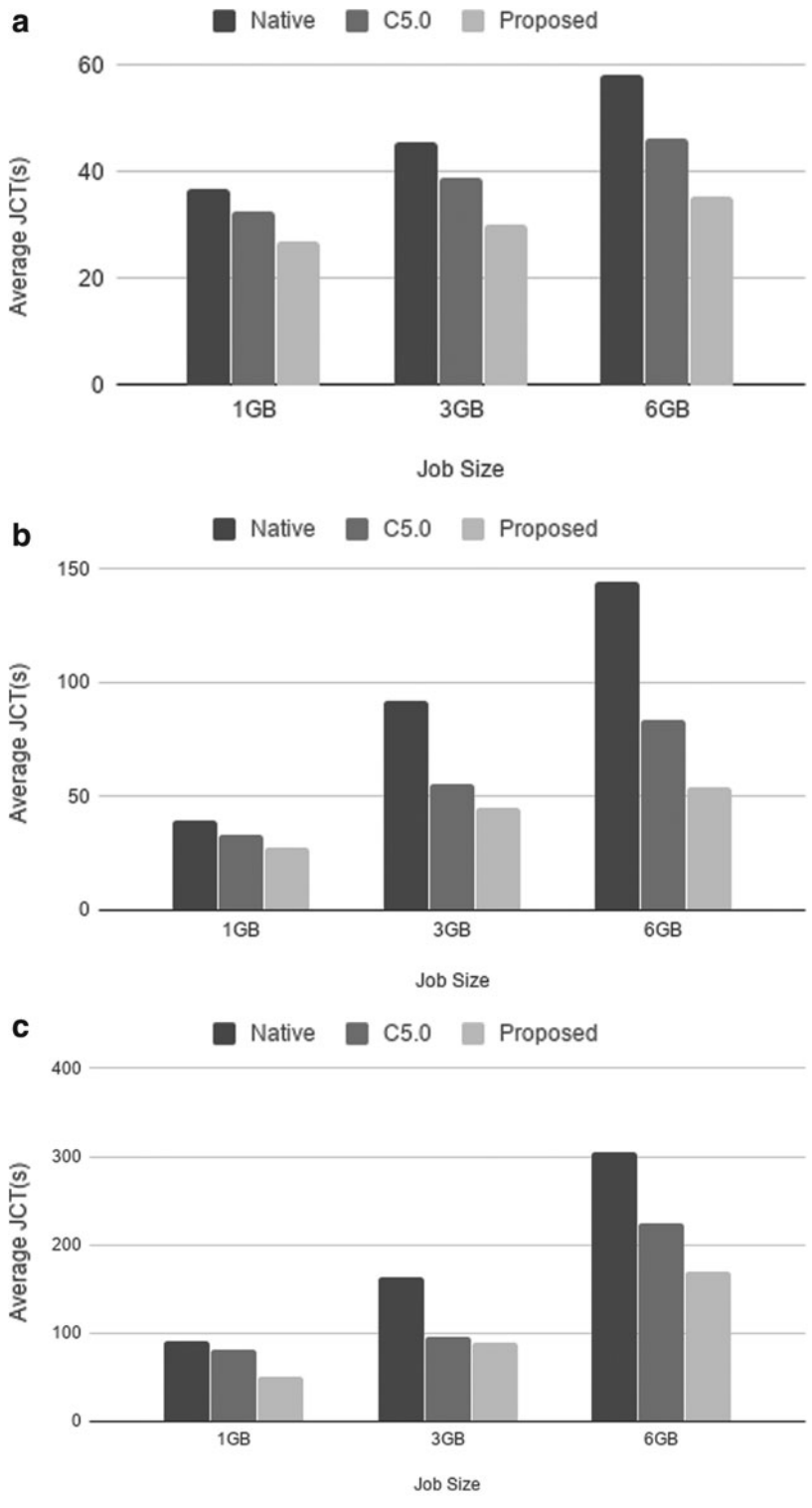

FIG. 11. **(a–d)** The optimization effect for different job size.

