OPOM: The Ordinal and Parallel Optimization Method of Spark multi-query applications

Bingyu Guan(✉)

School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
guanbingyu@hust.edu.cn

Abstract. Optimizing the configuration parameters of Spark applications presents a challenging problem [10,15] in cloud computing, with the optimization of Spark multi-query applications being the most challenging aspect. Multi-query applications consist of multiple queries, such as TPC-DS [27] and TPC-H [4]. As the amount of data increases, the optimization cost of multi-query applications will increase substantially, making these applications increasingly challenging to optimize.
In this study, we propose an Ordinal and Parallel Optimization Method (OPOM). Specifically, we execute queries in parallel and design the Partition Execution Order of queries (ordinal), which can significantly reduce the optimization time cost of multi-query applications. In the benchmarks (TPC-DS and TPC-H) of three data sizes and the Huawei Cloud, OPOM achieved the highest Cost Reduction of optimization time compared to Cherrypick [2], ROBOTune [18], and LOCAT [37], with maximum reductions of 290.84%, 277.26%, and 231.99%, and average reductions of 239.25%, 228.54%, and 194.89%, respectively. The maximum speedups for OPOM, Cherrypick, ROBOTune, and LOCAT are 12.13×, 3.39×, 3.68×, 3.73×, and the corresponding average speedups are 8.97×, 2.89×, 3.00×, 3.05×, respectively.

Keywords: Bayesian Optimization · multi-query applications · low time-cost · execution order of queries.

1 Introduction

Spark [39] is vital in cloud computing as a powerful distributed data processing framework. It boasts outstanding performance and scalability, handling petabytes of data and running efficiently on distributed clusters. Spark offers a rich set of functionalities and APIs, including batch processing [40], stream processing [1], machine learning [21], and graph processing [8], enabling users to accomplish various data processing tasks within the same framework. Spark's emergence has significantly changed how big data is processed, providing users with more efficient and reliable data processing solutions.

Due to Spark's importance and wide application in big data processing, there is a significant demand for its performance. To enhance performance when handling different applications, Spark defines hundreds of configuration parameters

to tune the performance of applications. Due to the varying degrees of influence each configuration parameter has on the performance of the application and the interdependencies between configurations, identifying the optimal configuration parameters for an application becomes an NP-hard problem [16]. Selecting the optimal configuration from a vast number of parameter combinations is a challenging yet crucial task , as the performance (such as runtime) achieved with good configurations often surpasses that obtained with poor configurations by several orders of magnitude [34].

To optimize multi-query applications, existing research methods [37,11] execute each query serially and consider all queries as a whole. They did this to control the variables and ensure that only one query is running simultaneously, eliminating any impact on the application beyond the spark configuration parameters. However, in an actual situation, especially in a cloud environment, there are often multiple applications running together, so it is very practical to consider the parallel running of applications; specifically, the execution of queries in multi-query applications is independent, and they do not need to provide intermediate data to each other,which provides the possibility of running queries in parallel. Inspired by this characteristic of query independence, we propose a parallel optimization approach for multi-query applications to fully utilize resources and further improve performance.Multi-query applications need to successfully execute all queries to complete one run. If machine learning methods are used to model and search for the optimal configuration, the time cost would be prohibitively high. Therefore, we adopt a Bayesian optimization-based method to search for an optimal configuration.

Optimizing queries in parallel presents more significant challenges, as the execution time of each query is influenced not only by configuration parameters but also by the execution order of queries. To address this challenge, we propose the Partitioning Execution Order method (Section 4.1), which divides queries into several query subsets and executes them in parallel with the ultimate goal of achieving nearly simultaneous completion of queries within each subset. The key to this method lies in integrating a greedy algorithm that dynamically adjusts the execution order based on observed query execution times. This adaptive strategy ensures efficient resource utilization and reduces the optimization time cost.

The execution order of queries can significantly impact the runtime of applications. Even with the same configuration parameters, different query execution orders can lead to variations in the application execution time. To address this issue, we propose a Knowledge Transfer method (Section 4.2) that assigns execution orders to each run of data and transforms the execution time in the observation data into a theoretically optimal execution time. To search for configuration parameters, we introduce the Variable Bayesian optimization method (Section 4.3), which effectively reduces the search space and optimizes the number of queries, thereby substantially reducing the optimization time cost.

Specifically, the main contributions of this paper can be summarized as follows:

- The Partitioning Execution Order method partitions queries into multiple query subsets, aiming to ensure that queries within each subset can finish running simultaneously as much as possible. This approach enhances resource utilization and reduces the cost of optimization time.
- The Knowledge Transformation method transforms the execution time in the running data into theoretically optimal execution time, thereby mitigating the impact of query execution order.
- As the amount of observation data increases, the Variable Bayesian Optimization method narrows the search space and simplifies the optimization objective, significantly Speeding up the optimization process.
- Extensive experimental study to verify that OPOM significantly outperformed existing approaches in both parallel and serial modes.

The rest of this paper is organized as follows. Section 2 presents the background and motivation of this paper;Section 3 and Section 4 offer OPOM method, consisting of Partition Execution Order, Knowledge Transfer and Variable Bayesian Optimization methods;Section 5 describes the experimental setup and analyzes the experimental results; Section 6 describes related work; Finally,Section 7 concludes this paper.

