Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/mbe

MBE, 20(8): 14464–14486.
DOI: 10.3934/mbe.2023647
Received: 05 March 2023
Revised: 07 June 2023
Accepted: 20 June 2023
Published: 03 July 2023

Research article

An innovative parameter optimization of Spark Streaming based on D3QN with Gaussian process regression

Hong Zhang[1], **Zhenchao Xu**[1*], **Yunxiang Wang**[1] **and Yupeng Shen**[2]

[1] School of Cyber Security and Computer, Hebei University, Baoding, China
[2] Bureau of Geophysical Prospecting, Baoding, China

* **Correspondence:** Email: xuzhenchao@stumail.hbu.edu.cn.

Abstract: Nowadays, Spark Streaming, a computing framework based on Spark, is widely used to process streaming data such as social media data, IoT sensor data or web logs. Due to the extensive utilization of streaming media data analysis, performance optimization for Spark Streaming has gradually developed into a popular research topic. Several methods for enhancing Spark Streaming's performance include task scheduling, resource allocation and data skew optimization, which primarily focus on how to manually tune the parameter configuration. However, it is indeed very challenging and inefficient to adjust more than 200 parameters by means of continuous debugging. In this paper, we propose an improved dueling double deep Q-network (DQN) technique for parameter tuning, which can significantly improve the performance of Spark Streaming. This approach fuses reinforcement learning and Gaussian process regression to cut down on the number of iterations and speed convergence dramatically. The experimental results demonstrate that the performance of the dueling double DQN method with Gaussian process regression can be enhanced by up to 30.24%.

Keywords: Spark Streaming; Gaussian process regression; dueling double DQN; parameter optimization

1. Introduction

Due to big data's phenomenal growth, the demand for real-time big data processing and related applications is rising quickly. Storm [1], Spark Streaming [2] and Flink [3] are the most popular computing frameworks for streaming data. With a complete ecosystem and more cutting-edge technologies, Spark Streaming, an extension API of Spark, has become the industry standard for processing and analyzing streaming data. It can facilitate the rapid processing of big datasets at a high level and handle the real-time processing of billions of records. At the same time, Spark Streaming offers a ton of flexibility, supporting a variety of data sources, including Apache Kafka and Flume, and it has a wide range of output targets, such as HDFS, HBase and BI reports. The execution process of Spark Streaming is shown in Figure 1.

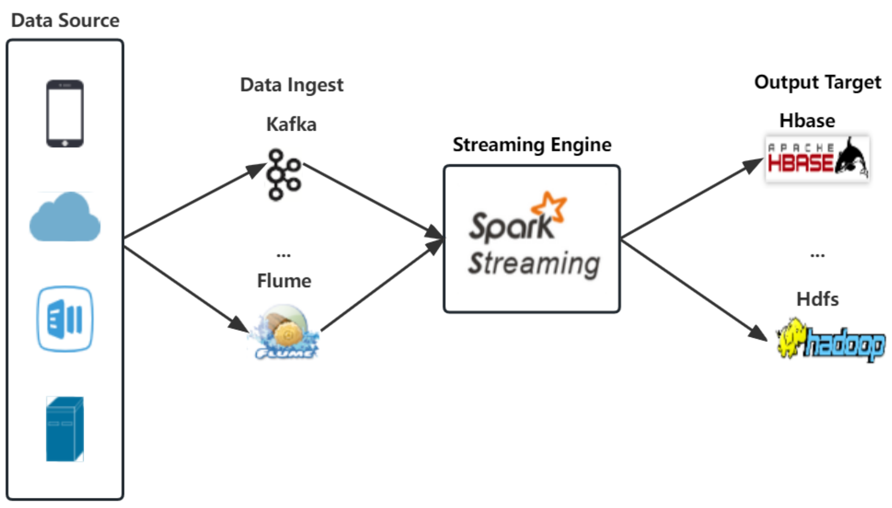

Figure 1. Execution process for Spark Streaming.

Nevertheless, Spark Streaming cannot execute at its maximum potential because of an irrational parameter configuration. The appropriate adjustment of Spark's parameters is thus a significant area of research to improve Spark Streaming's performance. Despite the fact that users can manually configure parameters to improve the performance of applications, this is obviously incredibly inefficient and would really be counterproductive owing to the complex influence among them. Several experts have so far conducted research in this field by employing reinforcement learning [4–7], but relying solely on reinforcement learning would lead to problems of incessant iterative training and poor convergence.

For such problems, we can choose appropriate methods for function approximation during the iterative process of reinforcement learning to accelerate convergence and improve learning effectiveness. One of them, Gaussian process regression, has a number of benefits over other methods [8]. First off, Gaussian process regression is a non-parametric model, allowing it to flexibly adapt to different functional forms in model selection without the requirement to predetermine the number and types of parameters. Second, a confidence range can be specified for each predicted value to estimate the degree of prediction uncertainty, which is crucial to making a correct judgment and remarkably decreasing the searching space. Additionally, Gaussian process regression improves the efficiency of parameter tuning through a combined probability distribution based on priors and data. Since gathering data for reinforcement learning is a time and resource intensive procedure, speeding up the tuning efficiency can dramatically increase the algorithm's availability in real-world applications. Finally, Gaussian process regression enables us to incorporate previous knowledge into the modeling process, such as prior distribution and kernel function selection, to further avoid unnecessary parameter detection.

To solve the aforementioned challenges, this paper introduces an improved dueling double deep Q-Network (D3QN) [9, 10] strategy for Spark Streaming parameter optimization that fuses reinforcement learning [11, 12] and Gaussian process regression [13]. When a program is running, reinforcement learning is added to the execution process of the program. Different parameter values are set for the program during each iteration, and then the parameter configuration is continuously optimized according to the obtained Q value. However, this leads to the problem of too long training times and slow convergence due to a large action space. In this paper, Gaussian process regression is added to reinforcement learning, and the Q value obtained each time is modified by Gaussian process regression to reduce the time of reinforcement learning training and accelerate convergence. The following describes this paper's main contributions:

- In this paper, we employ the batch gradient descent algorithm of the Gaussian process regression to update the Q value of reinforcement learning. The deviation problem caused by the noise of a single sample is avoided in this study by computing the gradient of a batch of samples in each iteration. Moreover, the vectorization computation of the batch gradient technique can accelerate the convergence process when training.
- In this paper, an improved D3QN-Gaussian process regression (GPR) is introduced to the Spark Streaming program to dynamically select the optimal parameters. To determine whether to reward or punish depending on the outcomes of each program execution, D3QN with Gaussian process regression can perform well on small datasets and also support uncertainty measurements for prediction.
- Experimental results on Hibench, a big data benchmark platform with various applications, reveal the effectiveness and convergence of our improved reinforcement learning model for Spark Streaming.

The remainder of this paper is structured as follows. In Section 2, we describe the theoretical concepts of reinforcement learning and summarize the related work. In Section 3, we discuss the design of the enhanced reinforcement learning model D3QN for the deep Q-Network (DQN) and introduce the Gaussian process regression method. On this basis, we propose an improved D3QN-GPR model that combines Gaussian process regression with D3QN to solve parameter configuration problems. The experimental results are analyzed in Section 4. Finally, in Section 5, the paper is concluded and future work is discussed.