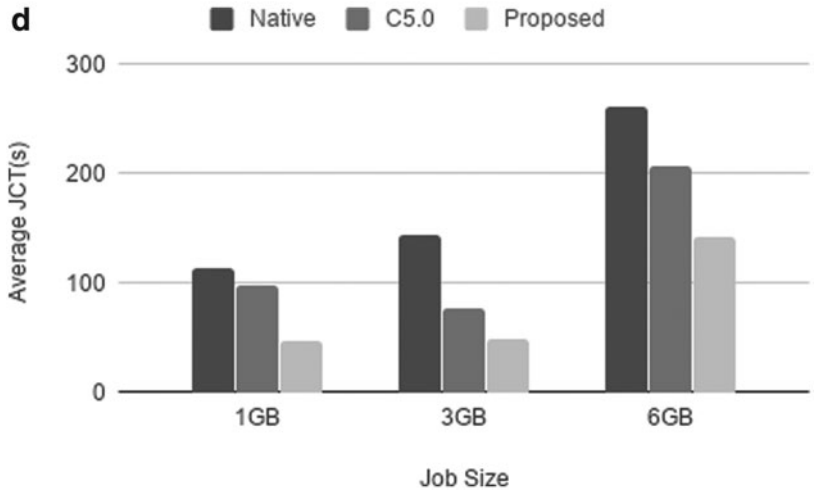

Fig. 11. (Continued).

method to study the automatic tuning of hyperparameters. Their goal optimize a learning algorithm to make the right parameters choice when faced with these network degrees of freedom.[36–38] But in the field of big data analysis platform, the goal is to optimize the whole platform performance at runtime. The challenges are different for different scenarios. How to simultaneously leverage selecting a learning algorithm and setting its hyperparameters is a big challenge in AutoML research. Hundreds of configuration parameters have high dimensionality, naive exhaustive search is not feasible is a big challenge in big data analysis platform. Therefore, we cannot use "one-size fits all" approach to solve different challenges in different scenarios.

Performance optimization

In view of the performance optimization problem of Spark platform, this article improves from the perspective of parameter tuning and proposes a system called MonkeyKing based on DRL algorithm. Although the use of RL to solve optimization problems is not new, and even some researchers have made good progress on parameter optimization problems, the existing use of RL techniques to study parameter problems focuses on optimizing the hyperparameters of machine learning algorithms. The number of configuration parameters of Spark is not only hundreds but also includes different types. To this end, we define the Markov Decision Process that conforms to the Spark parameter optimization scenario.

Performance modeling

Specifically, we divide MonkeyKing into two key modules: parameter selection and parameter tuning. Compared with previous studies, in addition to applying RL techniques to solve Spark performance tuning, we have also added parameter selection work. Considering parameter selection is because other parameter optimization problems (such as hyperparameter optimization) do not involve a lot of parameters, and unlike Spark platform, only individual parameters will affect performance. At the same time, we also considered job type, obviously different types of jobs have different optimal configurations. In the parameter selection module, with the help of LASSO algorithm, we can recommend different parameter configurations for different types of jobs. In the parameter tuning module, we mainly apply the mature DQN framework in DRL technology. We have studied four frameworks: DQN, Double DQN, Dueling DQN, and the combination of Double DQN and Dueling DQN. The results show that the combination of Double DQN and Dueling DQN has better convergence.

Scalable space

We are pleased to apply DRL to Spark's parameter tuning and achieve acceptable results. It is worth mentioning that this technology can also be extended to the

parameter optimization of other big data analysis platforms, such as Hadoop and Storm. Although the structure of each platform is not exactly the same, there are differences in parameters, the ideas provided by MonkeyKing are universal. Although MonkeyKing provides great convenience for our parameter tuning work, there are still some imperfections. For example, the system training time is too long, and the convergence of DQN needs to be improved. These issues are the work that we will study now and for some time to come.

Related Work

Much of the previous work on auto-tuning on big data analytics platform focused on choosing the best cost or performance design, which we describe in turn. The popularity of big data analytics platforms has made big data analysis fast and efficient. However, many default resource configurations are no longer sufficient for program developers. Therefore, many researchers have analyzed parameters to improve performance by modifying the configuration. At present, the parameter optimization of big data analytics platforms can be summarized in the following three aspects.