2 Background And Motivation

2.1 Multi-Query Application

Some Spark SQL applications have multiple queries called multi-query applications (such as TPC-DS and TPC-H Eq. 1), allowing flexible querying and processing to meet various data analysis needs or query tasks. Each query represents a specific data query, transformation, or analysis operation, typically targeting the same dataset, but with potentially different purposes or requirements. Multi-query applications can better support complex data processing and analysis needs, improving the efficiency and flexibility of data processing [29,23,28].

$$MQA = \{q_1, q_2, q_3, \ldots, q_N\} \tag{1}$$

where MQA is the Multi-Queries Application; N is the number of queries.

Queries are often [37,11] executed serially to find the optimal configuration for multi-query applications by controlling the variables and ensuring that only one query is running simultaneously, eliminating any impact on the application beyond the spark configuration parameters. However, in an actual situation, especially in a cloud environment, multiple applications often run together, and executing queries serially can be time-consuming for multi-query applications, making it impractical for optimization tasks that require fast completion. Therefore, the runtime must be reduced to solve this problem. Jin Han Xin et al. [37] found that some queries are sensitive to configuration parameters while others are not. This means that despite the changes in the configuration parameters,

the runtime of some queries remains relatively stable. This observation suggests that if queries are executed in parallel, the runtimes of some queries may remain the same. Even if the runtime of some queries increases, the runtime of multi-query applications remains within an acceptable range. Therefore, we conjecture that running queries in parallel can significantly reduce the running time, and the experiment proves that it is feasible. Therefore, we propose a new query execution method that executes queries in parallel and designs the Partition Execution Order of the queries in multi-query applications.

The query execution method divides N queries into p subsets and runs them simultaneously (represented by the columns in Eq. 2). Each subset executes queries in the order q_{i1} to q_{iN_i} until all queries are completed. See (Section 4.1) for further details.

$$\begin{array}{|cccc|}\hline q_{11} & q_{21} & \cdot\cdot & q_{p1} \\ q_{12} & q_{22} & \cdot\cdot & \vdots \\ \vdots & q_{23} & \cdot\cdot & q_{p(N_p-1)} \\ q_{1N_1} & \vdots & \cdot\cdot & q_{pN_p} \\ \hline\end{array} \tag{2}$$

where p is the number of parallelism; N_i is the number of queries in the ith subset.

GP offers advantages such as robustness to noisy observations and support for gradient-based methods [32]. Therefore, in this paper, we adopt GP as the surrogate model for Bayesian Optimization and leverage EI [22] as the acquisition function, which performs better than others across a wide range of test cases [32].

3 System Overview

As shown in Fig.1 is an overview of the OPOM method and the workflow is as follows:

Step 1: Initially, collect samples from Spark configuration parameters for multi-query applications (For example, Latin hypercube sampling [14] is used to collect spark configuration parameter samples and execute the samples to obtain the corresponding execution time of each query.);

Step 2: According to the runtime samples of each query collected in step 1, the average running time of each query is calculated. According to the current running environment and the situation of the multi-query application, a specific degree of parallelism is set. We (described in Eq. 2) divide N queries into P subsets and run them simultaneously (represented by the columns in Eq. 2), based on the average runtime of each query and the specific degree of parallelism. Each subset executes queries in the order of q_{i1} to q_{iN_i}(Eq. 2) until all queries are completed. See (Section 4.1) for details.

Steps 3-4: According to Partition Execution Order method (Section 4.1), perform the query execution order division for each sample. Based on the query Partition Execution Order, we obtain the theoretical minimum runtime achievable under the configuration parameters of that sample, and replace the actual

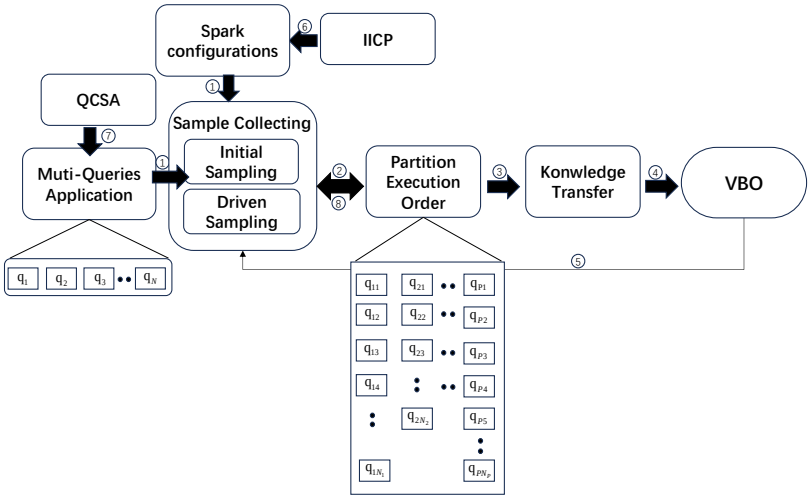

Fig. 1: The overview of OPOM. BO — Bayesian Optimization. During the search process, the query execution order is divided (step 2), the samples are transformed(step 3), and VBO selects the spark configuration parameters for the next search based on the transformed samples. When the number of samples is sufficient, QCSA and IICP are executed until the search-stop condition is reached.QCSA — Query Configuration Sensitivity Analysis [37]. IICP— Identifying Important Configuration Parameters [37].

runtime in the sample with this theoretical runtime to obtain new sample data (see Section 4.2 for details). The new sample data are passed to the surrogate model of Bayesian optimization as the training data.

