Journal of Computer and Communications, 2020, 8, 1-19
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

An Improvement of Data Cleaning Method for Grain Big Data Processing Using Task Merging

Feiyu Lian[1,2*], Maixia Fu[2], Xingang Ju[1]

[1]Key Laboratory of Grain Information Processing and Control (Henan University of Technology), Ministry of Education, Zhengzhou, China
[2]College of Information Science and Technology, Henan University of Technology, Zhengzhou, China
Email: *lfywork@163.com, fumaixia@126.com, xigangju@163.com

How to cite this paper: Lian, F.Y., Fu, M.X. and Ju, X.G. (2020) An Improvement of Data Cleaning Method for Grain Big Data Processing Using Task Merging. *Journal of Computer and Communications*, **8**, 1-19.
https://doi.org/10.4236/jcc.2020.83001

Received: January 31, 2020
Accepted: March 2, 2020
Published: March 5, 2020

Abstract

Data quality has exerted important influence over the application of grain big data, so data cleaning is a necessary and important work. In MapReduce frame, parallel technique is often used to execute data cleaning in high scalability mode, but due to the lack of effective design, there are amounts of computing redundancy in the process of data cleaning, which results in lower performance. In this research, we found that some tasks often are carried out multiple times on same input files, or require same operation results in the process of data cleaning. For this problem, we proposed a new optimization technique that is based on task merge. By merging simple or redundancy computations on same input files, the number of the loop computation in MapReduce can be reduced greatly. The experiment shows, by this means, the overall system runtime is significantly reduced, which proves that the process of data cleaning is optimized. In this paper, we optimized several modules of data cleaning such as entity identification, inconsistent data restoration, and missing value filling. Experimental results show that the proposed method in this paper can increase efficiency for grain big data cleaning.

Keywords

Grain Big Data, Data Cleaning, Task Merging, Hadoop, MapReduce

1. Introduction

In recent years, with the applications of Internet of Things in agricultural field, grain big data have been occurring gradually. In the field of big data processing, MapReduce is a famous programming frame [1], which has already been applied

in Google, Facebook, Tencent, Alibaba and other large internet companies. However, its current status of only keeping a watchful eye on data analysis, not data itself, may probably lead to grave consequence. In the food industry of China, old or incomplete data in grain situation database reach 13.6% - 30% [2]. Data quality problem may probably make research or analysis meaningless, even lead to disastrous consequence. For example, due to the decision-making mistakes originated from the grain data error, macro-control of grain market failed in 2008 in China [3]. Therefore, for big data analysis and mining, data quality plays a key role, and data cleaning is a powerful tool to guarantee data quality.

At present, data cleaning for big data, including data consistency [4] [5] [6], entity identification [7] [8] [9] [10], data preprocessing [11] [12] and so on, has been researched by many scholars [13] [14] [15] [16]. However, few researchers have studied optimization of data cleaning on MapReduce frame [17]. Based on the conception of the error distribution law and position accuracy of the GPS data, Yang *et al.* proposed a data cleaning method for this kind of spatial big data using movement consistency. GPS data are cleaned based on the similarities of GPS points and the movement consistency model of the sub-trajectory [4]. Tang tries to provide an overview of recent work in different aspects of data cleaning: error detection methods, data repairing algorithms, and a generalized data cleaning system [13]. Yan *et al.* proposed an iterative data cleaning method based on time sequence analysis because the power device status information can be made equivalent to the multivariate time sequence of each state [18]. Gueta *et al.* applied User-level data cleaning to biodiversity databases, and presented a new framework to quantify the effect of data cleaning on SDMs [19]. Xu *et al.* proposed an incorrect data detection method based on an improved local outlier factor (LOF), and used a simulation of vibration data generated by a defective rolling element bearing to verify the effectiveness of the proposed method [20]. All the above scholars studied big data cleaning methods in specific fields, but there is still a lack of general methods for big data cleaning. Nowadays, almost all of data analysis tasks can be finished using MapReduce, but at the same time, a large number of redundancies, with the explosion of data, are also generated. In China, due to the lack of top-level design and related standards, the grain informatization process initiated by the government has produced a large number of dirty data, which has brought negative impact on macro decision-making. It is necessary to generate a better method to specifically clean grain big data. In this paper, for grain big data, we proposed a new optimized method based on task merge to reduce these redundancies.

A traditional big data cleaning system is shown in **Figure 1**. It runs on

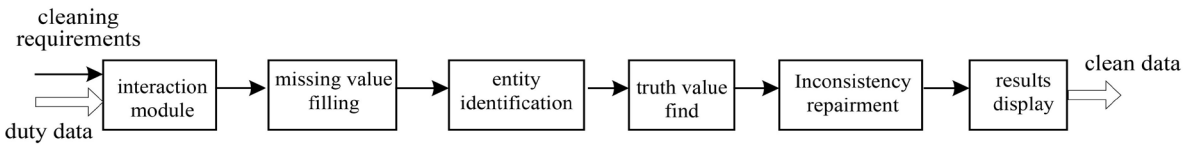

Figure 1. The structure chart of the function modules for traditional big data cleaning system.

Hadoop platform, and deals with different kinds of data with different levels of quality in a flexible way. This system consists of several modules, and each module deals with a type of data quality. The interaction module provides an input interface for files or data that need to be cleaned. The display module gives a comparison between dirty data and cleaned data. Entity identification and true value discovery module are used to reduce redundancy. Inconsistency detection module is used to recover data, and data filling module tests missing data and finishes data filling. Users may select required modules to deal with data quality problem that they encountered.

The above data cleaning system often requires multiple MapReduce operations. For a complicated problem, we need to divide it into many simple tasks, and each task is executed via one round of MapReduce operation. In the majority of cases, this division is excessive, which leads to superfluous MapReduce operations. In this paper, we merged some tasks to optimize the MapReduce operations for the above big data cleaning system. To be specific, the optimized technique proposed in this paper merged redundancy computations or simple computations from the same files to reduce cycle number of MapReduce. Thus, system running time reduced greatly, and system performance has also been improved obviously.