Cost-based optimization

Cost-based optimization refers to the performance cost required by calculating the various stages of task execution process, and the corresponding optimization scheme is obtained. Starfish[30] is a self-tuning system, which is based on cost modeling to search for the job configurations required for MapReduce workloads. But Starfish only works for Hadoop, and the mechanisms between other parallel distributed platforms and Hadoop are quite different. SBAC-PAD 2016[39] presented a novel Impala simulation framework that simulates the behavior of a complete software stack and the activities of cluster components (such as storage, networking, processors, and memory) using a shared-nothing parallel database architecture. Impala aims to bridge the gap between near real-time data analysis on the Hadoop stack to help IT professionals understand their performance behavior. AROMA[40] automates job configuration and resource allocation through leveraging a two-phase machine learning and optimization framework for heterogeneous clouds.

Performance-based optimization

Performance-based optimization is the monitoring of performance data or other data. By analyzing the monitoring results, performance bottleneck is obtained, and then, performance can be optimized. Alipourfard et al.[4] proposed CherryPick, a system that can adaptively mine the best cloud configuration for big data analysis, using Bayesian optimization to build performance models for various applications. While CherryPick is ideal for resource allocation issues for large applications. Lee and Jos[41] proposed an approach that avoids problems of previous self-tuning approaches based on performance models or resource usage; the proposed approach uses a fuzzy-prediction controller for self-optimization of the number of concurrent MR jobs. Shi et al.[42] discussed the impact of MapReduce and Spark on large-scale data analysis, where the impact of parameters on the platform is not much. Petridis et al.[43] studied some related parameters that have a significant impact on Shuffle and Compress phases of Spark and proposed a parameter tuning method based on trial and error. Bao et al.[44] used Latin hypercube sampling to generate effective samples in the high-dimensional parameter space, and multiple bound-and-search to select promising configurations in the bounded space suggested by the existing best configurations. Yigitbasi et al.[45] used the support vector regression model on Hadoop platform for auto-tuning. Although the algorithm works well, it is obvious that the trial and error method is time-consuming. Wang et al.[46] analyzed the shortcomings of cost-based modeling methods and proposed a new method based on machine learning to optimize Spark configuration. Wang et al.[47] proposed a speculative parallel decompression algorithm based on Apache Spark to extend parallelism and improve the decompression efficiency of large-scale data sets.

Heuristic-based optimization

The heuristic-based optimization method mimics the way machine learning is done, storing some good configurations in a certain number of experiments that have been run in the optimizer. Gopalan and Suresh[48] proposed an improved Hadoop Fair Scheduler delay scheduling and implementation in Hadoop. The proposed algorithm does not blindly wait for the local node, but first estimates the time to wait for the local node to use for job, and avoids waiting if the location is not possible within the predefined delay threshold while completing the same location. The authors present a heuristic approach to reducing the operational costs of virtual machines running Hadoop in the study of Shyamasundar et al.[49] Heuristics are simple and efficient, extending the number of Hadoop nodes based on the type and size of jobs submitted.

Conclusion

In this article, to solve the problems of key parameter selection and tuning for different job types of big data platforms and reuse of historical information, we propose a system called MonkeyKing. MonkeyKing mainly includes three parts: parameter selection, parameter tuning, and historical information base. First, the feature selection technique is used in parameter selection module to determine the parameters that have the strongest impact on performance of jobs, and then, DRL algorithms are selected in parameter tuning module to dynamically optimize parameters. At the same time, the historical information base will save the running information of jobs for subsequent reuse. In terms of the selection of key parameters, we conduct research on different job types, enabling our method to recommend relevant key parameters for different workloads. In terms of parameter tuning, we choose DQN structure and its four classic algorithms. We finally found that the combination of Double DQN and Dueling DQN is more convergent and the obtained parameter optimal value is more stable. The experimental results show that compared with the C5.0 and native parameter configuration, the recommended parameter configuration of MonkeyKing can effectively reduce JCT by $\sim$25%.

Author Disclosure Statement

No competing financial interests exist.

Funding Information

No funding was received for this article.