Step 5: We next use Driven Sampling to collect samples. Specifically, Bayesian optimization selects configuration parameters through the acquisition function. Step 8: executes the multi-query application according to the query execution order divided in step 2. Continuously execute Steps 2-5 until a sufficient number of samples are collected, and then the important configuration parameters (IICP [37] Step 6 to narrow down the search space for BO, thereby enhancing the convergence speed. Continue executing Steps 2-5 until adequate samples are obtained. Then, query Configuration Sensitivity Analysis (QCSA [37] Step 7) will be conducted to identify configuration-sensitive queries. Adjust the optimization objective to focus on configuration-sensitive queries, significantly reducing the optimization time costs; Continue executing Steps 2-5 until the stopping criteria (such as achieving the optimization goal or algorithm convergence) are reached.

4 OPOM Method

The OPOM method mainly consists of three parts: knowledge transfer, partition execution order, and Variable Bayesian Optimization:

1. **Partition Execution Order (PEO)**: Based on the current average execution time of each query, queries are grouped to ensure that the total execution time of each group is as equal as possible.
2. **Knowledge Transfer (KT)**: The method transforms the execution time of multi-query applications into theoretical minimum runtime based on PEO, thereby mitigating the impact of query execution order.
3. **Variable Bayesian Optimization (VBO)**: Compared with traditional Bayesian optimization, the three essential components of the VBO in this paper are variable, which are the search space, optimization objective, and observation data.

Section 3 introduces the connection between the three components. Next, we will provide detailed explanations of the three components.

4.1 Partition Execution Order

When queries are executed in parallel, the number of running queries may not always match the parallelism, leading to the issue of varying query execution times and resource wastage. If we strive to keep the number of concurrently running queries constant, ensuring that queries finish their execution together as much as possible, resource wastage can be minimized.

We adopted a partitioning execution order method to reduce resource wastage. Specifically, the entire query set of a multi-query application is divided into multiple query subsets, with each subset containing queries that are executed together. We use the average execution time of queries as a criterion and aim to divide N queries into P subsets, such that the sum of the average execution times of queries within each subset is approximately equal, which increases the likelihood of achieving our goal. A simple method is to first calculate the mean runtime of the query subset (MRQS Eq. 3), and the second step involves calculating the deviation between the runtime of the query subset and MRQS for all partition combinations (Eq. 4).

$$\mathrm{MRQS} = \frac{\sum_{j=1}^{N} \mathrm{RQ}_j}{\mathrm{P}} \tag{3}$$

where MRQS is the Mean Runtime of Queries Subset;RQ_j is the Runtime of Query-j;P is the number of subset;

$$\text{Deviation } = \sum_{\mathrm{i}=1}^{P} |\mathrm{RQSS}_i - \mathrm{MRQS}| \tag{4}$$

where RQSS is the Runtime of Query SubSet; Deviation is the deviation between the RQSS and MRQS.

However, this simple method encounters a problem: when dividing N queries into P subsets, there are $\mathrm{C}_N^P * P^{(N-P)}$ possible partition combinations, which can increase exponentially with the number of queries. When optimizing applications such as TPC-DS with 103 queries and a parallelism degree set to four,

there would be a large number of partition combinations ($1.78 * 10^{66}$) using this method. This imposes a significant additional time overhead on the optimization of multi-query applications.

To solve this problem, we propose a greedy algorithm that reduces the time complexity to $N * P$, avoiding the time overhead associated with finding the partition combination with the smallest deviation. The specific process is outlined in Algorithm 1: First, sort the runtime of queries (RQ) in descending order (line 1). Lines (4-13) allocate the N queries one by one to the subset with the shortest current runtime. Specifically, lines (5-9) are used to find the MIN with the shortest current runtime, and lines (10-11) add the query with the shortest runtime to MIN and update the runtime of its corresponding subset.

Optimization of Query Execution Process. In practical optimization scenarios, there may be situations in which some subsets of queries have already completed execution, whereas others still have pending queries. To address this, the subsets that have finished execution can assist the subsets with pending queries, thereby preventing such situations. Additionally, because the subsets are partitioned based on the average execution time of queries from largest to smallest, the execution order of queries within each subset follows the same pattern; therefore, the pending queries also have shorter average execution times. This approach helps to minimize the deviation as much as possible.

Algorithm 1 Partition Execution Order

Input: Number of queries (N); Runtime of Queries (RQ); number of Parallelism (P); Queries SubSet (QSS); Runtime of Query SubSet ($RQSS$)
Output: QSS,$RQSS$

```
1: sort RQ in descending order
2: RQSS={}, QSS={}
3: MIN=RQSS_1 , k=1
4: for i = 1 to N do
5:    for j = 1 to P do
6:       if MIN>RQSS_j then
7:          MIN=RQSS_j , k=j
8:       end if
9:    end for
10:   RQSS_k +=RQ_i , QSS_k=QSS_k+Q_i
11:   MIN=RQSS_k
12: end for
13: return QSS,RQSS
```

$$REQS = \mathrm{Max}\left(RQSS_n\right)_{(n=[1,\mathrm{P}])} \tag{5}$$

where REQS is the Runtime of Entire Query Set; RQSS is the Runtime of Query SubSet;P is the number of parallelism.