2. Optimization for Missing Data Filling

2.1. Traditional Missing Data Filling

The automatic data collection produces amount of missing data. Normally, Naive Bayes Classifier (NBC) is applied in resolving missing data. In the traditional data cleaning system, the missing data filling module roughly contains three parts: parameter estimation module, linking module and filling module [21]. The main task of parameter estimation module is to compute the probability of each attribute value, and takes the value with the highest probability as filling value. Specially, when the sample space is large enough, the probability is replaced approximatively by the frequency that the attribute value appears in. The linking module associates attribute value with its probability. The input data of linking module are the output of parameter estimation module as original data to be filled, and the output is a file that contains the relationship between missing value and its probability. Filling module is executed in the manner of a cycle of MapReduce. First, a linking computation is executed between the output of linking module and the original input data via an offset as key value. Its Map stage is similar to linking module, and its Reduce stage uses bayes formula to select a maximum probability value as a filling value. In this paper, we only considered discrete missing data filling.

2.2. Analysis and Optimization for the Missing Data Filling Module

We first have analyzed the data stream and relationship between modules in the above data filling framework. The whole filling process needs two kinds of con-

ditional probabilities (CP): 1) Parameter estimation module uses tuples that don't contain missing data in its input data to compute a conditional probability (entitled CP1) of an attribute value that needs to be filled; 2) Another conditional probability (entitled CP2), which is one group of special values, is used in the process of filling value. The relationship between the two CPx is determined by the values of dependency attributes. The group of special values is associated with the tuples that need to be filled in original data via the offsets of the tuples. Therefore, the linking module is required between parameter estimation module and filling module.

After observing the data stream in above system carefully, we found that both map input and map output contain the offset of the tuples in parameter estimation, but the output of the Reduce only contains attribute value and CP2, which leads to it is necessary to add a linking computing to link CP2 with the offset of the tuples to be filled.

Aiming at the above circumstance, we proposed an optimized scheme that merged the tasks in parameter estimation module and linking module. Firstly, in parameter estimation module, we associated the exporting conditional probability with the offset of the tuples that contain missing data. Its algorithm is as follow **Algorithm 1**.

In the above code segment, value is the attribute value of each tuple that contains missing value, and offset is the skewing quantity of original record to original file, probable_value.txt is the text file that contains all possible value for missing data. The Reduce process is shown in **Figure 2**.

For example, **Table 1** is the dataset that contains missing data, and the missing

Algorithm 1. Parameter estimation algorithm.

```
Input: data file containing missing value and possible missing value
Output: conditional probability
Map (Object, DataText, DataText, DataText)
Input: key :=offset, value:=tuple
    FOR each (key, value)    DO
        IF missing_value exists in tuple THEN
            FOR each value in probable_value.txt DO
                FOR each property in tuple DO
                    output_key := probable_value
                    output_value := "#" + offset + property_ID + property
Reduce (DataText, DataText, DataText, DataText)
    FOR each value in value_table DO
        IF "#" exists in value THEN
            APPEND property_ID + property on likelihood
        else
            compute conditional probability of each property_ID + property
        IF property_ID + property exists in likelihood THEN
            output_key := offset
            output_value := "#" + list of probable value + conditional probability
```

data probably may be V1 or V2. The former two tuples don't contain missing data, so we can split them only according to their attributes. The third tuple contains missing value, so we should split it according to each probable value. The outputs of Map stage are shown in Table 2. In Reduce stage, firstly, the prefixes of all input data are checked. Then, some calculations on contingent

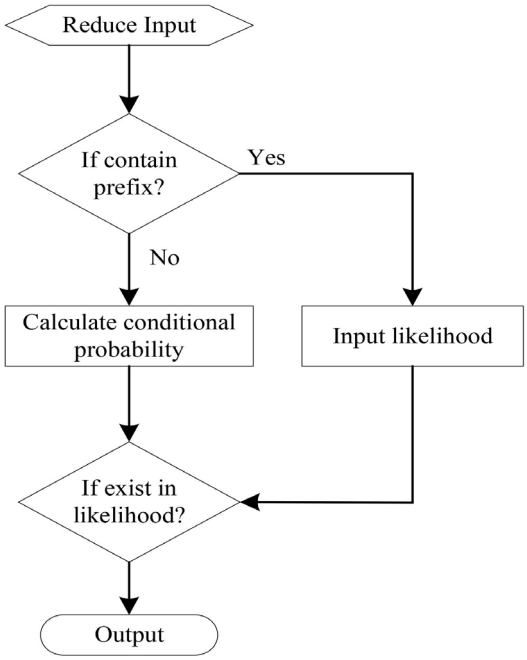

Figure 2. The reduce flow chart for parameter estimation.

Table 1. The data including missing-value.

offset	Field 1	Field 2	Field 3
0	V1	V2	V1
S1	V2	V1	V2
S2	?	V1	V1

Table 2. The map output for parameter evaluation.

Key	Value
tuple_value/probable_value	Prefix + offset + property ID + property
V1	0, 1, V2
V2	0, 2, V1
V2	S1, 1, V1
V2	S1, 2, V2
V1	#S2, 1, V1
V2	#S2, 1, V1
V1	#S2, 2, V1
V2	#S2, 2, V1

Table 3. The Reduce output for parameter evaluation.

Key	Value
117	#V2 0.5, #V1 0.5

probabilities are executed if missing values are contained in the input data, otherwise, the input data are entered into linklihood that is used to determining whether contingent probabilities should be output. And at last, we selected the contingent probabilities that attribute values are in likelihood to output results, and the output results are shown in Table 3.