2. Background

In this section, we first describe the theoretical concept and the execution of reinforcement learning in this part. Then, a summary of the present Spark Streaming research hotspots is presented.

2.1. Reinforcement learning

One of the machine learning techniques, reinforcement learning [14], is distinguished by its interactive learning approach [15]. It is used to define and address the problem of how an agent can optimize its object or achieve a specific goal by using learning strategies while interacting with the environment.

In reinforcement learning, the agent learns through a trial-and-error process. It starts with little or no prior knowledge about the environment and gradually improves its decision-making abilities through interactions. The agent explores the environment, takes actions, receives feedback in the form of rewards or penalties and adjusts its future actions based on this feedback. The goal is to find an optimal policy that maximizes the cumulative reward during iterations.

The reinforcement learning process involves four key components:

- State Space (S): It represents the set of possible states that the agent can occupy. Each state captures relevant information about the environment at a specific time.
- Action Space (A): It denotes the set of possible actions that the agent can take. An action is chosen based on the current state and is aimed at influencing the future states of the environment.
- State Transition Function (P): It defines the probability distribution of transitioning from one state to another based on the agent's actions. Given the current state and action, the state transition function provides the next state.
- Reward Function (R): It assigns a numerical reward or penalty to the agent based on the outcome of its action in a particular state. The reward function guides the agent towards desired behaviors and goals.

The reinforcement learning process can be summarized as follows:

- Step 1: the agent observes the current state of the environment, denoted as $S_t \in S$, at time t.
- Step 2: based on the current state, the agent selects an action $A_t \in A$ according to a specific policy.
- Step 3: the action affects the environment, resulting in a transition to a new state S_{t+1} based on the state transition function P.
- Step 4: the environment provides a reward R_t to the agent based on the outcome of the action.
- Step 5: the agent updates its knowledge and adjusts its future actions based on the received reward and the observed state transition.
- Step 6: the aforementioned steps are executed repeatedly until the agent receives the optimal objective or completes its intended goal.

Through this iterative process of exploration and exploitation, a reinforcement learning agent can learn an effective strategy to make better decisions in complex and dynamic environments. The execution flow chart of reinforcement learning is illustrated in Figure 2.

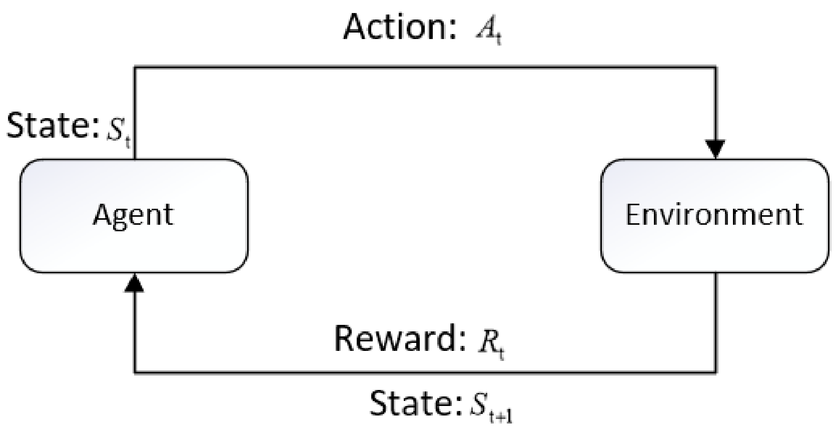

Figure 2. Reinforcement learning process.

2.2. *Related work*

A generic distributed streaming data computing framework is Spark Streaming. Because this computer platform has become more widely used, its performance has come under increasing scrutiny. A popular area of research is how to improve Spark Streaming's performance. Currently, schedule optimization, parameter optimization and data skew processing are the key components of Spark Streaming optimization.

Scheduling optimization refers to the Spark Streaming framework's dynamic scheduling of tasks in accordance with various job requirements in order to boost performance. Lin et al. [16] proposed a method of modeling and simulation to predict the response time of Spark Streaming programs; they configured the parameters to address the issue of resource over provisioning and poor performance. This allowed stream processing system throughput and resource requirements to be determined in advance. The drizzle system was created by Venkataraman et al. [17] optimized the scheduling of flow processing workload and other control plane operations for high throughput and minimal delay. It is nonetheless restricted to the infrequent changes in workload and cluster characteristics. On the Spark Streaming system, Ajila and Majumdar [18] implemented the data-driven priority scheduler prototype. Users can give input data items a priority this way. The scheduler ensures that higher priority data items take precedence over lower priority data items, minimizing delays for high priority data items and lessening the effects of resource limitations. Cheng et al. [4] used the online parameter adjustment problem of Spark Streaming as a reinforcement learning process to accomplish adaptive parameter configuration of the workload, although the issue of reinforcement learning's delayed convergence is still present. To increase the effectiveness of resource allocation Petrov et al. [19] suggested an adaptive performance model that may dynamically grow the Spark Streaming platform on AWS. Without taking into account throughput, another crucial element of the streaming computing framework, this model increases and decreases the system's resources in accordance with the length of the processing delay. Li et al. [20] used the proposed sparse regularized ADMM algorithm and the non-intrusive method, adding a new DStream transformation to the current Spark flow framework to support flexible micro batches, realizing the optimization of scheduling tasks only by altering the application workflow. On the basis of the Spark Streaming architecture, Zhao et al. [21] developed a traffic flow controller based on a grey prediction model, which is used for the batch interval dynamic adjustment layer and the flow prediction-orientated prediction layer in Spark Streaming. This meets the application throughput needs and reduces end-to-end delay. A dynamic resource allocation mechanism for many applications was proposed by Liu et al. [22]. It lowers the delay of Spark Streaming and increases the usage of cluster resources by controlling the allocation of dynamic resources by adding application global variables.

Many systems must take into account the optimization direction known as parameter optimization. Users typically utilize the default parameter configuration for the Spark Streaming computing framework, which saves time. However, using the default configuration hinders performance, especially when many apps have varied setup requirements. Additionally, there are specific correlations between each of the more than 200 factors that make up Spark Streaming. However, manual parameter tuning frequently relies on expert experience, which has several drawbacks. As a result, automating parameter configuration is a key component of improving Spark Streaming speed.

In order to dynamically plan batch processing processes in Spark Streaming and automatically alter parameters to optimize performance and decrease energy consumption, Cheng et al. [23] proposed the A-scheduler scheduling method. The Monkeyking system, developed by Du et al. [5], can optimize parameter configuration by implementing reinforcement learning after adjusting parameters using historical data and new execution outcome data. The DQN convergence problem has not been resolved, though, and the system's training period is excessively long. Before streaming media applications were ever deployed, Prasad and Agarwal [24] proposed a linear regression model that could predict how well they would work based on a particular parameter combination. They just pay attention to the block interval, batch processing and data processing parallelism factors. Kordelas et al. [6] proposed an enhanced solution, known as KORDI (knowledge based orchestrated resource distribution), aimed at optimizing the allocation of Spark resources for streaming, but it is only applicable to real-time applications of SARIMAX models. Deep reinforcement learning-based parameter optimization for Spark Streaming was proposed by Liu et al. [7] By integrating the weight state space transfer method and the reinforcement learning method, this method enhances performance. However, without taking into account the throughput of the stream computing architecture, the primary objective of this paper is to decrease latency.

