Research Article

Efficient ELM-Based Two Stages Query Processing Optimization for Big Data

Linlin Ding, 1 Yu Liu, 1 Baoyan Song, 1 and Junchang Xin 2

1 School of Information, Liaoning University, Shenyang, Liaoning 110036, China
2 College of Information Science & Engineering, Northeastern University, Shenyang, Liaoning 110819, China

Correspondence should be addressed to Baoyan Song; bysong@lnu.edu.cn

Received 22 August 2014; Accepted 27 November 2014

Academic Editor: Yi Jin

Copyright © 2015 Linlin Ding et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

MapReduce and its variants have emerged as viable competitors for big data analysis with a commodity cluster of machines. As an extension of MapReduce, ComMapReduce realizes the lightweight communication mechanisms to enhance the performance of query processing applications for big data. However, different communication strategies of ComMapReduce can substantially affect the executions of query processing applications. Although there is already the research work that can identify the communication strategies of ComMapReduce according to the characteristics of the query processing applications, some drawbacks still exist, such as relative simple model, too much user participation, and relative simple query processing execution. Therefore, an efficient ELM-based two stages query processing optimization model is proposed in this paper, named ELM to ELM (E2E) model. Then, we develop an efficient sample training strategy to train our E2E model. Furthermore, two query processing executions based on the E2E model, respectively, Just-in-Time execution and Queue execution, are presented. Finally, extensive experiments are conducted to verify the effectiveness and efficiency of the E2E model.

1. Introduction

Nowadays, MapReduce [1] has become a widespread programming framework for big data analysis. Hadoop (http://hadoop.apache.org/) is a popular open-source implementation of MapReduce that many academies and industrial organizations adopt it in research and production deployments. The success of MapReduce stems from hiding the details of parallelization, fault-tolerance, and load balancing in a simple programming framework. MapReduce and its variants are used to process the big data applications, such as Web indexing, data mining, machine learning, financial analysis, scientific simulation, and bioinformatics research [2–10].

As one of the successful extensions of MapReduce, ComMapReduce [3, 4] adds simple lightweight communication mechanisms to generate the certain shared information and implements the query processing applications for big data. In ComMapReduce framework, three basic and two optimization communication strategies are presented to solve the problem of how to communicate and obtain the shared information of different applications. After further analyzing the ComMapReduce execution course and the abundant experiments, we find out that different communication strategies of ComMapReduce can substantially impact the performance of query processing applications.

To identify the communication strategy, the existing research work [11] proposes a query optimization model, named ELM CMR, based on ELM [12] which has the classification performance at an excellent standard. According to the characteristics of query processing programs, ELM CMR can identify the communication strategy of ComMapReduce. However, ELM CMR still has the following drawbacks. First, regardless of the MapReduce or ComMapReduce framework, a program only consists of black_box Map and Reduce functions, without knowing the distributed details about the framework. But the configuration parameters of the framework can fully influence the performance of query processing. Finding the most suitable configuration parameters setting itself is difficult. The burden falls on the user who submits the MapReduce or ComMapReduce job to specify
settings for all configuration parameters, which highlights the challenges the user faces. How to identify the proper configuration parameters according to the user program is a difficult problem. Second, the implementation algorithm of multiple queries in ELM_CMR only processes the queries based on the classification results and predicted execution time, but there is no consideration of the characteristics among different queries. Different queries can share computation and data as to enhance the performance further. Third, ELM_CMR only adopts simple training method to gain the classification model. If more efficient training method is adopted to obtain the training data, the accuracy of the classification model can be improved.

Therefore, in this paper, for resolving the above problems and drawbacks, we propose an efficient ELM-based two stages query processing optimization model, named E2E model. According to the characteristics of the users programs, the first stage can gain the feature parameters using ELM algorithm. After that, according to the results of the first stage, the second stage can identify the final classification result by ELM algorithm too. Furthermore, an efficient sample training strategy is presented to train the E2E model. We also develop two query processing executions based on the E2E model, Just-in-Time execution and Queue execution. The contributions of this paper can be summarized as follows.

(i) We propose an efficient ELM-based two stages query processing optimization model, named E2E model, which can realize the most optimal executions of query processing applications in MapReduce or ComMapReduce framework.

(ii) We develop a sample training strategy to train our E2E model and two query executions based on E2E model, Just-in-Time execution, and Queue execution.