In the parameter estimation module, the algorithm complexity on Map stage is $O(m+NX)$, where m is the number of tuples that don't contain missing values, N is the number of tuples that contain missing values, and X is the number of probable missing values. Normally, due to $N \le m$, $O(m+NX)=O((1+X)m)$ is true. Because X is smaller m greatly, the algorithm complexity is $O(m)$, and its complexity on Reduce stage is also $O(m)$. Therefore, the algorithm complexity of the parameter estimation module is $O(m)$.

Although the algorithm complexity of the parameter estimation module always is $O(m)$, both the MapReduce cycle number and IO number in the missing value filling module reduce, as shown in the above example, from 3 to 2, which indicates obvious optimization effect.

3. Optimization of Entity Identification

Entity identification is to recognize the form of an entity. For same entities, the data from different sources produce different presentations, even probably produce some errors in data storage or transformation. In MapReduce framework, although there are many researches on entity identification [22] [23], these researches basically solved the identification of anonymous entities, but few of them can solve problems on the identification of homonymous entities faultlessly [24] [25]. In this paper, we tried to solve the two problems simultaneously.

We found that both basic cluster module and entity identification module repeatedly use preprocessing result M times (M is the number of attributes that each tuple contains), and the subsequent entity identification module also works for single attribute. If we consider the preprocessing module and the entity identification module as a whole model, we need to scan input files many times, and can only use the part of input data, which results in the low data utilization rate. In addition, system requires extra resource for each task allocation. In view of this, we need a scheme that can process all of attributes for each tuple in one MapReduce cycle.

For this purpose, we proposed following optimization idea for entity identification module: firstly, use basic cluster module to process all the attributes synchronously, and then produce an attribute index table for all attribute values. Thus, we can merge these separated preprocessing procedures together. The detailed scheme is described as follows.

3.1. Basic Cluster Module

This module doesn't output the ith attribute value, but output all attribute values. Considering that the entity IDs in attribute index table are from different attributes, we added a prefix to the keys to translate "attribute value" into "attribute sequence * attribute value". Because MapReduce classifies attributes according to keys, the entities with same attribute values will be in same attribute table. On Reduce stage, we added attribute sequence to entities as the prefix of entity IDs. The following is the optimized basic cluster algorithm (**Algorithm 2**).

In the above algorithm, the entity ID is the code of each tuple. We set an unique entity ID for each tuple (data-in-line) in preprocessing stage. In addition, property ID is the sequence code of an property in tuple.

The following is an illustration of an optimized algorithm flow. **Table 4** shows the data that need to be recognized. In Map stage, we split all tuples according to their properties and then output results, as shown in **Table 5**. The entities with same property values will be input into the same Reduce, and output the property index table, as shown in (01 * 01, <02 * 01, 02 * 03>, <03 * 01, 03 * 02>, 01 * 02, 02 * 02, 01 * 03, 03 * 03).

For the above basic cluster module, the algorithm complexity in Map stage is $O\left(m\left(x+x^2\right)\right)$, where x is the number of the attributes. Because x normally is a very small constant, the algorithm complexity is around $O(m)$. In Reduce stage, there are no any changes except an additional attribute ID, so the algorithm complexity is still $O(m)$. To sum up, the algorithm complexity of the whole basic cluster module is $O(m)$.

Algorithm 2. Basic cluster algorithm.

```
Input: relationship table
Output: attribute index table
Map (LongWritable, DataText, DataText, DataText)
Input: key:=entity ID, value:=tuple
FOR each (key, value) DO
   FOR each property DO
      output_key := property ID + "*" + property value
      output_value := entity ID
Reduce (DataText, DataText, NullWritable, DataText)
FOR each value in value_table
   entity ID:= property ID + "*" + entity ID
   output_key:= null
   output_value:= all of entity ID in value_table
```

Table 4. Entity identification data.

Entity ID	Wheat variety	Year	Production place
01	Yunong 416	2017	Henan province
02	Zhongmai 895	2016	Henan province
03	Ganmai 16	2017	Hebei province

Table 5. Output of basic cluster map.

property ID + "*" + property value	entity ID
01*Yunong416	01
02*2017	01
03*Henan province	01
01*Zhongmai895	02
02*2016	02
03*Henan province	02
01*ganmai16	03
02*2017	03
03*Hebei province	03

3.2. Entity Identification Module

Because the above improvement on the entity cluster module guarantees that the entities with same attribute values belong to a same attribute index table, the algorithm of entity identification module doesn't need to be changed. Thus, the time complexity of its algorithm is still.

Because we only performed optimum operation for partial data on Hadoop, we regarded entity partition module as constant, and the time complexity didn't be changed. Before optimization, the cycle times of MapReduce is 1 + M (1 + 4) = 5M + 1, and after optimization, it is change to be 1 + 1 + 4 = 6, so speed-up rate reaches (5M + 1)/6. Normally, due to M > 1, speed-up rate is bigger than 1, and with the increase of M, the speed-up effect is more evident, and at the same time, the times of IO also reduced from 5M + 1 to 6, which reduced system uptime for IO. In addition, due to the reduction of cycles on MapReduce, the time for task scheduling and the used resource are also reduced.

In general, theoretically, the scheme proposed in this paper can provide obvious optimization result.

4. Reparation Optimization to Inconsistent Data

In the real application or database, there are amounts of inconsistent data due to various reasons. In the proposed system, we defined an integrity constraint according to conditional function dependency principle in theory of data dependency, and used the integrity constraint to repair inconsistent data. The purpose of this paper is to improve the performance of the reparation module to inconsistence data. As for how to guarantee that the repair process is correct, it is depended on conditional functions. The detailed explanations can be seen in reference [5] [6].