Big data processing systems frequently encounter the issue of data skew. It leads to task pileup and negatively impacts system performance if it is not handled properly. The Spark stream structure similarly struggles with the issue of skewed data. A brand-new partitioning technique called SP-partitioner was put out by Liu et al. [25] It balances the workload of activities and solves the issue of data skew that arises when applications are in the shuffle stage. This approach decreases the average processing time of a batch of data while simultaneously enhancing its performance at various levels. An improved range divider (ImRP), put out by Fu et al. [26], develops a partitioning scheme using the intermediate data distribution determined by the prior batch of processing as opposed to pre-run sampling. This partitioning strategy can equalize the burden across tasks and lessen the skew of jobs in Spark Streaming.

3. Methods

In the first part of this section, we develop a reinforcement learning model based on the features of the Spark Streaming program. Then, we describe the theory of Gaussian process regression, discuss how to use it for regression analysis and provide examples of when it can be effectively used for reinforcement learning. Finally, we implement the D3QN-GPR model by integrating these two.

3.1. Reinforcement learning model design

There are several implementation strategies for reinforcement learning in various contexts. In cases where the state space and action space are straightforward and low-dimensional, the Q-learning [27] approach can be used to store the behavior value function Q in a table. This enables the implementation of the reinforcement learning experiment. However, when dealing with complex and high-dimensional state and action spaces, this strategy can lead to a dimension catastrophe. To address this issue, Mnih et al. [28] proposed the DQN technique, which combines deep learning and reinforcement learning. In this approach, the Q table is approximated using a deep neural network [29], denoted as $Q(s, a; \theta)$, where θ represents the parameters of the neural network. The state vector is input to the neural network, which outputs the value function for each action, thus resolving the aforementioned challenges.

However, the direct combination of deep learning and reinforcement learning in the DQN method introduces new issues:

- The basis for training a neural network is the assumption that the training data are independent and uniformly distributed, and that there is a high degree of correlation between the sequential data collected through agent interaction, which makes network training susceptible to instability.
- DQN network parameter θ is constantly updating, creating $Q(s, a)$ and $maxQ(s', a')$ with the same network results amid the constant changing of the neural network's time sequence difference target, which is detrimental to the algorithm's ability to converge.
- Early in the training process, the model is not stable enough and the value function estimation is inaccurate. By using $maxQ(s', a')$, the model overestimates the expected return of an action, leading the agent to choose the incorrect action and discover the best course of action.

The D3QN algorithm, an enhanced version of the DQN algorithm, is the reinforcement learning technique we selected. The method fixes the following issues in light of the aforementioned issues:

- The experience replay mechanism is used to sequentially store each experience gained from interacting with the environment in the experience pool. The model randomly selects a certain batch of data from the experience pool when it has accumulated to a specified number in order to train the neural network. Random extraction experience breaks data correlation and enhances generalization performance. Additionally, it enhances the stability of network training.
- The estimation network $Q_E(s, a; \theta)$ and target value network $Q_T(s, a; \theta')$ are built as two neural networks with the same structure. To choose actions and parameters, the estimating network is used. They are updated frequently; the target network is used to determine the time interval between the target value y and the target value. Every second or so, the parameter θ' is fixed and changed with the most recent valuation network parameters. This is how the target value y is determined:

  $$y = r + \gamma Q_T(s', argmax_{a'} Q_E(s', a'; \theta);); \theta') \tag{3.1}$$

  The evaluation network's relatively stable Q_E convergence objective y is made when θ' remains constant throughout time, which promotes convergence. The target value network's and the estimation network's maximum value functions' activities are not always identical. It is possible to prevent the model from choosing the overestimated suboptimal action and successfully address the overestimation issue of the DQN algorithm by using Q_E to generate the action and Q_T to determine the target value.
- The neural network's structure has been improved, and its output has been split into two portions. The state value function $V(s; \theta, \mu)$, which represents the quality of each state, is one component. The dominance function $A(s, a; \theta, \omega)$, which separates right from wrong behavior in a certain state, is the other component.

  $$Q(s, a; \theta, \mu, \omega) = V(s; \theta, \mu) + A(s, a; \theta, \omega) \tag{3.2}$$

  Finally, the target value of the D3QN model is as follows:

  $$y = r + \gamma Q_T(s', argmax_{a'} Q_E(s', a'; \theta);); \theta', \mu', \omega') \tag{3.3}$$

The following network parameters should be updated with the mean square error as the loss function:

$$L(\theta, \mu, \omega) = E[(Y - Q_E(s, a; \theta, \mu, \omega))^2] \tag{3.4}$$

This paper's parameter optimization problem can be reformulated as the reinforcement learning problem of pursuing precise objective. The Spark Streaming program interacts with the environment where the parameter configuration is chosen by calculating performance in order to select the parameter configuration with the best performance while it is running. We utilized the D3QN technique because the parameter space is discrete and enormous.

Designing the action space (A), state space (S), state transfer function (P) and reward function (R) is crucial for building a reinforcement learning model. In this paper, we focus on the Spark Streaming parameter optimization problem and implement the following approach. The operation of parameter values is referred to as the action space. The action space can be designed as $A = [A_0, A_1, A_2]$, corresponding to "raise 1", "unchanged" and "drop 1," respectively, depending on the value and change of the Spark Streaming parameter. The parameter configuration combination of the parameters that must be optimized is referred to as the state space. There is a range of values for each parameter. The state space of the model is formed by selecting a value from each parameter's value range to create a combination of parameter configurations. The state space is denoted by the expression $S = \{S_0, S_1, ...S_k, ..., S_n\}$, where S_k denotes a set of parameter configurations. The changing of various parameter configurations is referred to as a state transition. But because there are so many parameters and their range of values is so wide, there are a great number of conceivable parameter combinations, which slows down the agent's exploration process. This work suggests using the Gaussian process regression in conjunction with the parameter space transition to speed up convergence. In the current state S, the reward function refers to the immediate benefit that can be gained after executing action A. The reward function is built in the manner described below:

$$r = \frac{new\_per - old\_per}{old\_per} \tag{3.5}$$

In Eq (3.5), new_per represents the system's performance in the current state S after applying action A, while old_per represents the system's performance in state S before applying action. The reward is denoted by r to reflect the proportional performance changing. Performance improvement is indicated by a positive reward, whereas performance deterioration is represented as a negative reward. Due to it adaptively varying, the reward function has a more powerful incentive effect on modeling optimization.

3.2. Gaussian process

The Gaussian process [8], commonly referred to as the normal stochastic process, is a "Bayesian" [30] regression procedure that uses incremental learning. It alludes to the random process that may be observed at any given time t and whose random variables adhere to the Gaussian distribution.

The Gaussian process is uniquely determined by the mean function and the covariance function. In the field of machine learning, the machine learning method developed by combining the Gaussian random process and Bayesian learning theory is called a Gaussian process, which satisfies such a random process: the distributions of the set of benevolent finite variables are Gaussian distributions, that is, for any integer $n \geq 1$ and any group of random variables x, the joint probability distribution of the process state $f(x)$ at its corresponding time t obeys the n-dimensional Gaussian distribution. All of the statistical characteristics of a Gaussian process are completely determined by its mean function $m(t)$ and covariance function $k(t,t')$. Its definition is as follows:

$$f(t) \sim GP[m(t), k(t,t')] \tag{3.6}$$

When dealing with difficult issues like high dimensionality and nonlinearity, the Gaussian process works well. The Gaussian process model also has fewer parameters and is simpler to converge than approaches like neural networks, support vector machines [31] and others.