(iii) The experimental studies using synthetic data show the effectiveness and efficiency of the E2E model.

The remainder of this paper is organized as follows. Section 2 briefly introduces the background, containing the ELM and ELM_CMR. Our E2E model is proposed in Section 3. The two executions for query processing applications based on E2E are presented in Section 4. The experimental results to show the performance of E2E model are reported in Section 5. Finally, we conclude this paper in Section 6.

2. Background

In this section, we describe the background of our work, which includes a brief overview of the traditional ELM and the detailed descriptions of the ELM_CMR.

2.1. Review of ELM. Nowadays, extreme learning machine (ELM) [12] and its variants [13–28] have the characteristics of excellent generalization performance, rapid training speed, and little human intervene, which have attracted extensive attention from more and more researchers. ELM is originally designed for the single hidden-layer feedforward neural networks (SLFNs [29]) and is then extended to the "generalized" SLFNs. ELM algorithm first randomly allocates the input weights and hidden layer biases and then analytically computes the output weights of SLFNs. Contrary to the other conventional learning algorithms, ELM reaches the optimal generalization performance at a very fast learning speed. ELM is less sensitive to the user defined parameters, so it can be deployed fast and convenient. That is the reason we choose ELM as our basic method to construct the query processing optimization model.

For $N$ arbitrary distinct samples $(x_j, t_j)$, where $x_j = [x_{j1}, x_{j2}, \ldots, x_{jn}]^T \in \mathbb{R}^n$ and $t_j = [t_{j1}, t_{j2}, \ldots, t_{jm}]^T \in \mathbb{R}^m$, standard SLFNs with hidden nodes $L$ and activation function $g(x)$ are mathematically modeled as

$$\sum_{i=1}^{L} \beta_i g_i(x_j) = \sum_{i=1}^{L} \beta_i (w_i \cdot x_j + b_i) = o_j \quad (j = 1, 2, \ldots, N),$$

where $L$ is the number of hidden layer nodes, $w_i = [w_{i1}, w_{i2}, \ldots, w_{im}]^T$ is the weight vector between the $i$th hidden node and the input nodes, $\beta_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^T$ is the weight vector connecting the $i$th hidden node and the output nodes, $b_i$ is the threshold of the $i$th hidden node, and $o_j = [o_{j1}, o_{j2}, \ldots, o_{jm}]^T$ is the $j$th output vector of the SLFNs.

The standard SLFNs can approximate these $N$ samples with zero error. The error of ELM is $\sum_{j=1}^{N} \|o_j - t_j\| = 0$ and there exist $\beta_i, w_i$, and $b_i$ such that

$$\sum_{i=1}^{L} \beta_i g(w_i \cdot x_j + b_i) = t_j \quad (j = 1, 2, \ldots, N).$$

Equation (2) can be expressed compactly as follows:

$$H\beta = T,$$

where

$$H(w_1, w_2, \ldots, w_L, b_1, b_2, \ldots, b_L, x_1, x_2, \ldots, x_N) = \begin{bmatrix} h(x_1) \\ h(x_2) \\ \vdots \\ h(x_N) \end{bmatrix} = \begin{bmatrix} g(w_1 \cdot x_1 + b_1) & g(w_2 \cdot x_1 + b_2) & \ldots & g(w_L \cdot x_1 + b_L) \\ g(w_1 \cdot x_2 + b_1) & g(w_2 \cdot x_2 + b_2) & \ldots & g(w_L \cdot x_2 + b_L) \\ \vdots & \vdots & \ddots & \vdots \\ g(w_1 \cdot x_N + b_1) & g(w_2 \cdot x_N + b_2) & \ldots & g(w_L \cdot x_N + b_L) \end{bmatrix}_{N \times L},$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \beta_2^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m}, \quad T = \begin{bmatrix} t_1^T \\ t_2^T \\ \vdots \\ t_N^T \end{bmatrix}_{N \times m}.$$

$H$ is set as the hidden layer output matrix of the neural network. The $i$th column of $H$ is called the $i$th hidden node.
output with respect to inputs \( x_1, x_2, \ldots, x_N \). The smallest norm least-squares solution of the above multiple regression system is shown as follows:

\[
\hat{\beta} = H^T \mathbf{T},
\]

where \( H^T \) is the Moore-Penrose generalized inverse of matrix \( H \). Then the output function of ELM can be modeled as follows:

\[
f(x) = h(x)^T \beta = h(x)^T H^T \mathbf{T}.
\]

The computational course for ELM is shown in Algorithm 1. Only after properly setting the related parameters, ELM can start the training process. The first step is to generate \( L \) pairs of hidden node parameters \((w_i, b_i)\) (Lines 1–3). The second step actually calculates the hidden layer output matrix \( H \) by using (4) (Line 4). The third step mainly computes the corresponding output weight vector \( \hat{\beta} \) (Line 5). After completing the above training process, the output of the new dataset can be predicted by ELM according to (7).