4.1. Optimization for Inconsistent Data Reparation

The main steps of the inconsistent data reparation module are as follows [26]: (1) The data files and the CFDs (centralized file directory system) files are input

to the system to be executed for preprocessing, then are transformed to a proper format, and are checked for subsequent processing; (2) check and repair the outputs from preprocessing and get primary reparation results; (3) check primary reparation results, determine whether introduce new inconsistent. If produce new inconsistent, execute step (1) again, otherwise continue to execute step (3). To avoid trapping into endless loop, an upper limit for these steps is necessary; (4) Process the reparation results, and change data format to original format to be used by another system normally.

4.2. Analysis and Optimization for Inconsistent Data Reparation

A main shortcoming of the inconsistent data reparation module is to divide a task that can be finished by only one cycle of MapReduce into several tasks, which results in the reduction of system performance. Therefore, we merged multiple tasks into one task for the optimization under the condition of not changing algorithm complexity.

1) Preprocessing Module

The function of the preprocessing module is only to setup index for input data, and doesn't involve data decomposition and merge. We may use a map function to realize this process, and the algorithm is shown as follow Algorithm 3.

Obviously, the algorithm complexity of this module is $O(m)$.

2) Detection and Reparation Module

Detection and reparation of constant violation can be finished only by one cycle of MapReduce. Because the Map stage dispenses one tuple N copies (N is the number of constant violations), and although N value is not big and has almost no influence to algorithm complexity of the reduce stage, it still can enlarge middle data quantity N times, which causes large load to communication. We found that we can repair these constant violations when calculating their suggested values, so it is not necessary to separate seeking process and reparation process. Therefore, we proposed an optima scheme that finished the process of the constant violation detection and reparation using one map function.

After finishing constant violation reparation via one map function, the data are guided directly to variable reparation module. The formats of both the input files are same, especially, if there are no constant violations in original data, both the input files are same entirely. Based on this viewpoint, we proposed an optimized scheme that merges the constant violation reparation and the variable

Algorithm 3. Preprocessing algorithm.

```
Input: dirty data files
Output: preprocessing results
Map (Object, Data_Text, NullWritable, Data_Text)
Input: key = offset, value = tuple
  FOR each (key, value) DO
    output_key := null
    output_value := key +value
```

violation reparation together. In this scheme, we arranged the constant violation reparation on the front of the first MapReduce process, and make its output results to be used to variable violation directly in Map function. The algorithm is as follow **Algorithm 4**.

Where offset is the index of the tuples, and fixFlag is the reparation flag that indicates whether or not we need the reparation, and "0" indicates we need the reparation due to a violation, "1" indicates we needn't the reparation. cfdseq is a cfd serial number of a tuple with violation, ptseq is a serial number of the tuple in mode table, and propseq is an attribute serial number of the inconsistent data of the tuples, and the output_value is the reparation of the attribute.

The computation complexity of **Algorithm 4** is $O(m)$, and the flow chart of the algorithm is shown in **Figure 3**.

In terms of time complexity, the algorithm did not change the computational complexity of each module and each MapReduce within each module before and after optimization. In terms of MapReduce rounds and IO times, the MapReduce rounds of the system changed from 1 + 1 + 2 + 1 + 1 + 1 = 7 before optimization to 1 + 2 + 1 + 1 = 5 after optimization. From the perspective of MapReduce rounds alone, the acceleration ratio of the system is 7/5 = 1.4. In addition, the optimization of the system also makes the MapReduce of the preprocessing module become a map, which will correspondingly reduce the running time of the system. With the reduction of MapReduce rounds, the IO times of the system are correspondingly reduced, which also reduces the IO burden of the system.

5. Experimental Results

The computer cluster that used to do experiments was composed of ten nodes, including one task-tracker (name node) and nine job-trackers (data node). The

Algorithm 4. The Map algorithm in the first cycle of MapReduce for inconsistent data detection and reparation.

```
Input: the preprocessing results
Output: the output of Map in the first cycle of MapReduce for variable violation
Input: key=offset, value=tuple
Map (Object, Data_Text, Data_Text, Data_Text, Data-Text)
    FOR each cfd r = (R:X→Y, tp) DO
      IF tuple[X] = tp[X] and tuple[Y] ≠tp[Y] THEN
          tuple[Y] = tp[Y]
    FOR each cfd r = (R:X→Y,Tp) DO
      IF fixFlag THEN
        output_key := (cfdseq, ptseq, propseq,1)
        output_value := (offset, tuple)
    FOR tuples not match cfd with varialbles
    DO
        output_key :=   (cfdseq, ptseq, propseq, 0)
        output_value := (offset, tuple)
```

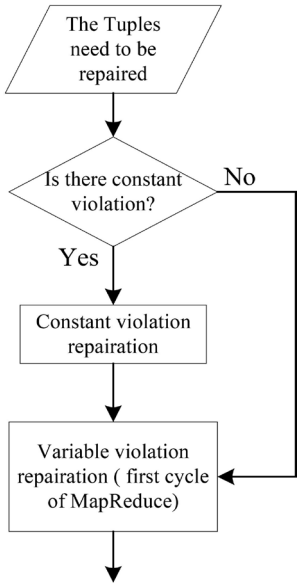

Figure 3. The flow of **Algorithm 4**.

hardware configuration of each node was as follows: Intel i7 7700k processor, 4.2 GHz main frequency, 8 G memory, and 1TB hard disk. The whole system was developed using java on Eclipse environment, and run on Hadoop 3.0 platform based on Centos 7.5.

5.1. Experiment for Entity Identification Optimization

Considering the optimization effectiveness for dataset scale, we used the real dataset that is from National Warehouse Grain Condition Monitoring Project (WGCM). It is a grain storage information depository in China. We selected three attributes, including warehouse temperature, grain temperature, and warehouse humidity, and five data scales, including 11.7 M, 45.3 M, 70.2 M, 115.6 M and 150.9 M as our experimental environment. The attributes of data are allocated weight values as 0.9, 0.1 and 0.1. The experimental results are shown in **Figure 4**.