4.2 Knowledge Transfer

When queries are executed in parallel, a problem arises: it is impossible to ensure that the order of query execution in multi-query applications remains consistent across multiple runs. This inconsistency has a significant impact on the optimization of multi-query applications. Even with the same configuration parameters, variations in the order of query execution can lead to differences in overall runtime. To address this issue, this study aims to minimize the impact of the query execution order on runtime, allowing for a more accurate evaluation of the configuration parameters. A simple approach is to calculate the theoretical minimum runtime for parallel execution under the current configuration parameters, based on the runtime of individual queries.

This study proposes a knowledge transfer method in which the runtime of each query under the given configuration parameters is used as knowledge. Following Algorithm 1, the theoretical minimum runtime for parallel execution under a given configuration is calculated. This minimum runtime is used as the runtime for the current configuration parameters. Finally, the updated data (knowledge) are used to train the surrogate model of BO, as detailed in Algorithm 2. By employing a knowledge transformation, more accurate runtime estimates can be obtained. The ablation experiments in Section 5.3 demonstrated the effectiveness of the knowledge transformation method.

Algorithm 2 Knowledge Transfer

Input: the Runtime of Queries in n-th search, m is the number of Searches ($RQ_{n(n\subseteq[1,\mathrm{m}])}$); number of Parallelism (P) .
Output: new Runtime of Entire Query Set($REQS^{new}$) .

1: $REQS^{new}$=Max=0
2: **for** $i = 1$ to m **do**
3: Max=0
4: $QSS_i, RQSS_i = PartitionExecutionOrder(RQ_i)$
5: **for** j=1 to P **do**
6: **if** Max<$RQSS_i^j$ **then**
7: Max=$RQSS_i^j$
8: **end if**
9: **end for**
10: $REQS_i^{new}$=Max
11: **end for**
12: **return** $REQS^{new}$

Lines 2-11: Based on the runtime of queries in searches, the query execution order is sequentially divided for the runtime of queries in every search. Subsequently, the runtime of queries in every search is transformed according to the theoretical minimum runtime. Specifically, lines 2-4: The query execution order is sequentially divided based on the runtime of queries in searches; lines 5-10:

The longest runtime of the subset (Max) is chosen as the runtime of entire query set. Finally, the new Runtime of Entire Query Set($REQS^{new}$) is returned.

4.3 Variable Bayesian Optimization

We employed Bayesian optimization [26] to search for the optimal configurations. Compared with classical Bayesian optimization, Variable Bayesian optimization has three variable features in this study: search space, optimization objective, and observation data. Next, we introduce the three variable features of variable Bayesian optimization.

Variable search space: When sufficient data is available (step7:the number of iterations is k), we begin to identify important configuration parameters (IICP [37]). The important configuration parameters are then considered as a new search space for Bayesian optimization, which accelerates the convergence speed of the algorithm.

Variable optimization objective: In multi-query applications, some queries are sensitive to configuration parameters, while others are not. When sufficient data are available (step 8: the number of iterations is j), we conduct query configuration sensitivity analysis (QCSA [37]) to identify configuration-sensitive queries (CSQ). We then use CSQ as a new optimization objective, which significantly reduce the optimization time cost

Variable observation data: Whenever new data is added, it is necessary to recalculate the execution times in the new data using Algorithm 2; After identifying the important configuration parameters (IICP), the configuration parameters in the training data need to be updated to the important configuration parameters; Subsequently, after Query Configuration Sensitivity Analysis (QCSA), the execution times in the training data must be recalculated based on the CSQ.

Next, we detail the setup of the variable Bayesian optimization in this study. VBO utilizes the Gaussian Process (GP) [31] as a surrogate model in Bayesian optimization. OPOM incrementally builds a GP model starting with three samples generated by Latin Hypercube Sampling(LHS) [14]. For the acquisition function, we selected the Expected Improvement (EI). By maximizing EI, we balance exploration and exploitation and efficiently navigate the parameter space towards promising configurations; the number of iterations k and j is set to 20 and 25 in the experiments in this study, respectively, which is the same setting as that of IICP and QCSA in LOCAT [37].

5 Experiment

To evaluate our framework, in this section, we list two insights that we want to investigate:1) **Generality**, our method consistently outperforms state-of-the-art (SOTA) spark-tuning methods on two Spark SQL benchmarks; and 2) **Efficiency**, which is equipped with the three techniques, accelerates the tuning process to different degrees.