2.2. ELM_CMR Model. ComMapReduce [3, 4] is an improved MapReduce framework with lightweight communication mechanisms. A new node, named the Coordinator node, is added to store and generate the certain shared information of different applications. In ComMapReduce, three basic communication strategies, LCS, ECS, and HCS, and two optimization communication strategies, PreOS and PostOS, are proposed to identify how to receive and generate the shared information. In short, without affecting the existing characteristics of the original MapReduce framework, ComMapReduce is a successful parallel programming framework with global shared information to filter the unpromising data of query processing programs.

The existing ELM_CMR [11] is an efficient query processing optimization model based on ELM. It can identify the communication strategies of query processing applications in ComMapReduce according to the features of queries. Figure 1 shows the architecture of ELM_CMR model. The four components of ELM_CMR are, respectively, the Feature Selector, the ELM Classifier, the Query Optimizer, and the Execution Fabric.

The Feature Selector mainly examines the training query processing programs and selects the configuration parameters that can wholly affect the query performance by the job profiles. Naturally, the parameters of program \( p \) can be divided into three types, parameters that predominantly affect Map task execution; parameters that predominantly affect Reduce task execution; and the cluster parameters. Then, in each cluster, we adopt the minimum-redundancy-maximum-relevance (mRMR) [30] feature selection to find the optimal parameters sharply affecting the performance. And then, we generate the globally optimal configuration parameter settings by combining the results of each subspace. Therefore, the near-optimal configuration parameter setting can be generated.

After selecting the features of training data, the Feature Selector sends the extracted training data to the ELM Classifier. It uses the training data to construct the ELM model by the traditional ELM algorithm. After that, when there are one or multiple queries to be processed, the ELM Classifier can rapidly obtain the classification results of the queries and then sends them to the Query Optimizer.

The Query Optimizer applies the classification results of the ELM Classifier and combines the implementation patterns to choose an optimized execution order. After gaining the execution order, the query is sent to the Execution Fabric.

The Execution Fabric implements the program in ComMapReduce framework. When there is one query to be processed, the Execution Fabric implements the query according to the classification result of the Query Optimizer in ELM_CMR. When there are multiple queries to be processed, the multiple queries can be classified by ELM Classifier and gain the best communication strategy of each program. Then, a Task Scheduler Simulator is used to simulate the execution time of queries. According to the execution time and the classification results of the queries, the Query Optimizer designs an execution order following the common principle of Shortest Job First (SJF) to implement multiple queries.

3. E2E Model

In this section, the overview of our E2E model is introduced first in Section 3.1. Then, training the E2E model is shown in Section 3.2. Finally, predicting the E2E model is proposed in Section 3.3.

3.1. Overview of E2E Model. Our E2E model can identify the optimal communication strategies of query processing programs in MapReduce or ComMapReduce, which contains three main phases, respectively, the training phase, the prediction phase, and the execution phase. Figure 2 shows the whole workflow of query processing in the E2E model. The main workflow is as follows.

First, the training phase is responsible for extracting the training query processing programs that have a large effect on the E2E model. In the training phase, we use sample-based training strategies to run our workload, in isolation, pairwise, and at several higher multiprogramming levels. After gaining the training samples, they can be used to generate the E2E model in the prediction phase. The details of training phase will be introduced in Section 3.2.

Second, in the prediction phase, by using the training samples from the training phase, the two stages E2E model can be generated based on the traditional ELM algorithm. According to the user’s programs, the first stage can obtain the most optimal feature parameters of the programs by ELM. And then, using the feature parameters of the first stage, the second stage can identify the classification results of the query processing programs by ELM. The details of the prediction phase will be shown in Section 3.3.

Third, after gaining the E2E model, for the query processing programs submitted to MapReduce or ComMapReduce,
Algorithm 1: ELM.