3.3. Gaussian process regression

A non-parametric model called Gaussian process regression uses the Gaussian process before performing a regression analysis on the data. It is a technique used in supervised learning in the field of machine learning to approximate functions. Using the sample data, this method aims to capture the distribution of the full value function. A Gaussian process can be created to represent a function distribution from the viewpoint of function space, and Bayesian inference can be executed directly in function space.

A brief explanation of how to apply the Gaussian process regression model is provided in this section. Consider a training sample dataset with the formula $D = \{(x_i, y_i), i = 1, 2, ..., n\}$, where x_i is the d-dimensional input vector and y_i is the 1-dimensional output. We must now estimate the new x_i's associated output y_i in accordance with the sample dataset D. We take into account the model below:

$$y_i = f(x_i) + \varepsilon \tag{3.7}$$

where x_i is the input variable, f is the real output value of the function, and ε is noise. Generally, we assume that the noise follows the distribution

$$\varepsilon \sim N(0, \sigma_n^2) \tag{3.8}$$

y is the observation target value affected by noise and its prior distribution is

$$y \sim N(0, K + \sigma_n^2 I) \tag{3.9}$$

where k is an $n \times n$ order symmetric positive definite covariance matrix, and the term K_{ij} in the matrix measures the correlation between x_i and y_i.

The purpose of the Gaussian process regression model is to determine f. We may express the noise factor more simply by converting it to the covariance function. Since the independent noise error accounts for the difference between the real curve and the observed value, we may define the noise error using the covariance function, as illustrated below:

$$k(x,x') = \sigma_f^2 exp[\frac{-(x-x')^2}{2l^2}] + \sigma_n^2 \delta(x,x') \tag{3.10}$$

Among them, l, σ_f, σ_n are the hyperparameters of the function, which have a great influence on the result. The optimal hyperparameter can be obtained using the maximum likelihood method. $\delta(x,x')$ is the Kronecker function. Only when $x = x'$ is the function value equal to 1, otherwise the function value is 0.

In the research of this paper, we can utilize the Gaussian process regression approach to analyze the performance obtained by the interaction between the execution of the Spark streaming program and the environment in reinforcement learning, so as to accelerate the convergence speed of reinforcement learning.

3.4. D3QN-GPR model design

The D3QN-GPR model combines the D3QN and the Gaussian process regression methods for reinforcement learning. The Gaussian process has the following benefits for reinforcement learning:

- In contrast to reinforcement learning, which explores the state space directly, state space exploration in the learning process can be guided by the probability distribution of the target value.
- The covariance matrix can be constructed by using an incremental approach, which is better for cutting down on calculations and making full use of the results of those calculations.

We explain this model as follows. Assume that a set of point data (corresponding to the state action pair in reinforcement learning) can be obtained as $\{(x_i, t_i)_{i=1}^{N}\}$, where x_i is the description value of the sample but t_i is the target value. Then ,according to the Bayesian method, we can establish a distribution model of t_{N+1} for a given sample description value x_{N+1}, that is,

$$P(t_{N+1}|(x_i, t_i), ..., (x_N, t_N), x_{N+1}) \tag{3.11}$$

In the Gaussian process, the observed target value is set as $t_N = [t_1, ..., t_N]^T$, which usually assumes a joint Gaussian distribution, and these target values are obtained from a real value with Gaussian noise; then,

$$P(t_N|x_i, ..., x_N, C_N) = \frac{1}{z} exp(-\frac{1}{2}(t_N - \mu)^T C_N^{-1}(t_N - \mu)) \tag{3.12}$$

where μ is the average value of the target value, C_n is a covariance matrix and Z is a normalized constant. Through the above analysis, it can be concluded that the D3QN-GPR model is feasible and effective for optimizing Spark streaming parameters.

Table 1 shows the description of symbols in the D3QN-GPR model. Characters used in this article are represented by symbols, while the interpretation of the previous symbol is represented by descriptions.

Table 1. Notations.

Notation	Definition
θ	Current Q network parameters
θ'	Parameters of target Q' Network
A	Current action
A'	Next action
S	Current state
S_j	State of the jth time
S'_j	Next state of the jth time
T	Total iteration number
γ	Attenuation factor
ε	Exploration rate
P	Target Q network parameter update frequency
m	Number of samples per random sampling
D	Replay buffer
R	Reward
R_j	Reward obtained for the jth time
$loss$	Mean square loss function
Con	Optimal parameter configuration
DDT	The performance of the Spark Streaming program
DDT'	The performance of executing Gaussian process regression
Par	Parameter combination
y_i	Target value of D3QN model
$batch\_size$	Neural network hyperparametric
$reset$	Reset function in reinforcement learning
$action$	Selective action function in reinforcement learning
$step$	Executive action function in reinforcement learning
$push$	The function of storing in replay buffer
$loss$	The function of calculate loss
GPR	Gaussian process regression

The implementation process of the model is as follows. In the D3QN-GPR model, we first input the initialized parameters, then use the agent in reinforcement learning to continuously interact with the environment, optimize the parameter configuration of Spark Streaming according to the results of each parameter configuration and finally output the optimized parameter configuration.

Table 2. D3QN-GPR model algorithm.

```
Algorithm 1: D3QN-GPR model algorithm
Input:
    Initialize parameters $\theta, \theta \to \theta', Q', T, \gamma, \varepsilon, P, m, D$
Output:
    $Con$
 1: for t = 1 to T do
 2:     //reset the environment and get the status $S, done$, initialize $R$
 3:     $S, done = env.reset()$
 4:     $R = 0$
 5:     while True do
 6:         //Execution times
 7:         $step\_count+ = 1$
 8:         //Select the corresponding action A under the current state S using the $\varepsilon - greedy$ method
 9:         $A = D3QN.action(S)$
10:         //Execute $A$ to obtain $s'$ and $R$, $done$,$Con$, $Par$.
11:         $S', R, done, Con, Par = env.step(A)$
12:         //Store $\{S, S', A, R, done\}$ in replay buffer D
13:         $D.push(S, S', A, R, done)$
14:         if done then
15:             $break$
16:         else
17:             //Take $m$ samples from D: $\{S_j, S'_j, A_j, R_j, done_j\}, j = 1, 2, 3, ..., m$; $D$, $y_i$ of the current Q
                network is calculated according to Eq (3.3):
18:             $y_i = r + \gamma Q_T(S', argmax_{A'} Q_E(S', A'; \theta); ); \theta', \mu', \omega')$
19:             //Use the mean square loss function to calculate loss
20:             $loss = D3QN.learn(D, batch_size, step_count, \gamma)$
21:         end if
22:     end while
23:     //After a certain number of execution, the Q value is corrected by Gaussian process regression,
        and the subsequent execution uses the new Q value
24:     $DDT' = GPR(Par, DDT, S)$
25:     //Update $\theta', S, DDT, t$
26:     $S \leftarrow S'$
27:     $DDT \leftarrow DDT'$
28:     $t \leftarrow t + 1$
29: end for
30: return $Con$
```

Figure 3 shows the execution flow chart of the D3QN-GPR model. Based on the reinforcement learning method D3QN, the implementation flow chart adds a Gaussian process regression to modify the Q value so as to accelerate the model convergence.