5.1 Experiment Setup

Benchmark Programs. We select two benchmarks, TPC-H [4] and TPC-DS [27], as representative benchmarks to evaluate the performance of OPOM. To evaluate how OPOM adapts to changes in input data size, we employ three different data sizes (20G, 50G, 100G) in our experiments. **(1)TPC-H** is a benchmark test set for decision support, with 22 queries. TPC-H includes an instant query and concurrent data modification, which has more extensive industry relevance than TPC-DS. **(2)TPC-DS** contains 103 queries, which has been widely used in the Spark SQL system for the research and development of optimization techniques [7,17,30]. TPC-DS is a basic test for decision support that provides a common way to meet decision support systems, including data query and maintenance.

Environment. Our experimental platform is in the Huawei cloud environment, using a 3-node cluster including a master node and 2 slave nodes. All servers are connected to a 1-Gigabit Ethernet network, and each server is equipped with two Xeon(R) Gold 5218R @ 2.10GHz with 40 physical cores, 128 GB DDR4 memory, and 6TB HDD.

Baselines. We compare OPOM with the following SOTA BO-based tuning approaches: **(1)Cherrypick** [2] is a classic Bayesian Optimization method. Its initial sample points are selected from quasi-random sequences [35], the prior density function adopts the Gaussian process, and the acquisition function selects EI. Cherrypick adds noise to the optimization goal, which reduces the influence of cloud environment fluctuations on the experimental results.**(2)RoBOTune** [18] is a modified Bayesian Optimization method. It takes the Latin hypercube method to select the initial sample point. It adopts the adaptive Hedge strategy to calculate the acquisition function's gain and then uses the cumulative sum method to select the acquisition function. **(3)LOCAT** [37] is a method based on the Bayesian Optimization algorithm. It first selects the CSQ of the workload and then only optimizes the CSQ. Its acquisition function combines EI and the Markov Chain Monte Carlo (MCMC) hyperparameter marginalization algorithm. It is the first study to use QCSA, significantly reducing the optimization time.

Objective& Evaluation Metrics & Settings. (1)Objective. The tuning objective is runtime. **(2)Evaluation Metrics.** We use the following two metrics: **Speedup**. The Speedup of execution time of the best-found configuration relative to the default configuration. **Cost Reduction**. The Cost Reduction is relative to the OPOM method, which is defined as $\frac{\text{Cost} - \text{Cost}_{\text{opom}}}{\text{Cost}_{\text{opom}}}$, cost is the runtime of the entire optimization process. **(3)Settings.** The parallel mode of the compared methods employ a simple execution order partition method that sequentially runs queries from q_1 to q_n; when the number of concurrently running queries is less than the degree of parallelism, the following query in the query set is executed. The BO component is implemented using the Bayesian-optimization library [25]. The number of iteration rounds is 30, and the number of initial samples is 6 for all methods. All experiments are repeated three times to reduce errors.

Table 1: In the two benchmarks, the speedup (Cost Reduction%) comparison between the OPOM method and the other three SOTA methods in both serial and parallel modes. For example, cherrypick-4 refers to the cherrypick method running with parallelism=4. The parallelism of the OPOM method is 4. When the parallelism is 4, the performance is the best in our environment, and the space problem is not further introduced.

	OPOM	**cherrypick**	**RoBoTune**	**LOCAT**	**cherrypick-4**	**RoBoTune-4**	**LOCAT-4**
tpch-20G	6.52	2.81(154.29%)	2.85(146.65%)	2.89(132.00%)	4.59(50.42%)	4.82(43.47%)	5.26(23.94%)
tpch-50G	11.39	3.54(274.23%)	3.54(264.39%)	3.59(225.00%)	8.39(60.90%)	8.68(48.17%)	9.30(24.35%)
tpch-100G	12.13	3.39(290.84%)	3.68(277.26%)	3.73(231.99%)	8.09(61.86%)	9.14(49.61%)	9.75(24.40%)
tpcds-20G	5.78	2.06(208.57%)	2.10(202.52%)	2.16(169.00%)	4.15(49.80%)	4.35(45.28%)	4.78(21.00%)
tpcds-50G	8.25	2.53(249.11%)	2.77(234.72%)	2.83(193.00%)	5.39(61.90%)	6.06(51.20%)	6.50(26.80%)
tpcds-100G	9.74	3.01(258.47%)	3.03(245.68%)	3.09(218.37%)	6.46(62.15%)	6.83(52.91%)	7.46(30.54%)

5.2 Method Comparison

In this subsection, we first compare the OPOM method with three other SOTA methods, analyzing the advantages of the OPOM method from the perspectives of Speedup and Cost Reduction. To eliminate the influence of parallel execution, we modify the three SOTA methods to parallel execution methods and compare them with the OPOM method with the same parallelism. We conduct a comparative analysis from two perspectives: Speedup and Cost Reduction.

Speedup. As shown in Table 1, it can be observed that the OPOM method exhibits a significant improvement in speedup compared to the three SOTA methods in both serial and parallel execution modes. The considerable lead of the OPOM method over the serial execution modes of the three SOTA methods is particularly noteworthy.

In the six test cases under the two benchmarks, the maximum speedups for OPOM, Cherrypick, ROBOTune, and LOCAT are 12.13×, 3.39×, 3.68×, 3.73×, respectively, and the corresponding average speedups are 8.97×, 2.89×,3.00×,3.05×. For OPOM, Cherrypick-4, ROBOTune-4, and LOCAT-4, the maximum Speedup are 12.13×, 8.09×, 9.14×, 9.75×, respectively, with average Speedup of 8.97×, 6.18×, 6.65×, 7.18×, respectively.Among the methods excluding OPOM, LOCAT-4 has the highest Speedup. However, compared with LOCAT-4, OPOM achieves a maximum speedup increase of 2.88× and an average speedup increase of 1.79×.