Figure 4 shows the time consuming on different scale dataset using Naïve, BlockSplit, PairRange that used in Ref. [22], and the proposed method in this paper based on tasks merge.

Following the increase of dataset scale, both the unoptimized system and the optimized system increase their run-time, but the run-time ratio of the unoptimized system to optimized system is about 2.3 due to the only three attributes that were used in this experiment for each data. Based on the analysis to optimization effect in section 2.2, the theoretical ration value is $(5 * 3 + 1)/6 = 2.7$, which is in accordance with experimental results. Because the entity identification based on BlockSplit and PairRange is more complicated than the method based on tasks merge, their run-times are longer than that of the method that was proposed in this paper. In summary, this experiment illustrated the good expandability of the optimization scheme.

1) Influence of Parallelization Level to Optimization Effectiveness

Considering the influence of Reduce number in cluster to optimization results, we used the real WGCM dataset as experimental data. We selected three attributes of the experimental data that are titled warehouse temperature, grain temperature, and warehouse humidity to construct an experimental dataset with 100,000 records. We set weight values for each attribute 0.9, 0.1 and 0.1 respectively, and set Reduce number 2, 4, 6, 8 and 10 respectively. As shown in Figure 5, the optimization effectiveness of the proposed method is obvious under the different parallelization levels.

From Figure 5, we can see that, following the increase of parallelization degree, system uptime increases. The main reason for this phenomenon is that the data size for experiment is too small to provide benefits. However, system still reaches 2.3 speed-up ratio under different parallelization modes, which indicates the optimization result is in line with forecast.

2) Influence of Parallelization Level to Optimization Effectiveness

In this experiment, we studied the influence of feature number of the input

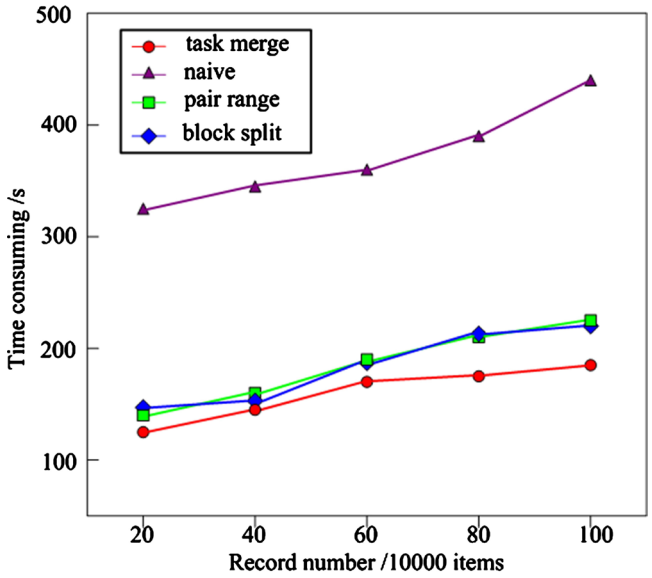

Figure 4. The influence of dataset scale to optimization results.

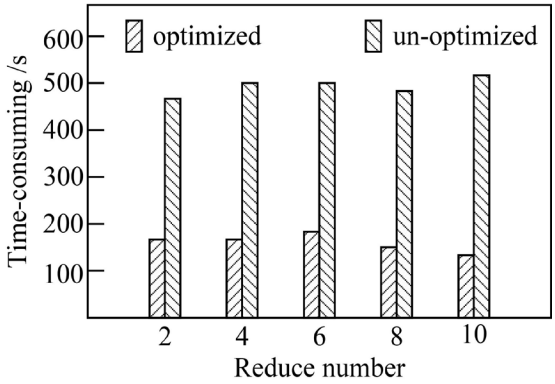

Figure 5. The influence of Reduce number to optimization effectiveness.

tuples on optimization effectiveness. The experimental results are shown in Figure 6. When dealing with the same scale records, the optimization effectiveness got better and better with the increase of the feature number. From Figure 6, we can see that, the optimization results are the worst, even are lower than the un-optimized when processing only one record, but the optimization results became better after increasing feature number. The reason that caused the above experimental results was that the optimized scheme generated more middle data than non-optimized scheme, and was more complicated than the non-optimized scheme. Because the purpose of the optimized scheme proposed in this paper was to utilize the input data adequately when dealing with multiple attributes, the optimized scheme presented more advantages than non-optimized scheme with the increase of the attributes.

5.2. Optimization Experiments for Inconsistent Data Reparation

We used a real dataset that is from Zhengzhou grain trade market in China, which is named ZZGR, and an artificial dataset that generated from Transaction Processing Performance Council (TPC-H) to verify the operative mode of the system in real environment. We did an experiment for speed-up ratio verification on the real dataset, and at the same time, we also did an experiment for expansibility and parallelism verification on the artificial dataset.

1) Speed-up Ratio Experiment

In this experiment, six tuples without missing values were selected from the ZZGR dataset and were put into some errors intentionally to violate several restraints. The experiment conditions and results are shown in Table 6. The experimental results show that there is prominent speed-up effectiveness on the real dataset. The optimization scheme provided actual 1.3 speed-up ratio and less than 1.4 speed-up ratio in theory (see Algorithm 4).

2) Expansibility Experiment

The aim of this experiment was to verify the same effectiveness on different scale dataset. The experiment dataset is composed of six attributes from

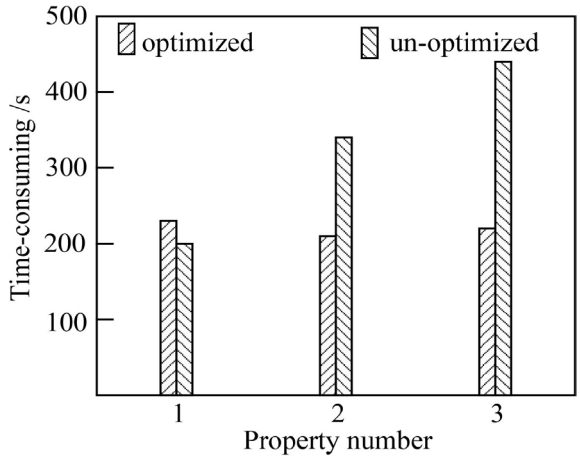

Figure 6. The influence of attribute number to optimization effectiveness.