In our D3QN-GPR model, we first initialize the model parameters and associated variables. Before starting an iteration, the environment state is reset and the current state is given a cumulative reward of 0. Then, based on the current state, we employ the ε-greedy method to choose actions within each iteration. After the chosen action is carried out, the next state, reward, program end flag, and other pertinent information are updated and stored in the experience replay buffer. A random batch of samples is selected from the experience replay buffer for the agent to update in each iteration unless the program terminates. The mean square error loss function is introduced to calculate the intended output value for each sample and update the DQN. Then, the Q value is adjusted using Gaussian process regression after a predetermined number of executions, and the updated Q value is applied to subsequent executions.

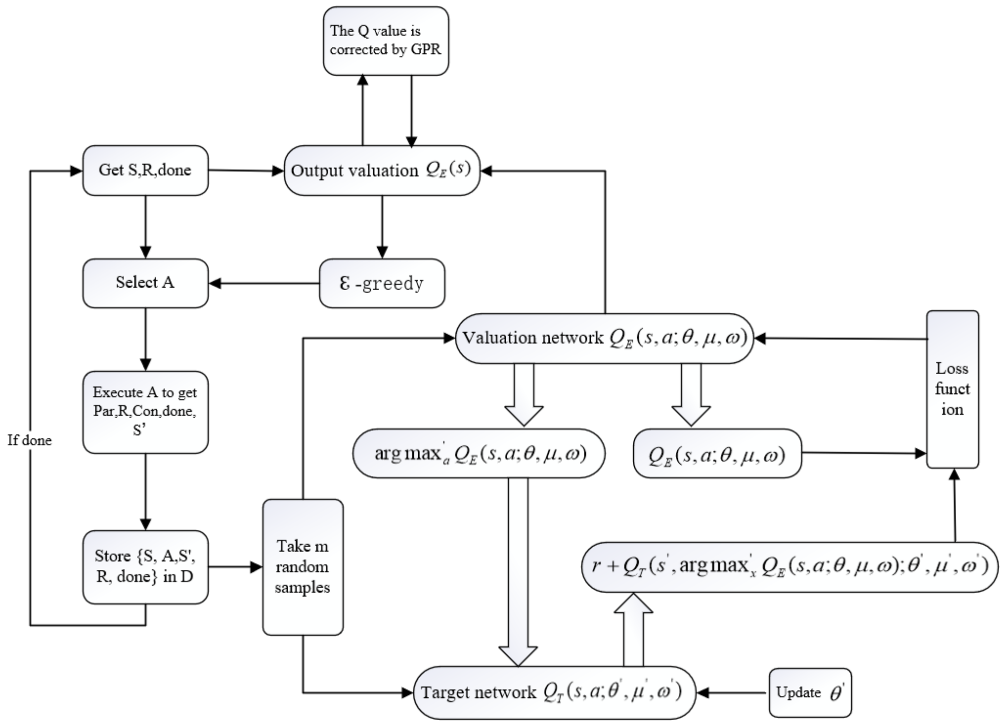
The Q value is corrected by GPR
Get S,R,done
Output valuation $Q_E(s)$
Select A
ε -greedy
Valuation network $Q_E(s,a;\theta,\mu,\omega)$
Loss function
Execute A to get Par,R,Con,done, S'
If done
$\arg\max_a' Q_E(s,a;\theta,\mu,\omega)$
$Q_E(s,a;\theta,\mu,\omega)$
Store {S, A,S', R, done} in D
Take m random samples
$r+Q_T(s',\arg\max_x' Q_E(s,a;\theta,\mu,\omega);\theta',\mu',\omega')$
Target network $Q_T(s,a;\theta',\mu',\omega')$
Update θ'

Figure 3. D3QN-GPR execution flow chart.

4. Experimental evaluation

In this study, in order to assess the performance of the D3QN-GPR model, we employed HiBench [32], a big data benchmark platform, and a distributed cluster with six nodes (one master node and five slave nodes). The configuration of the distributed cluster environment is as follows: CentOS 8.2, Spark-2.4.0-bin-hadoop 2.7, JDK 1.8.0_144, Hadoop 2.7.2, Kafka-2.11-2.4.0, Scala-2.11.8, Zookeeper-3.4.8. The cluster was equipped with 64 GB of memory and an i9-10900k CPU.

HiBench is a big data benchmarking toolkit that may be used to assess how efficiently the big data framework uses system resources in terms of speed, throughpu and utilization. The Spark framework is just one of the numerous frameworks that it supports in the big data ecosystem. The benchmark test for Spark Streaming apps is supported by the Spark framework. As a result, we can use the Spark

Streaming HiBench benchmark tool. A general breakdown of the six test categories in HiBench's 19 test directions is as follows: microtest, machine learning, SQL, graphics, network search and streaming media. Fixwindow, Repartition and WordCount are the three benchmarks that we have chosen from the HiBench benchmark suite's streaming media test category. The Spark Streaming framework's window operation performance was tested using Fixwindow. The effectiveness of data shuffle in the streaming framework was tested using partitioning. To evaluate the effectiveness of stateful operators in streaming frameworks, the CPU-intensive WordCount application was utilized.

4.1. Evaluating indicator

Because the streaming processing system calls for minimal delay and high throughput, we applied the average $delay/throughput$ as the assessment criteria during the experiment. As shown in Figure 4, the delay of a streaming processing system is the amount of time Spark Streaming must wait before receiving the new batch of data after processing the old batch.

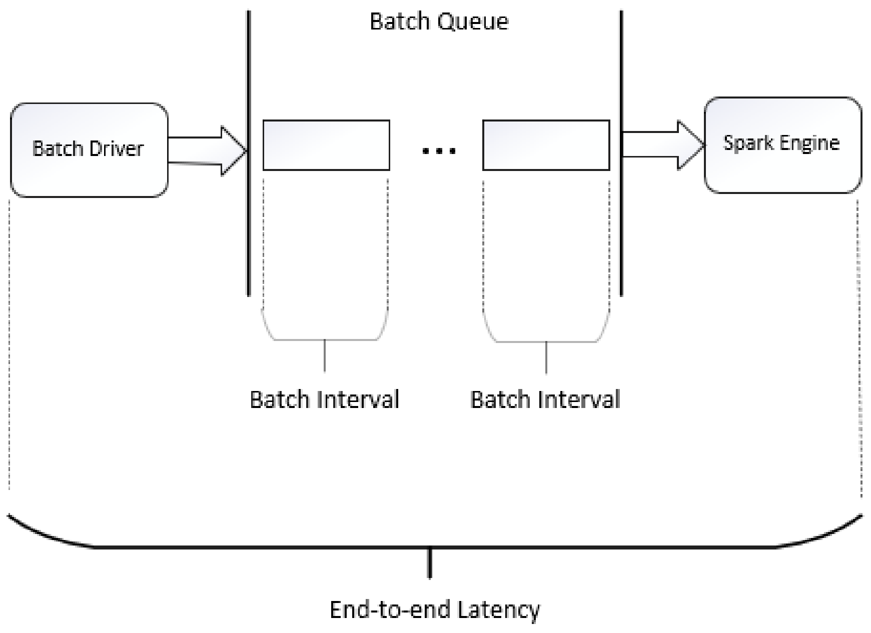

Figure 4. Delay of streaming processing system.