Cost Reduction. as shown in Table 1, it can be observed that OPOM achieves significant results in terms of Cost Reduction compared to the other six methods, especially when compared to the three SOTA methods using serial execution. In these comparisons, the Cost Reduction decreased by several times.

In the six test cases of the two benchmarks, OPOM achieves the highest Cost Reduction compared to Cherrypick, ROBOTune, and LOCAT, with maximum reductions of 290.84%, 277.26%, and 231.99% and average reductions of 239.25%, 228.54%, and 194.89%, respectively. Compared to Cherrypick-4, ROBOTune-4, and LOCAT-4, OPOM achieves maximum Cost Reductions of 62.15%, 52.91%, and 30.54% and average reductions of 57.84%, 48.44%, and 25.17%, respectively.Excluding OPOM, the method with the lowest cost is LOCAT-4. Com-

Table 2: In two benchmarks, the Speedup and Cost Reduction comparisons between the OPOM method and three ablation methods; for example, PEO- refers to the OPOM method without PEO.

Benchmark \ method	OPOM	-PEO	-kT	-VBO
tpch-20G	6.52	5.63(18.00%)	6.27(7.00%)	5.69(14.55%)
tpch-50G	11.39	9.71(19.00%)	10.19(9.00%)	10.43(19.97%)
tpch-100G	12.13	9.82(21.00%)	11.29(6.00%)	10.87(13.64%)
tpcds-20G	5.68	5.02(17.00%)	5.19(8.00%)	4.54(19.21%)
tpcds-50G	8.14	6.79(19.00%)	7.59(9.00%)	6.93(18.81%)
tpcds-100G	9.88	8.15(22.00%)	9.09(7.00%)	8.79(11.83%)

pared with LOCAT-4, OPOM achieves a maximum Cost Reduction of 30.54% and an average Cost Reduction of 25.17%.

We draw the following conclusions from the comparative experiments: (1) Running applications in parallel significantly reduces execution time compared to serial execution, leading to a substantial improvement in Speedup and Cost Reduction. Parallel execution of applications allows for efficient utilization of resources, thus significantly reducing the time required for each run. Therefore, both Speedup and Cost Reduction are considerably enhanced. (2) Optimizing the query execution order is effective. The main difference between OPOM and LOCAT-4 lies in optimizing the query execution order. OPOM achieves an average improvement of approximately 25% compared with LOCAT-4, demonstrating the effectiveness of optimizing the query execution order.

5.3 Albation Studies

In this subsection, we further analyze the three compontents proposed in Section 4 to evaluate their effectiveness.

Partition Execution Order. As shown in Table 2, the maximum speedups achieved by OPOM and -PEO are 12.13× and 9.83×, respectively, with average speedup values of 8.96× and 7.52×. In comparison, OPOM exhibits a maximum Cost Reduction of 22% and an average Cost Reduction of 19.33% compared with -PEO. The experimental results align with the examples in Section 4.1, indicating that the Partitioning Execution Order enables queries running parallel to finish simultaneously, thus maximizing the resource utilization. To further validate this, we conducted a performance analysis by examining the proportion of concurrently running queries during the runtimes of OPOM and -PEO. As shown in Table 3, the PEO method significantly increases the proportion of four queries running simultaneously, thereby improving resource utilization. This certainly validates the effectiveness of the PEO method, which groups queries by their average runtime so that the total runtime of each group of query is essentially the same.

Knowledge Transfer. As shown in Table 2, the maximum speedups achieved by OPOM and -KT are 12.13× and 11.29×, respectively, with average speedup

Table 3: The proportion of concurrently running queries in the runtime of OPOM and -PEO.The data in the table is arranged as OPOM(-PEO).

Benchmark \ Parallelism	4	3	2	1
tpch-20G	94.05%(80.10%)	2.32%(4.21%)	1.71%(8.37%)	1.92%(7.33%)
tpch-50G	95.29%(80.07%)	1.81%(4.16%)	1.10%(10.43%)	1.81%(5.33%)
tpch-100G	96.32%(79.61%)	1.15%(4.45%)	1.19%(7.54%)	1.35%(8.40%)
tpcds-20G	93.72%(80.10%)	2.04%(5.21%)	2.15%(5.38%)	2.09%(9.31%)
tpcds-50G	96.33%(80.95%)	1.60%(6.37%)	0.93%(5.35%)	1.13%(7.33%)
tpcds-100G	97.31%(79.76%)	0.60%(5.53%)	1.27%(8.31%)	0.82%(6.39%)

values of 8.96× and 8.27×. In comparison, OPOM exhibits a maximum Cost Reduction of 9% and an average Cost Reduction of 7.67% compared to -KT. The experimental results confirm the effectiveness of knowledge transfer, primarily because adopting knowledge transfer avoids the impact of changes in execution order on the runtime during the optimization process.