Table 6. Speed-up ratio experiment on the ZZGR dataset.

Items	Data
Data source	ZZGR
Record number	50000
cfd number	3
tp number	6
Reduce number	2
Run time before optimization	212
Run time after optimization	166
Speed-up ratio	1.28

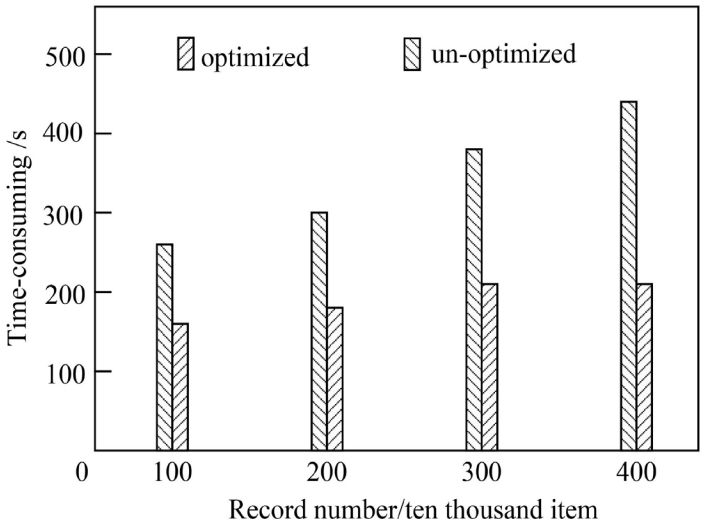

Figure 7. Expansibility experiment.

lineitem.tbl that was generated by TPC-H, and CFDs are composed of one cfd including three lps. The experimental results that are shown in Figure 7 indicate that the optimization effectiveness gets better with the increase of dataset scale. Thus it can be seen that our optimization scheme is extensible easily.

From Figure 7, we can see that the uptime of the non-optimized system increase with the enlargement of the dataset, and the uptime of the optimized system also increase but the slope is lower than the former. Compared to the former, the speed-up ratio of the optimized system improved from 1.6 to 2.2. All modules are in overloaded works before optimization, but the optimized system reduces the burdens of various modules except the modules of data inconsistency test and reparation. In addition, we can also see that the non-optimized system firstly enters full load status with the enlargement of the dataset compared to the optimized system, as shown in Figure 7.

3) Parallelism Experiment

To verify the influence of parallelism to optimization effectiveness, this experiment used six attributes from lineitem.tbl that was generated from TPC-H to

form dataset. CFDs were composed of one cfd including three tp samples, as shown in Figure 8.

From Figure 8, we can see that the system speed-up ratio reached up to 2.3 under the low degree of parallelism with 2 reduces, then with the increase of degree of parallelism, the system speed-up ratio reduces. In addition, for the non-optimized system, the uptime become shorter with the increase of the degree of parallelism, but the uptime of optimized system remains unchanged. The reason why the above phenomenon generates is that the non-optimized system has the weak processing capacity. We may only add the degree of parallelism to improve processing capacity, which means the system uptime becomes shorter with the increase of the degree of parallelism. But for the optimized system, due to the big handling capacity, it always is in the underloading status when processing the same data size, which means the advantages are not obvious when increasing the degree of parallelism.

5.3. Optimization Experiment for Missing Value Filling

In this experiment, the used data are from the real dataset entitled ZZGR and the artificial dataset that generated from TPC-H. To verify the operation status in real environment for the optimization system, we used the two datasets to verify the impact of missing rate on optimization result on the ZZGR dataset, and to verify expansibility and parallelism on the artificial dataset.

1) Impact of Miss Rate on Optimization Effectiveness

We have also studied the impacts of various miss rates on optimization results. The data for the experiments were generated by emptying some data in used dataset based on certain proportion. In this experiment, we selected eight discrete features and missing features with six values. The experimental results are shown in Figure 9. From Figure 9, we can figure out that the speed-up ratio stabilizes at 1.5 roughly, which matches theoretical value of 3/2 in this module, under the miss rates that are shown in Figure 9.

2) Verification Test for Expansibility

In this experiment, we selected six features from lineitem.tbl that is generated

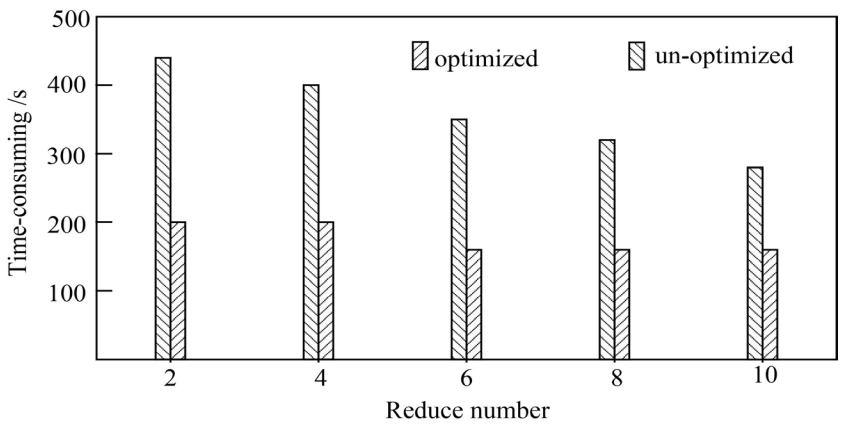

Figure 8. Parallelism experiment.