The system's responsiveness is impacted by the delay. Spark Streaming has strict constraints on delay because it is a quasi-real-time streaming computing engine. High delay results in a backlog in data processing and has a significant impact on the system's performance. Low latency is therefore crucial for distributed flow systems. High throughput refers to the ability of the streaming processing system to maintain a fast rate of data generation and reception. Processing streaming data requires high throughput processing, which is crucial. For instance, LinkedIn [33] recently stated in a blog post that Kafka creates more than 1 trillion pieces of data per day. The time-series database at Twitter [34] reported that it must process 2.8 billion posts every minute. In order to keep up with these rates, it is necessary to use a distributed stream processing system whose throughput meets or exceeds the incoming data rate. We selected the average delay and throughput as the performance metrics for

Spark Streaming based on the analysis above. As a result, once the D3QN-GPR model has chosen the parameters, the following formula is used to assess how well the Spark Streaming execution program performed:

$$DDT = delay/throughput \tag{4.1}$$

where *DDT* represents the Spark Streaming program's performance. By evaluating from the perspective of *DDT* instead of a single standard of average latency or throughput, low latency and low throughput, high latency and high throughput, and the complexity of the two elements are all prevented.

4.2. Parameter Selection

In Spark Streaming, more than 200 parameters can be specified. The factors can be classified into 13 categories based on how they are used. However, not all of them, like the operating environment and encryption, have an effect on performance. As a result, choosing optimal parameters is a step that must be taken in the parameter optimization process. According to the previous studies of Du et al. [5], Ye et al. [35] and the Spark Streaming Programming Guide [36], we selected the following parameters in Table 3 to evaluate the performance of our framework:

Table 3. Selection of optimization parameters.

Notation	Definition
executor.cores	Set the CPU cores of per executor
executor.memory	Set the memory size of per executor
driver.memory	Set the Memory size of the driver process
executor.number	Set the number of executors during program execution
shuffle.parallelism	Set the number of parallel tasks in shuffle
map.parallelism	Set Set the parallelism of the operator "map"
streaming.batchInterval	Set the size interval of a batch of data streams read

To confirm the effects of the parameters in Table 3 on the effectiveness of Spark Streaming, we employed the single variable method. To test each parameter, we did our experiments on various Hibench benchmarks. We can determine the effect of each parameter on the performance of various benchmark programs by changing the value of each parameter within the range of optional values and then using the average *DDT* after five executions as the performance corresponding to the current value.

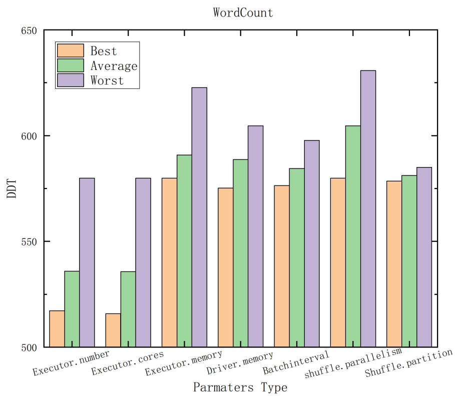

Figure 5. Comparison of WordCount program.

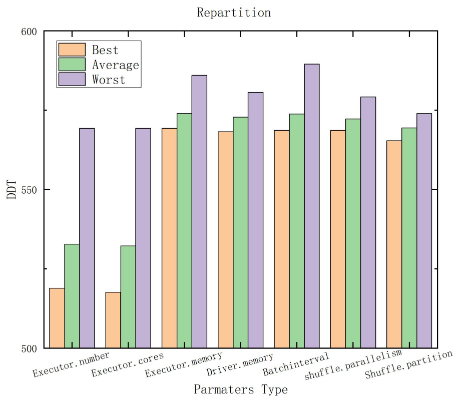

Figure 6. Comparison of Repartition program.

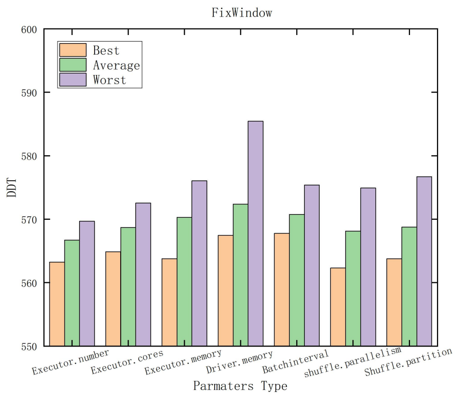

Figure 7. Comparison of FixWindow program.

Figures 5–7 display the best case, worst case and average effects of each parameter on the performance of the Spark Streaming streaming system for the following HiBench benchmark programs: WordCount, FixWindow and Repartition. Each figure shows that these seven parameters do affect system performance differently, and that the same parameters affect the performance of various applications differently. Because of this, the tuning parameters we chose are reasonable, and it is possible to increase Spark Streaming performance by changing the parameter configuration.

4.3. Experimental results and analysis

In order to determine the optimized parameter configuration for each program, we first utilized the D3QN-GPR model to run the three Hibench benchmark test programs. After that, in order to determine the program's ideal parameter configuration, we tested by utilizing No-Stop in the same environment. The default parameter setting was then put up against the two other parameter configurations. For the HiBench benchmark test, we decided to experiment with WordCount, Repartition and FixWindow. We employed five datasets with differing flow rates in various benchmark tests. We picked the five average *DDT* values for each data flow rate in order to compare the performance of these three parameter configurations under various benchmark test procedures.

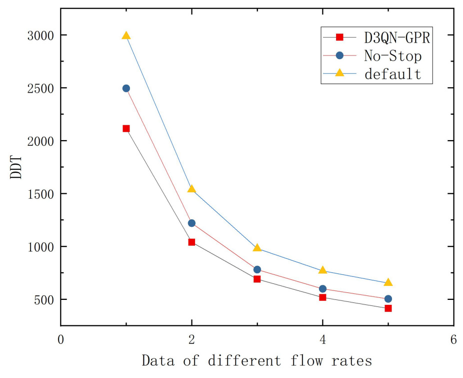

Figure 8. Comparison of WordCount program.

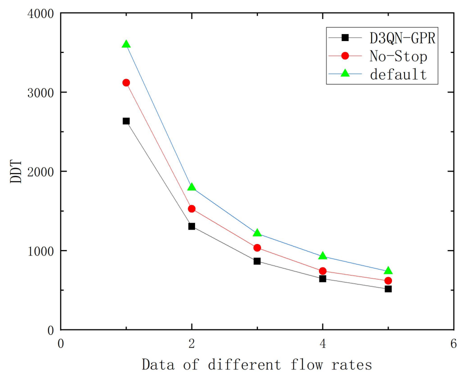

Figure 9. Comparison of Repartition program.