References

1. Hadoop. Apache hadoop. Available online at http://hadoop.apache.org/ (last accessed September 12, 2019).
2. Spark. Apache spark. Available online at http://spark.apache.org/ (last accessed September 18, 2019).
3. Storm. Apache storm. Available online at http://storm.apache.org/ (last accessed August 16, 2019).
4. Alipourfard O, Liu HH, Chen J, et al. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), Boston, MA: USENIX Association, 2017. pp. 469–482.
5. Gounaris A, Torres J. A methodology for spark parameter tuning. Big Data Res. 2017; DOI: 10.1016/j.bdr.2017.05.001.
6. Wang H, Cao Y. An energy efficiency optimization and control model for hadoop clusters. IEEE Access. 7:2019;40534–40549.
7. 4th International Conference on Advanced Technologies for Signal and Image Processing, ATSIP 2018, Sousse, Tunisia, March 21–24, 2018. IEEE, 2018.
8. Zhang F, Zhai J, Snir M, et al. (Eds): Network and Parallel Computing—15th IFIP WG 10.3 International Conference, NPC 2018, Muroran, Japan, November 29–December 1, 2018, Proceedings, volume 11276 of Lecture Notes in Computer Science. Springer, 2018.
9. Krasheninnikova E, García J, Maestre R, Fernández F. Reinforcement learning for pricing strategy optimization in the insurance industry. EngAppl AI. 2019;80:8–19.
10. Mnih V, Kavukcuoglu K, Silver D, et al. Riedmiller. Playing atari with deep reinforcement learning. CoRR. 2013;abs/1312.5602.
11. Hosu I-A, Rebedea T. Playing atari games with deep reinforcement learning and human checkpoint replay. arXiv preprint. 2016;arXiv: 1607.05077.
12. Silver D, Huang A, Maddison CJ, et al. Mastering the game of go with deep neural networks and tree search. Nature. 2016;529:484–489.
13. Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning. Comput Sci. 2015;arXiv:1509.06461.
14. Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for deep reinforcement learning. arXiv preprint. 2015;arXiv:1511.06581.
15. Huang S, Jie H, Dai J, et al. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In: IEEE International Conference on Data Engineering Workshops, 2010.
16. Du H, Han P, Chen W, et al. Otterman: A novel approach of spark autotuning by a hybrid strategy. In: 2018 5th International Conference on Systems and Informatics (ICSAI), November 2018. pp. 478–483.
17. Ferguson AD, Bodik P, Kandula S, et al. Jockey: Guaranteed job latency in data parallel clusters. In: European Conference on Computer Systems, 2012.
18. Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature. 2015;518:529–533.
19. Chen L, Lingys J, Chen K, Liu F. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In: SIGCOMM'18, August 20–25, 2018, Budapest, Hungary, 2018.
20. Hansen S. Using deep q-learning to control optimization hyperparameters. arXiv preprint. 2016;arXiv:1602.04062.
21. Ma S, Ilyushkin A, Stegehuis A, Iosup A. Ananke: A q-learning-based portfolio scheduler for complex industrial workflows. In: 2017 IEEE International Conference on Autonomic Computing, 2017.
22. Yan L, Chang K, Bel O, et al. Long. Capes: Unsupervised storage performance tuning using neural network-based deep reinforcement learning. In: International Conference for High Performance Computing, 2017.
23. Rodríguez P, Bautista MÁ, Gonzàlez J, Escalera S. Beyond one-hot encoding: Lower dimensional target embedding. CoRR 2018; abs/1806.10805.
24. Islam MR, Koh YS, Zhao Y, et al. Data Mining—16th Australasian Conference, AusDM 2018, Bahrurst, NSW, Australia, November 28–30, 2018, Revised Selected Papers, volume 996 of Communications in Computer and Information Science. Springer, 2019.
25. Du X, Zhu F. A novel principal components analysis (PCA) method for energy absorbing structural design enhanced by data mining. Adv Eng Softw 2019;127:17–27.
26. Salihoglu S, Zhou W, Chirkova R, et al. Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14–19, 2017. ACM, 2017.
27. Fraley C, Hesterberg T. Least angle regression and LASSO for large datasets. Stat Anal Data Min 2009;1:251–259.
28. Tibshirani RJ, Rinaldo A, Tibshirani R, et al. Uniform asymptotic inference and the bootstrap after model selection. Ann Stat 2018;46: 1255–1287.
29. Samadi Y, Zbakh M, Tadonki C. Performance comparison between hadoop and spark frameworks using hibench benchmarks. Concurr Comput Pract Exp 2018;30:e4367.
30. Herodotou H, Lim H, Luo G, et al. Starfish: A self-tuning system for big data analytics. In: CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9–12, 2011, Online Proceedings, 2011. pp. 261–272.
31. Wang G, Xu J, He B. A novel method for tuning configuration parameters of spark based on machine learning. In: 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 586–593. IEEE, 2016.
32. Gounaris A, Torres J. A methodology for spark parameter tuning. Big Data Res 2018;11:22–32.