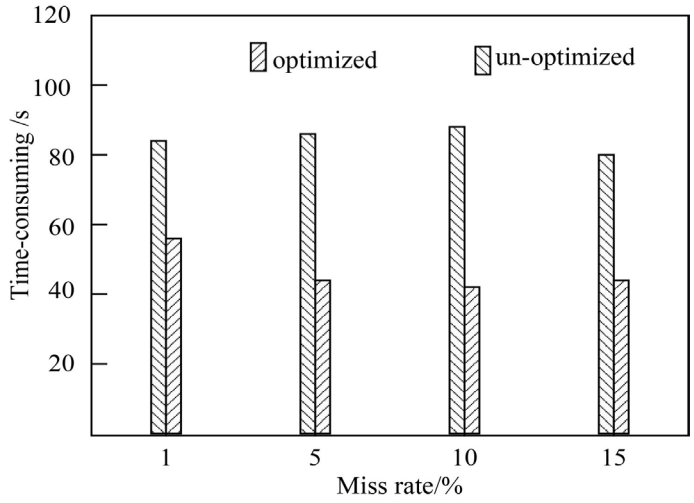

Figure 9. The impact of miss rate on optimization results.

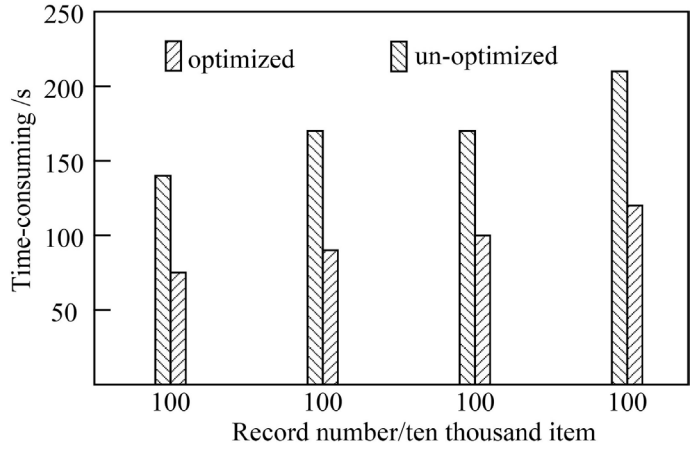

Figure 10. Verification test for expansibility.

from TPC-H to generated dataset. The experimental results are shown in Figure 10. From Figure 10, we can see that the system uptime increases with the enlargement of the dataset, but the speed-up ratio stays around 1.5, which coincides with the theoretical value of this module.

3) Verification Test for Parallelism

This experiment was designed to test the optimization effectiveness of the system under the different parallelism degrees. In this experiment, we used TPC-H to produce a data table entitled lineitem.tbl, including six attributes and 1,000,000 tuples. We randomly emptied 5% of the data in first column of the table and recorded the optimization results under different parallelism degrees. The experimental results are shown in Figure 11.

On the provided dataset, the operating efficiency of the non-optimized system and the optimized system are not to become better with the increase of parallelism degree yet. This is because for the given scale of dataset, the most appropriate number of Reduce is determinate, and only increasing parallelism degree will bring about more spending on task allocation for the system. In any case,

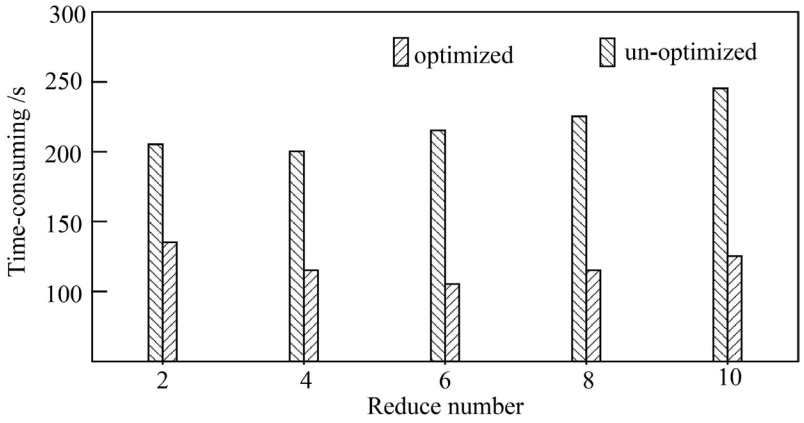

Figure 11. The impact of different numbers of Reduce on optimization results.

the optimization effectiveness is obvious under the different parallelism degrees.

6. Conclusion

Although we have acquired many achievements on study of Hadoop, most of the softwares based on MapReduce programming frame are inefficient due to the lack of the deep understanding of MapReduce. Therefore, we proposed a set of optimization methods for MapReduce programming frame in this paper, and the test for these methods passed as expected on massive data cleaning system. We only changed a little to the original system using these methods, and hardly changed its original complexity. The optimization objects are simply obtained by only reducing the cycle numbers of MapReduce and IO numbers, which indicates its simplification and practicability. In the future works, we will apply these methods to more systems based on MapReduce 3, and other MapReduce alternatives, like Spark, and do a more deep analysis to find the disadvantages of these methods to improve their performances.