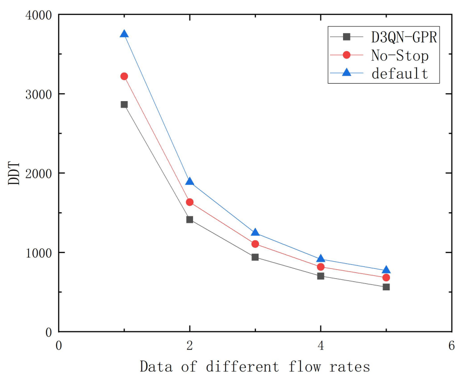

Figure 10. Comparison of FixWindow program.

According to Figures 8–10, the execution performance of D3QN-GPR improved with varying degrees for the benchmark programs WordCount, Repartition, and FixWindow, with respective percentages of 30.24, 12.38 and 19.35%. It also improved somewhat in comparison to the No-Stop mode, with percentages of 8.85, 6.68 and 6.14%.

Figures 11–13 show how the performance of the parameter configuration utilizing the D3QN-GPR model significantly improved over that of the default parameter when the identical program was run. It also provides certain speed advantages over using No-Stop optimization to get the settings needed to run various programs. As a result, Spark Streaming can definitely perform better under various loads when the D3QN-GPR model's parameter configuration is optimized.

The effectiveness of the D3QN-GPR model was then assessed by contrasting the two models' rates of convergence during the parameter optimization process under three different loads. This was done in order to compare the reinforcement learning method without Gaussian process regression with the D3QN-GPR algorithm.

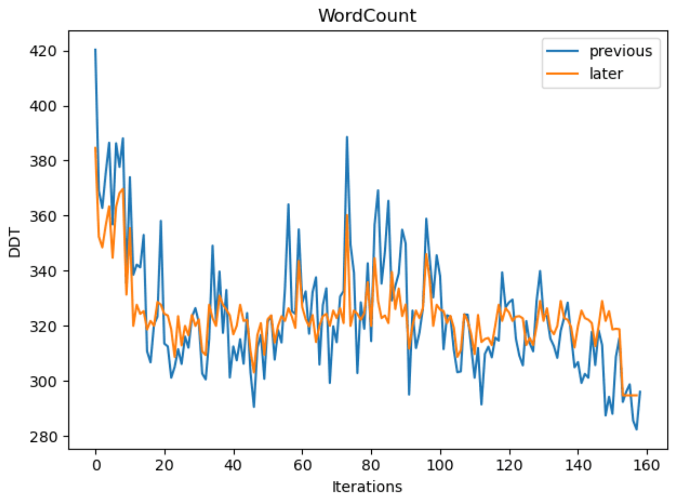

Figure 11. Comparison of WordCount program.

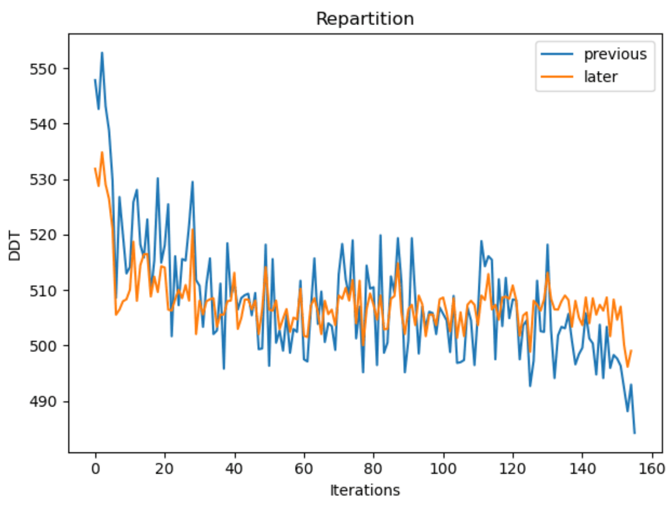

Figure 12. Comparison of Repartition program.

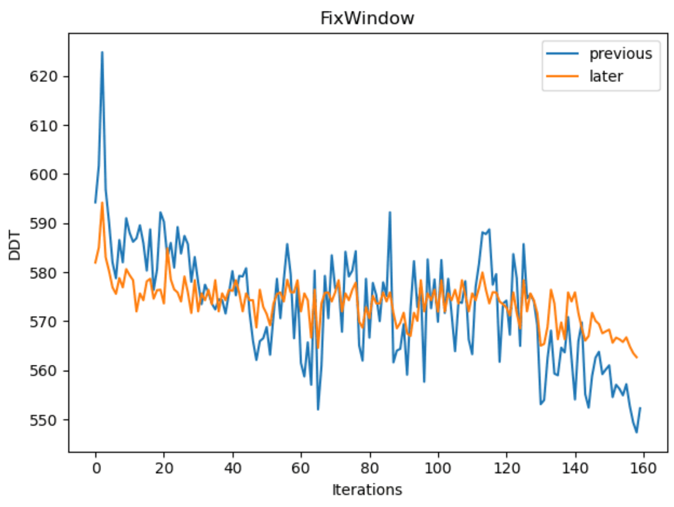

Figure 13. Comparison of FixWindow program.

According to the findings in Figures 11–13, as compared to D3QN without Gaussian process regression, our proposed method can increase the exploration efficiency of reinforcement learning in interactions with the environment, accelerate the convergence of the D3QN method and shorten the time for parameter configuration optimization.

5. Conclusions and future work

This study evaluates the effects of various factors on Spark streaming programs and discusses how they may impact the performance of the technology.

In the study, we propose a D3QN-GPR model for Spark streaming performance optimization, which speeds up the convergence of training. The D3QN with Gaussian process regression works well on small datasets and also has the ability to offer uncertainty measurements for the prediction process. In order to accelerate the computation and achieve quick convergence of reinforcement learning, we have also introduced the batch gradient descent algorithm and vectorization calculation of Gaussian process regression. In the experimental section, we compared our novel D3QN-GPR parameter optimizer with other state-of-the-art technologies. The experimental results demonstrate that the parameter optimization of the D3QN-GPR model yields significant performance improvements as compared with the default parameter configuration. Specifically, the WordCount, Repartition and FixWindow benchmarks achieved performance gains of 30.24, 12.38 and 19.35%, respectively. Compared to the parameter optimization of the No-Stop model, these benchmarks showed performance improvements of 8.85, 6.68 and 6.14%, respectively. Furthermore, the D3QN-GPR model exhibited a significant reduction in convergence time as compared with the reinforcement learning algorithm without Gaussian process regression. These experimental findings provide strong evidence for the superior performance optimization and convergence time reduction achieved by the D3QN-GPR model.

In our future work, we will consider a wide variety of factors to improve the applicability and scalability of the setup parameters of the D3QN-GPR model. The effects of just seven parameter combinations on Spark streaming performance were investigated in this work. We plan to introduce more parameters and develop more cunning parameter selection strategies in our upcoming work, referring to other works and domain knowledge.

Use of AI tools declaration

The authors declare that they have not used artificial intelligence tools in the creation of this article.

Acknowledgments

This work was supported by the Science and Technology Research Project of Hebei Higher Education Institutions (No. QN2020133) and the Natural Science Foundation of Hebei Province of China (No. F2019201361).

Conflict of interest

The authors declare that there is no conflict of interest.