(1) for \( i = 1 \) to \( L \) do
(2) Randomly generate the hidden node parameters \((w_i, b_i)\);
(3) end for
(4) Calculate the hidden layer output matrix \( H \);
(5) Calculate the output weight vector \( \beta = H^T \).

Figure 1: Architecture of ELM CMR model.

Figure 2: Workflow of E2E model.
we can predict their optimal communication strategies in the execution phase. We propose two query executions based on E2E model, the Just-in-Time execution which tackles one query processing program, and the Queue execution which tackles multiple concurrent queries. The details of the execution phase will be illustrated in Section 4.

3.2. Training E2E Model. To obtain the prediction model more accurately, we need to train the E2E model. Different from the simple training course of ELM_CMR model, the training phase of our E2E model consists of running the queries in isolation, pairwise as well as at several higher multiprogramming levels. In addition, in order to gain the measurements that capture the interactions, we omit the first few samples in our experiments. This approach makes us ignore the overhead of the initial query setup and caching the initial supporting structures.

The E2E model realizes the training phase by sampling approach, containing isolation, pairwise, and higher degree of concurrency, which allows us to approximate the running of queries in MapReduce or ComMapReduce. The course of the training phase is displayed in Figure 2. First, we sample the workload in isolation to gain how each query behaves, which can be seen as the baseline of training. Second, according to the query types in our experiments, we build a matrix of interaction by running all unique pairwise combinations. Pairwise sample can help us to simply estimate the degree of concurrency. Here, Latin hypercube sampling approach (LHS) can uniformly distribute our samples throughout our prediction space, which can be realized by creating a hypercube with the same dimension as the multiprogramming level. We adopt LHS to sample at pairwise or several higher multiprogramming levels. We then select the samples that every value on every plane gets inter selected exactly one. A simple example of two-dimensional LHS is shown in Table 1. In Table 1, Query-A, Query-B, Query-C, and Query-D stand for four queries to be processed by pairwise combinations. After combining using LHS, the four queries can obtain the optimal combinations.

Furthermore, the above strategy shows how to run the samples, in isolation, pairwise, and at several higher multiprogramming levels using LHS. As we know, the number of queries to be processed is large, and in order to enhance the training accuracy, we need to choose the queries that is the representative queries in each query type and then construct the samples to be trained. The corresponding cluster algorithms based on DBSCAN [31] can be also adopted to gain the most representative query in each query type. In practice, although the training time of E2E may be long, it is worth mentioning that training phase allows our model to be extremely lightweight once it reaches the prediction phase and enhance the accuracy.

3.3. Predicting E2E Model. Then, we introduce the details of the prediction phase. A MapReduce job $j$ can be expressed by a MapReduce program $p$ running on input data $d$ and cluster $r$, which can be expressed as $j = \langle p, d, r, c \rangle$ in short. We call the $d$, $r$, and $c$ the feature parameters of $p$. The users only submit their jobs to MapReduce or ComMapReduce without knowing the internal configuration details of the system.

However, a number of choices can be made in order to fully specify how the job should be executed. These choices, represented by $c$ in $(q, d, r, c)$, stand for a high dimensional space of configuration parameter settings, such as the number of Map and Reduce task, the block size, and the amount of memory. The performance changes a lot in different configuration parameters. For any parameter, its value is not specified explicitly during job submission, the default values either shipped with the system or specified by the system administrator. However, the normal users do not understand the running details of MapReduce. That is to say, finding good configuration settings for MapReduce job is time consuming and requires extensive knowledge of system internals. Because the users have little information of the parallelization details of MapReduce, it is necessary to gain the suitable configuration parameters. The first stage of our E2E model is to generate the model for identifying the suitable configuration parameters of query processing programs by ELM. The main goal of the first stage is to construct a black box feature parameters of queries, containing $(d, r, c)$. The configuration parameters wholly affecting the performance adopted by E2E are the same as our ELM_CMR approach as shown in Table 2. The corresponding information of the job can be obtained from sampling a few tasks of the job. After the first stage, the feature parameters setting can be obtained by ELM algorithm.