Variable Bayesian Optimizaiton. As shown in Table 2, the maximum speedups achieve by OPOM and -VBO are 12.13× and 10.87×, respectively, with average speedup values of 8.96× and 7.88×. In comparison, OPOM exhibits a maximum Cost Reduction of 19.97% and an average Cost Reduction of 16.33% compared with -VBO. The difference between OPOM and -VBO lies in whether to adopt the Identifying Important Configuration Parameters (IICP) and Query Configuration Sensitivity Analysis (QCSA) methods. The experimental section of LOAT [37] elaborately verifies the effectiveness and reasons behind these two methods. The experimental results of this paper are consistent with theirs, further confirming the effectiveness of these two methods. ICCP can identify important parameters in the configuration, whereas QCSA can distinguish sensitive queries from queries. These two methods significantly reduce the complexity of the optimization problem by reducing costs and improving the performance.

6 Related Work

In the Spark computing environment, the existing methods for optimizing configuration parameters can fall into two categories. (1) Machine learning methods [36,38,19,5,24]: Construct a prediction model using the machine learning method, and then determine the optimal configuration within the prediction model. However, this method requires a large number of samples to ensure the accuracy of the prediction model for determining the optimal configuration. Otherwise, the performance of the obtained configuration is often worse than that of the sample configuration. (2) Search methods [11,12,20,9,41]: Belonging to the methods discussed in this study. We select observation points in the parameter search space based on the search algorithm and then select the next round of observation points based on feedback until the algorithm converges or reaches

a stopping condition. However, such methods are prone to falling into the local optima.

For multi-query applications, the main challenge lies in the cost of optimization time being too long. Recent studies have been conducted to reduce the cost of optimization time, and The Rover [33] method is an online Spark SQL tuning service that improves tuning performance based on external knowledge and reduces optimization costs. It introduces general transfer learning, including expert-assisted Bayesian optimization and controls historical transfer. Guo et al. [13] introduced a GML approach employing GAN to generate training data with higher average performance, thereby reducing the time needed for data collection. However, a considerable amount of data is required for model training. Bao et al. [3] utilized the test platform method to construct a prediction model, reducing the costs associated with sample collection by considering factors like actual data size, cluster size, and scale reduction. Nevertheless, simulating the black box model using these indicators poses challenges, and the experimental results are inconclusive. Cheng et al. [6] employed projection sampling for iterative modeling, enhancing prediction model accuracy and significantly decreasing sampling time costs, albeit requiring substantial data collection.

In multi-query applications, Jasper Snoek et al. [37] first introduced the Query Configuration Sensitivity Analysis (QCSA) method to identify configuration-sensitive queries (CSQ) from all queries. They then replaced the optimization target from all queries with CSQ, significantly reducing the optimization time cost. Based on this work, our study further investigated altering the execution method of queries and integrating query execution order with parallelism to further reduce the optimization time cost.

7 CONCLUSION

In this paper, we introduced the OPOM method, which considers the execution order of queries in the optimization of Spark multi-query applications based on Bayesian optimization. The OPOM method, along with PEO, KT, and VBO, played a significant role in the optimization process in the Huawei cloud environment. PEO significantly increases the proportion of queries running simultaneously at the level of parallelism in the overall execution time, thus enhancing resource utilization. KT optimizes the execution order of each query, mitigating the impact of inconsistent query execution orders on overall runtime. VBO narrows the search space for configuration parameters and optimizes the number of queries, resulting in a significant reduction in the optimization time. In future work, we will further consider the characteristics of queries and explore methods to optimize the execution order of queries.