33. Abadi M, Barham P, Chen J, et al. Tensorflow: A system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), 2016. pp. 265–283.
34. Wang Z, Zhao Y, Liu Y, Lv C. A speculative parallel simulated annealing algorithm based on apache spark. Concurr Comput Pract Exp 2018;26: e4429.
35. Awan AJ, Brorsson M, Vlassov V, Ayguade E. How data volume affects spark based data analytics on a scale-up server. In: The Workshop on Big Data Benchmarks, 2015. pp. 81–92.
36. Feurer M, Klein A, Eggensperger K, et al. Efficient and robust automated machine learning. Adv Neural Inf Process Syst, pp. 2962–2970, 2015.
37. Thornton C, Hutter F, Hoos HH, Leyton-Brown K. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, 2013. pp. 847–855.
38. Xu T, Liu Q, Zhao L, Peng J. Learning to explore via meta-policy gradient. In: International Conference on Machine Learning, 2018. pp. 5463–5472.
39. 28th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2016, Los Angeles, CA, USA, October 26–28, 2016. IEEE Computer Society, 2016.
40. Lama P, Zhou X. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In: International Conference on Autonomic Computing, 2012. pp. 63–72.
41. Lee GJ, Jos AB. Fortes. Hadoop performance self-tuning using a fuzzy-prediction approach. In: 2016 IEEE International Conference on Autonomic Computing (ICAC) IEEE, 2016, 2016.
42. Shi J, Qiu Y, Minhas UF, et al. Clash of the titans: Mapreduce vs. spark for large scale data analytics. Proc Vldb Endowment 2015;8:2110–2121.
43. Petridis P, Gounaris A, Torres J. Spark parameter tuning via trial-and-error. In: INNS Conference on Big Data, 2016. pp. 226–237.
44. Bao L, Liu X, Chen W. Learning-based automatic parameter tuning for big data analytics frameworks. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 181–190. IEEE, 2018.
45. Yigitbasi N, Willke TL, Liao G, Epema D. Towards machine learning-based auto-tuning of mapreduce. In: 2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 11–20. IEEE, 2013.
46. Wang G, Xu J, He B. A novel method for tuning configuration parameters of spark based on machine learning. In: IEEE International Conference on High Performance Computing and Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science and Systems, 2017.
47. Wang Z, Zhao Y, Liu Y, et al. A speculative parallel decompression algorithm on apache spark. J Supercomput 2017;73:1–30.
48. Gopalan NP, Suresh S. Modified delay scheduling: A heuristic approach for hadoop scheduling to improve fairness and response time. Parallel Process Lett 2015;25.
49. Shyamasundar RK, Shastri L, Janakiram D, Padmanabhuni S (Eds): Proceedings of the 6th ACM India Computing Convention, COMPUTE 2013, Vellore, Tamil Nadu, India, August 22–24, 2013. ACM, 2013.

Cite this article as: Du H, Han P, Xiang Q, Huang S (2020) Monkey-King: adaptive parameter tuning on big data platforms with deep reinforcement learning. *Big Data* 8:4, 270–290, DOI: 10.1089/big.2019.0123.

Abbreviations Used

DL = deep learning
DNN = deep neural network
DQN = deep Q-network
DRL = deep reinforcement learning
JCT = job completion time
OLS = ordinary least squares
PCA = principal components analysis
RL = reinforcement learning