### Table 1: An example of LHS.

<table>
<thead>
<tr>
<th>Query</th>
<th>Query-A</th>
<th>Query-B</th>
<th>Query-C</th>
<th>Query-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query-A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query-B</td>
<td>▲</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query-C</td>
<td></td>
<td>▲</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query-D</td>
<td>▲</td>
<td></td>
<td>▲</td>
<td></td>
</tr>
</tbody>
</table>

### Table 2: Feature parameters in the experiments.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>io.sort.mb</td>
<td>int</td>
<td>100</td>
</tr>
<tr>
<td>io.sort.factor</td>
<td>int</td>
<td>10</td>
</tr>
<tr>
<td>min.num.spills.for.combine</td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>mapred.compress.map.output</td>
<td>boolean</td>
<td>false</td>
</tr>
<tr>
<td>mapred.reduce.parallel.copies</td>
<td>int</td>
<td>5</td>
</tr>
<tr>
<td>mapred.reduce.copy.backoff</td>
<td>int</td>
<td>300</td>
</tr>
<tr>
<td>dfs.heartbeat.interval</td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>dfs.block.size (M)</td>
<td>int</td>
<td>64</td>
</tr>
<tr>
<td>mapred.map.task</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>mapred.reduce.task</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>mapred.tasktracker.map.task.maximum</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>mapred.tasktracker.reduce.task.maximum</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>Data size (G)</td>
<td>int</td>
<td>10</td>
</tr>
<tr>
<td>Data distribution</td>
<td>boolean</td>
<td>uniform</td>
</tr>
<tr>
<td>Number of slave nods</td>
<td>int</td>
<td>8</td>
</tr>
</tbody>
</table>
After gaining the feature parameters of \( j \) by the first stage of E2E model; (2) Generate the execution plan of \( j \) by the second stage of E2E model; (3) Execute \( j \) with its communication strategy.

**Algorithm 2:** Just-in-Time execution.

4. **Executions of E2E Model**

After generating the E2E model, the pending queries can be implemented under our E2E model. In this section, two scenarios are proposed for executing query processing programs using the E2E model. In the first scenario, one new query is being submitted for immediate execution, named as Just-in-Time execution. In the second scenario, a Queue-based multiple concurrent queries execution is proposed, where an ordered list of queries to run is gained.

4.1. **Just-in-Time Execution.** When there is one new query being submitted, the E2E model generates its classification result as soon as possible. According to the characteristics of the coming query, the first stage of E2E model can generate the feature parameters of this query based on ELM. After that, the user can make a decision that whether adopting ComMapReduce or which communication strategy can be adopted by the second stage of E2E model, and then implements the query processing application.

The course of Just-in-Time execution is shown in Algorithm 2. First, the most optimal feature parameters of query processing job \( j \) are extracted in the first stage of E2E model (Line 1). Second, after obtaining the feature parameters of job \( j \), the E2E generates the classification result of \( j \) (Line 2). Third, according to the classification of \( j \), the E2E ensures how to implement the query and sends it to MapReduce or ComMapReduce framework. The framework uses the optimization result to execute the query program (Line 3).

For example, for a submitted skyline query, the first stage of E2E can obtain the most optimal feature parameters of this skyline query. After abstracting its feature parameters, the E2E can generate its classification and then identifies the communication strategy of this skyline query, such as PostOS. After that, the skyline query will be implemented in ComMapReduce with PostOS.

4.2. **Queue Execution.** When there are multiple concurrent queries to be processed, we design an efficient execution order of queries based on E2E model. Under the execution order, the performance of multiple concurrent queries can reach a nice status. Of course, we can execute these queries one by one with their classification results of E2E, but a lot of calculations and execution time would be wasted. Therefore, the execution order is important to enhance the performance of multiple concurrent queries.

As we know, different queries usually contain the overlap parts, such as the same query type and the same processing data. Therefore, when processing multiple concurrent queries, if we can identify the correlations of query type and processing data of the multiple queries, then the sharing computation and data can be used to prevent the repeated processing and scanning.

The main processing course of Queue execution is shown in Figure 3. There are three main components in the Queue execution, respectively, the Analyzer, the Optimizer, and the Executor.

First, for the multiple concurrent queries to be executed, after obtaining the classification results by E2E model, the Analyzer analyzes the queries and divides the queries into some subsets by considering the share of computation. This dividing approach can divide the queries with the same query type into the same subset, then the queries in a subset can be computed together.