References

1. Apache flink: Stream and batch processing in a single engine

2. Alipourfard, O., Liu, H.H., Chen, J., Venkataraman, S., Yu, M., Zhang, M.: Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In: Symposium on Networked Systems Design and Implementation (2017)
3. Bao, L., Liu, X., Chen, W.: Learning-based automatic parameter tuning for big data analytics frameworks. Big Data pp. 181–190 (2018)
4. Boncz, P.A., Neumann, T., Erling, O.: Tpc-h analyzed: Hidden messages and lessons learned from an influential benchmark. In: TPC Technology Conference (2013)
5. Cheng, G., Ying, S., Wang, B.: Tuning configuration of apache spark on public clouds by combining multi-objective optimization and performance prediction model. J. Syst. Softw. **180**, 111028 (2021)
6. Cheng, G., Ying, S., Wang, B., Li, Y.: Efficient performance prediction for apache spark. J. Parallel Distributed Comput. **149**, 40–51 (2021)
7. Chiba, T., Yoshimura, T., Horie, M., Horii, H.: Towards selecting best combination of sql-on-hadoop systems and jvms. CLOUD pp. 245–252 (2018)
8. Dadu, V., Liu, S., Nowatzki, T.: Polygraph: Exposing the value of flexibility for graph processing accelerators. ISCA pp. 595–608 (2021)
9. Du, H., Han, P., Xiang, Q., Huang, S.: Monkeyking: Adaptive parameter tuning on big data platforms with deep reinforcement learning. Big data (2020)
10. Duan, S., Thummala, V., Babu, S.: Tuning database configuration parameters with ituned. Proc. VLDB Endow. **2**, 1246–1257 (2009)
11. Fekry, A., Carata, L., Pasquier, T., Rice, A.C., Hopper, A.: To tune or not to tune?: In search of optimal configurations for data analytics. SIGKDD (2020)
12. jiao Gong, Y., Li, J., Zhou, Y., Li, Y., hung Chung, H.S., hui Shi, Y., Zhang, J.: Genetic learning particle swarm optimization. IEEE Transactions on Cybernetics **46**, 2277–2290 (2016)
13. Guo, Y., Shan, H., Huang, S., Hwang, K., Fan, J., Yu, Z.: Gml: Efficiently autotuning flink's configurations via guided machine learning. IEEE Transactions on Parallel and Distributed Systems **32**, 2921–2935 (2021)
14. Helton, J.C., Davis, F.J.: Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems. Reliab. Eng. Syst. Saf. **81**, 23–69 (2002)
15. Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F.B., Babu, S.: Starfish: A self-tuning system for big data analytics. In: Conference on Innovative Data Systems Research (2011)
16. Hochbaum, D.S.: Approximation algorithms for np-hard problems. SIGACT News **28**, 40–52 (1997)
17. Ivanov, T., Beer, M.G.: Evaluating hive and spark sql with bigbench. ArXiv **abs/1512.08417** (2015)
18. Khan, M.M., Yu, W.: Robotune: High-dimensional configuration tuning for cluster-based data analytics. Proceedings of the 50th International Conference on Parallel Processing (2021)
19. Li, M., Liu, Z., Shi, X., Jin, H.: Atcs: Auto-tuning configurations of big data frameworks based on generative adversarial nets. IEEE Access **8**, 50485–50496 (2020)
20. Liao, G., Datta, K., Willke, T.L.: Gunther: Search-based auto-tuning of mapreduce. In: European Conference on Parallel Processing (2013)
21. Meng, X., Bradley, J.K., Yavuz, B., Sparks, E.R., Venkataraman, S., Liu, D., Freeman, J., Tsai, D.B., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M.J., Zadeh, R.B., Zaharia, M.A., Talwalkar, A.: Mllib: Machine learning in apache spark. J. Mach. Learn. Res. **17**, 34:1–34:7 (2015)

22. Mockus, J.: On bayesian methods for seeking the extremum and their application. In: IFIP Congress (1977)
23. Nambiar, R.O., Pöss, M.: The making of tpc-ds. In: Very Large Data Bases Conference (2006)
24. Nguyen, N., Khan, M.M.H., Wang, K.: Towards automatic tuning of apache spark configuration. CLOUD pp. 417–425 (2018)
25. Nogueira, F.: Bayesian Optimization: Open source constrained global optimization tool for Python (2014–)
26. Pelikán, M., Goldberg, D.E.: Hierarchical bayesian optimization algorithm. In: Scalable Optimization via Probabilistic Modeling (2006)
27. Poess, M.: Tpc-ds. Encyclopedia of Big Data Technologies (2019)
28. Pöss, M., Nambiar, R.O., Walrath, D.: Why you should run tpc-ds: A workload analysis. In: Very Large Data Bases Conference (2007)
29. Pöss, M., Smith, B., Kollár, L., Larson, P.Å.: Tpc-ds, taking decision support benchmarking to the next level. In: SIGMOD (2002)
30. Ramdane, Y., Boussaïd, O., Kabachi, N., Bentayeb, F.: Partitioning and bucketing techniques to speed up query processing in spark-sql. ICPADS pp. 142–151 (2018)
31. Seeger, M.W.: Gaussian processes for machine learning. International journal of neural systems **14 2**, 69–106 (2004)
32. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., de Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE **104**, 148–175 (2016)
33. Shen, Y., Ren, X., Lu, Y., Jiang, H., Xu, H., Peng, D., Li, Y., Zhang, W., Cui, B.: Rover: An online spark sql tuning service via generalized transfer learning. SIGKDD (2023)
34. Singhal, R., Singh, P.K.: Performance assurance model for applications on spark platform. In: TPC Technology Conference (2017)
35. Soboĺ, I.M.: On quasi-monte carlo integrations. Mathematics and Computers in Simulation **47**, 103–112 (1998)
36. Wang, G., Xu, J., He, B.: A novel method for tuning configuration parameters of spark based on machine learning. HPCC/SmartCity/DSS pp. 586–593 (2016)
37. Xin, J., Hwang, K., Yu, Z.: Locat: Low-overhead online configuration auto-tuning of spark sql applications. Proceedings of the 2022 International Conference on Management of Data (2022)
38. Yu, Z., Bei, Z., Qian, X.: Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing. Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (2018)
39. Zaharia, M.A., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster computing with working sets. In: USENIX Workshop on Hot Topics in Cloud Computing (2010)
40. Zaharia, M.A., Xin, R., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J.E., Shenker, S., Stoica, I.: Apache spark. Communications of the ACM **59**, 56 – 65 (2016)
41. Zhu, Y., Liu, J., Guo, M., Bao, Y., Ma, W., Liu, Z., Song, K., Yang, Y.: Bestconfig: tapping the performance potential of systems via automatic configuration tuning. Proceedings of the 2017 Symposium on Cloud Computing (2017)