References

1. Apache storm. Available from: https://storm.apache.org/.
2. Apache spark streaming. Available from: https://spark.apache.org/docs/latest/streaming-programming-guide.html.
3. Apache flink. Available from: https://flink.apache.org/.
4. D. Cheng, X. Zhou, Y. Wang, C. Jiang, Adaptive scheduling parallel jobs with dynamic batching in spark streaming, *IEEE Trans. Parallel Distrib. Syst.*, **29** (2018), 2672–2685. https://doi.org/10.1109/TPDS.2018.2846234

5. H. Du, P. Han, Q. Xiang, S. Huang, Monkeyking: Adaptive parameter tuning on big data platforms with deep reinforcement learning, *Big Data*, **8** (2020), 270–290.
6. A. Kordelas, T. Spyrou, S. Voulgaris, V. Megalooikonomou, N. Deligiannis, KORD-I: A framework for real-time performance and cost optimization of apache Spark Streaming, in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, (2023), 1–3.
7. L. Liu, G. Shen, C. Guo, Y. Cui, C. Jiang, D. Wu, A spark streaming parameter optimization method based on deep reinforcement learning, *Comput. Modernization*, **2021** (2021), 49–56.
8. J. Wang, An intuitive tutorial to Gaussian processes regression, preprint, arXiv:200910862. https://doi.org/10.48550/arXiv.2009.10862
9. Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, N. Freitas, Dueling network architectures for deep reinforcement learning, in *Proceedings of Machine Learning Research*, (2016), 1995–2003.
10. H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in *Proceedings of the AAAI conference on artificial intelligence*, **30** (2016). https://doi.org/10.1609/aaai.v30i1.10295
11. E. Schulz, M. Speekenbrink, A. Krause, A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions, *J. Math. Psychol.*, **85** (2018), 1–16. https://doi.org/10.1016/j.jmp.2018.03.001
12. X. Wang, S. Wang, X. Liang, D. Zhao, J. Huang, X. Xu, et al., Deep reinforcement learning: A survey, *IEEE Trans. Neural Networks Learn. Syst.*, **2022** (2022). https://doi.org/10.1109/TNNLS.2022.3207346
13. L. P. Swiler, M. Gulian, A. L. Frankel, C. Safta, J. D. Jakeman, A survey of constrained gaussian process regression: Approaches and implementation challenges, *J. Machine Learn. Model. Comput.*, **1** (2020). https://doi.org/10.1615/JMachLearnModelComput.2020035155
14. R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, MIT press, 2018.
15. Z. H. Zhou, *Machine Learning*, Springer Nature, 2021.
16. J. C. Lin, M. C. Lee, I. C. Yu, E. B. Johnsen, Modeling and simulation of spark streaming, in *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, (2018), 407–413. https://doi.org/10.1109/AINA.2018.00068
17. S. Venkataraman, A. Panda, K. Ousterhout, M. Armbrust, A. Ghodsi, M. J. Franklin, et al., Drizzle: Fast and adaptable stream processing at scale, in *Proceedings of the 26th Symposium on Operating Systems Principles*, (2017), 374–389. https://doi.org/10.1145/3132747.3132750
18. T. Ajila, S. Majumdar, Data driven priority scheduling on spark based stream processing, in *2018 IEEE/ACM 5th International Conference on Big Data Computin Applications and Technologies (BDCAT)*, (2018), 208–210. https://doi.org/10.1109/BDCAT.2018.00034
19. M. Petrov, N. Butakov, D. Nasonov, M. Melnik, Adaptive performance model for dynamic scaling Apache Spark Streaming, *Proc. Comput. Sci.*, **136** (2018), 109–117.
20. W. Li, D. Niu, Y. Liu, S. Liu, B. Li, Wide-area spark streaming: Automated routing and batch sizing, *IEEE Trans. Parallel Distrib. Syst.*, **30** (2018), 1434–1448.

21. H. Zhao, L. B. Yao, Z. X. Zeng, D. H. Li, J. L. Xie, W. L. Zhu, et al., An edge streaming data processing framework for autonomous driving, *Connect. Sci.*, **33** (2021), 173–200. https://doi.org/10.1080/09540091.2020.1782840

22. B. Liu, X. Tan, W. Cao, Dynamic resource allocation strategy in spark streaming, *J. Comput. Appl.*, **37** (2017), 1574. https://doi.org/10.11772/j.issn.1001-9081.2017.06.1574

23. D. Cheng, Y. Chen, X. Zhou, D. Gmach, D. Milojicic, Adaptive scheduling of parallel jobs in spark streaming, in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, (2017), 1–9. https://doi.org/10.1109/INFOCOM.2017.8057206

24. B. R. Prasad, S. Agarwal, Performance analysis and optimization of spark streaming applications through effective control parameters tuning, in *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications*, Springer, (2018), 99–110. https://doi.org/10.1007/978-981-10-3376-6_11

25. G. Liu, X. Zhu, J. Wang, D. Guo, W. Bao, H. Guo, SP-Partitioner: A novel partition method to handle intermediate data skew in spark streaming, *Future Gener. Comput. Syst.*, **86** (2018), 1054–1063. https://doi.org/10.1016/j.future.2017.07.014

26. Z. Fu, Z. Tang, L. Yang, K. Li, K. Li, Imrp: A predictive partition method for data skew alleviation in spark streaming environment, *Parallel Comput.*, **100** (2020), 102699. https://doi.org/10.1016/j.parco.2020.102699

27. J. Clifton, E. Laber, Q-learning: Theory and applications, *Ann. Rev. Stat. Appl.*, **7** (2020), 279–301. https://doi.org/10.1146/annurev-statistics-031219-041220

28. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, et al., Human-level control through deep reinforcement learning, *Nature*, **518** (2015), 529–533. https://doi.org/10.1038/nature14236

29. W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, K. R. Müller, Explaining deep neural networks and beyond: A review of methods and applications, *Proc. IEEE*, **109** (2021),247–278. https://doi.org/10.1109/JPROC.2021.3060483

30. M. G. Titelbaum, *Fundamentals of Bayesian Epistemology 2: Arguments, Challenges, Alternatives*, Oxford University Press, 2022.

31. B. Gaye, D. Zhang, A. Wulamu, Improvement of support vector machine algorithm in big data background, *Math. Prob. Eng.*, **2021** (2021), 1–9. https://doi.org/10.1155/2021/5594899

32. N. Ihde, P. Marten, A. Eleliemy, G. Poerwawinata, P. Silva, I. Tolovski, et al., A survey of big data, high performance computing, and machine learning benchmarks, in *Performance Evaluation and Benchmarking: 13th TPC Technology Conference*, (2021), 98–118. https://doi.org/10.1007/978-3-030-94437-7_7

33. Datanami, Kafka Tops 1 Trillion Messages Per Day at LinkedIn. Available from: https://goo.gl/cY7VOz.

34. Observability at twitter: Technical overview. Available from: https://goo.gl/wAHi2I.

35. Q. Ye, W. Liu, C. Q. Wu, Nostop: A novel configuration optimization scheme for Spark Streaming, in *50th International Conference on Parallel Processing*, (2021), 1–10.

36. Spark tuning guide. Available from: https://spark.apache.org/docs/latest/streaming-programming-guide.html#deploying-applications.