Second, in each subset, the Optimizer can design the local execution order according to the characteristics of the sharing computation and data. It can analyze the data being processed of the multiple concurrent queries in the same subset, and then the queries with sharing data can share the computation results of the sharing data. The Optimizer also can cache the corresponding intermediate results into memory or buffer by the characteristics of the queries so as to reduce the repeated implementations of the same operations and I/O cost. Then, similar to our former ELM-CMR, the Optimizer also uses a Task Scheduler Simulator to simulate the scheduling and executions of Map and Reduce tasks of each query. The execution time of a job \( j \) can be estimated by the Task Scheduler Simulator. So, according to the common principle of Shortest Job First (SJF) too, the execution order of each subset is gained.

Finally, after obtaining the execution order of each subset, the Executor also uses the SJF principle of the subsets by simple merging their execution time of each subset, so the global execution order (O.S) of the multiple queries is the ascending order of the simulated execution time of each subset. Then, the multiple queries can be implemented under the global execution order.

Algorithm 3 illustrates the whole execution course of the multiple concurrent queries. First, we can obtain the classification of each query by the E2E model (Lines 1–3). Then, the Analyzer divides the queries sharing computation into some subsets (Line 4). Then, in each subset, the Optimizer
can analyze and divide the sharing data, and then computes
the sharing data. After gaining the simulated time of each
query in each subset, the local execution order is obtained by
considering the sharing and features of the queries in each
subset by SJF (Lines 5–10). Finally, the global execution order
of the queries is generated by SJF too (Line 11).

Figure 4 shows an example of the implementation of the
multiple concurrent queries. Suppose that there are eight
queries to be processed. First, these queries can obtain their
classification results by $E2E$. After that, the Analyzer divides
these queries into four subsets by the sharing of computation,
the same query type, respectively, top-$k$, kNN, skyline, and
join.

In each subset, the Optimizer can make a local execution
order by the features of the queries. For example, the subset
of skyline query contains three skyline queries, $q_2, q_5,$ and
$q_8$, with their processing data sets $s_2 = \{p_1, p_2, p_3, p_4, p_5, p_6\}$,
$s_5 = \{p_1, p_2, p_3, p_7, p_8, p_9\},$ and $s_8 = \{p_4, p_5, p_6, p_7, p_8, p_9\}.$
For these skyline queries, the Optimizer divides the sharing
data and gains the following sharing subdatasets, $s_a =
\{p_1, p_2, p_3\}, s_b = \{p_4, p_5, p_6\},$ and $s_c = \{p_7, p_8, p_9\}.$ Then, the
Optimizer can transfer the above skyline queries into some
simple skyline queries, $s_2 = s_a \cup s_b, s_5 = s_a \cup s_c,$ and $s_8 =
s_1 \cup s_c$. So, the Optimizer can make a solution that the skyline
queries first compute the results of $s_a, s_b, and s_c, and then
gain a local execution order by SJF, $q_2, q_5,$ and $q_8.$ The similar
processing course is fit for the top-$k$ and kNN queries. For
our join query of small-big tables, the Optimizer can cache
the corresponding small table into memory of Map tasks so
as to reduce the repeated scan of the small table. Finally, the
global execution order $O_1, q_7, q_1, q_5, q_8, q_2, q_3, q_4,$ and $q_6$ can
be gained by simple merge of SJF of the four subsets.

5. Performance Evaluation

In this section, the performance of our $E2E$ model is evaluated
detail with various experimental settings. We first describe
the setup used in our experiments in Section 5.1. Then we
present and discuss the experimental results in Section 5.2.

5.1. Experimental Setup. The experimental setup is the same
as $ELM\_CMR$ model as follows. The experimental setup
is a Hadoop cluster running on 9 nodes in a high speed
Gigabit network, with one node as the Master node and
the Coordinator node and the others as the Slave nodes.
Each node has an Intel Quad Core 2.66GHz CPU, 4 GB
memory, and CentOS Linux 5.6. We use Hadoop 0.20.2 and
compile the source codes under JDK 1.6. The ELM algorithm
is implemented in MATLABR2009a.

$ELM\_CMR$ is an efficient query processing optimization
model based on ELM. It can identify the communication
strategies of query processing applications in ComMapReduce
due to the features of queries. So far, except the
$ELM\_CMR$ model, there is no any other method to study how
to choose the optimal strategies of ComMapReduce for query
processing applications. So, we evaluate the performance of
$E2E$ model in different implementations for Just-in-Time execution
and Queue execution by comparing with $ELM\_CMR$.

In order to fully evaluate the performance of $E2E$ model,
four typical query processing applications are adopted to
5.2. Experimental Results. First, we give the classification results of \textit{E2E} model and \textit{ELM\_CMR} model by processing the above four query processing applications in different data size shown in Table 3. We can see that for the same query type and data size, the classification results of the queries are different, so the performance of the two models is different too. In the following, we evaluate the performance of the queries according to the classification results of Table 3.

Second, the performance of Just-in-Time execution is shown in the following figures. Figure 5 shows the performance of top-\textit{k} queries (\( k = 1000 \)), with the data size is 2 G, 4 G, 8 G, and 16 G in uniform distribution. We can see that the performance of top-\textit{k} queries in the classification results of \textit{E2E} model is better than \textit{ELM\_CMR}. The reason is that \textit{E2E} model can effectively evaluate the optimal configuration parameters of the queries than \textit{ELM\_CMR} model. When \( k \) is much smaller than the original data, the global shared information of ComMapReduce can reach the most optimal one quickly, so the Mappers can retrieve the shared information in the initial phase to filter the unpromising data.

Figure 6 shows the performance of \textit{kNN} queries with different data size of uniform distribution. The performance of \textit{E2E} is also optimal to \textit{ELM\_CMR} model, where the reason is that \textit{E2E} model can gain the suitable classification results.

Figure 7 shows the performance of skyline queries in anticorrelated distribution with different data size. We can see that the performance of different execution plans is not obviously different, but PostOS is a little better. The reason is that the original data are skewed to the final results in anticorrelated distribution. The percentage of filtering is low, so the performance difference is not obvious. In this situation, although \textit{E2E} and \textit{ELM\_CMR} can obtain the classification, it can also choose the other communication strategies.
<table>
<thead>
<tr>
<th>Query</th>
<th>top-(k)</th>
<th>(k)NN</th>
<th>skyline</th>
<th>join</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data (G)</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>E2E</td>
<td>HCS-0.5</td>
<td>HCS-0.5</td>
<td>PreOS</td>
<td>PreOS</td>
</tr>
<tr>
<td>ELM,CMR</td>
<td>ECS</td>
<td>ECS</td>
<td>HCS-1</td>
<td>HCS-0.5</td>
</tr>
</tbody>
</table>
Figure 8 shows the performance of join queries in different data size of small-big tables, with the same data size of the small table 2 G, and the different data sizes of big table are shown in Table 3. The performance of E2E is much better than ELM_CMR. In ComMapReduce, the join attributes of the small table can be set as the shared information to filter the unpromising intermediate results.

Third, we evaluate the performance of the same kind of multiple queries implemented in different execution orders. Figure 9 illustrates the performance of two execution orders of four skyline queries in different data sizes, 0.2 G, 0.4 G, 0.8 G, and 1.6 G in anticorrelated distribution. The running time of our optimized execution order is shorter than the running time of the original order. Our E2E can identify the proper classifications and execution order of the queries to enhance the performance. In the original order in random, the queries do not have the optimal classifications and implementations with ELM_CMR model.

Figure 10 shows the performance of the multiple queries about different types under different execution orders, respectively, top-\(k\) (2 G), top-\(k\) (8 G), \(k\)NN (2 G), and \(k\)NN (8 G) in uniform distribution. We can see that the performance under our E2E model is much optimal than the ELM_CMR model. The optimized execution order of E2E model does not only consider the classification results of E2E model, but also consider the characteristics of the queries deeply.

6. Conclusions

In this paper, we propose an efficient query processing optimization model based on ELM, E2E model. Our E2E can effectively analyze the query processing applications and then generates the most optimized executions of query processing applications. After analyzing the problems of the former ELM_CMR model, we use two stages model to classify the query processing applications in ComMapReduce.
framework. Then, we propose an efficient training approach to train our model. We also give the query executions based on E2E model in two situations, Just-in-Time execution and Queue execution. The experiments demonstrate that the E2E model is efficient and the query processing applications can reach an optimal performance.

**Conflict of Interests**

The authors declare that there is no conflict of interests regarding the publication of this paper.

**Acknowledgments**

This research is supported by National Natural Science Foundation of China (nos. 60873068, 61472169, 61100022, and 61472069) and Talent Projects of the Educational Department of Liaoning Province (no. LR2017).

**References**


Submit your manuscripts at http://www.hindawi.com