Acknowledgements

This work was supported by Key scientific research projects of institutions of higher learning (18A51003), Henan, China.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Chen, C.H., Lin, J.W. and Kuo, S.Y. (2015) MapReduce Scheduling for Deadline-Constrained Jobs in Heterogeneous Cloud Computing Systems. *IEEE Transactions on Cloud Computing*, **6**, 127-140. https://doi.org/10.1109/TCC.2015.2474403

[2] Wang, J.G. (2007) Ideas about Improving Foodstuff Statistic. *Journal of Zhejiang Business Technology Institute*, **1**, 21-23. (In Chinese)

[3] Cheng, G.Q. and Zhu, M.D. (2013) The Situation and Policy Framework of Chinese

Grain Macro-Control. *Reform*, **1**, 18-34. (In Chinese)

[4] Yang, X., Tang, L., Zang, X. and Li, Q. (2018) A Data Cleaning Method for Big Trace Data Using Movement Consistency. *Sensors*, **3**, 824-825. https://doi.org/10.3390/s18030824

[5] Fan, W.F., Li, J.Z., Ma, S. and Tang, N. (2011) Interaction between Record Matching and Data Repairing. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Athens, 12-16 June 2011, 469-480. https://doi.org/10.1145/1989323.1989373

[6] Fan, W.F., Geerts, F., Tang, N. and Yu, W.Y. (2013) Inferring Data Currency and Consistency for Conflict Resolution. *Proceedings of the IEEE* 29*th International Conference on Data Engineering*, Brisbane, April 2013, 470-481. https://doi.org/10.1109/ICDE.2013.6544848

[7] Mittal, D., Pilli, E.S. and Gupta, M. (2017) Efficient Entity Resolution Using Multiple Blocking Keys for Bibliographic Dataset. *Proceedings of the* 2017 *International Conference on Intelligent Communication and Computational Techniques*, Jaipur, 22-23 December 2017, 108-113. https://doi.org/10.1109/INTELCCT.2017.8324029

[8] Mestre, D.G., Pires, C.E.S. and Nascimento, D.C. (2017) An Efficient Spark-Based Adaptive Windowing for Entity Matching. *Journal of Systems & Software*, **128**, 1-10. https://doi.org/10.1016/j.jss.2017.03.003

[9] Wang, D.X., Liu, X., Luo, H.Z. and Fan, J.P. (2015) A Novel Framework for Semantic Entity Identification and Relationship Integration in Large Scale Text Data. *Future Generation Computer Systems*, **64**, 198-210. https://doi.org/10.1016/j.future.2015.08.003

[10] Ran, H., Wang, H.Z., Zhu, R., Li, J.Z. and Gao, H. (2013) Map-Reduce Based Entity Identification in Big Data. *Journal of Computer Research & Development*, **50**, 170-179. (In Chinese)

[11] He, Q., Tan, Q., Ma, X., *et al.* (2010) The High-Activity Parallel Implementation of Data Preprocessing Based on MapReduce. *Proceedings of the International Conference on Rough Sets and Knowledge Technology*, Beijing, 15-17 October 2010, 646-654. https://doi.org/10.1007/978-3-642-16248-0_88

[12] Rodrigues, M., Santos, M.Y. and Bernardino, J. (2019) Big Data Processing Tools: An Experimental Performance Evaluation. *Wiley Interdisciplinary Reviews*: *Data Mining and Knowledge Discovery*, **9**, e1297. https://doi.org/10.1002/widm.1297

[13] Nan, T. (2014) Big Data Cleaning. In: Chen, L., Jia, Y., Sellis, T. and Liu, G., Eds., *Web Technologies and Applications*, Lecture Notes in Computer Science, Volume 8709, Springer International Publishing, Berlin, 13-24. https://doi.org/10.1007/978-3-319-11116-2_2

[14] Uno, T., Maegawa, T., Nakahara, H. and Hamuro, Y. (2017) Micro-Clustering by Data Polishing. *Proceedings of the* 2017 *IEEE International Conference on Big Data*, Boston, 11-14 December 2017, 4546-4554. https://doi.org/10.1109/BigData.2017.8258024

[15] Kiyoshi, M. and Shigeki, H. (2004) ICT Workers and Professional Spirit in the E-Business Era. *Journal of Electronic Science and Technology of China*, **3**, 108-113. (In Chinese)

[16] Xue, C.J. (2019) Guest Editorial Special Issue on Emerging Technologies for Big Data Processing. *Journal of Electronic Science and Technology*, **1**, 3-4. (In Chinese)

[17] Zhang, F., Xue, H.F., Xu, D.S., *et al.* (2013) Big Data Cleaning Algorithms in Cloud Computing. *International Journal of Online Engineering*, **3**, 77-81. https://doi.org/10.3991/ijoe.v9i3.2765

[18] Yan, Y., Sheng, G., Chen, Y., *et al.* (2015) Cleaning Method for Big Data of Power Transmission and Transformation Equipment State Based on Time Sequence Analysis. *Automation of Electric Power Systems*, **7**, 138-144.

[19] Gueta, T. and Carmel, Y. (2016) Quantifying the Value of User-Level Data Cleaning for Big Data: A Case Study Using Mammal Distribution Models. *Ecological Informatics*, **34**, 139-145. https://doi.org/10.1016/j.ecoinf.2016.06.001

[20] Xu, X., Lei, Y. and Li, Z. (2019) An Incorrect Data Detection Method for Big Data Cleaning of Machinery Condition Monitoring. *IEEE Transactions on Industrial Electronics*, **67**, 2326-2336. https://doi.org/10.1109/TIE.2019.2903774

[21] Sun, J., Li, J., Gao, H. and Wang, H. (2018) Truth Discovery on Inconsistent Relational Data. *Tsinghua Science & Technology*, **3**, 288-302. (In Chinese) https://doi.org/10.26599/TST.2018.9010004

[22] Jin, C., Chen, J. and Liu, H. (2017) MapReduce-Based Entity Matching with Multiple Blocking Functions. *Frontiers of Computer Science*, **11**, 895-911. https://doi.org/10.1007/s11704-016-5346-4

[23] Altowim, Y. and Mehrotra, S. (2017) Parallel Progressive Approach to Entity Resolution Using MapReduce. *Proceedings of the* 2017 *IEEE* 33*rd International Conference on Data Engineering*, San Diego, 19-22 April 2017, 909-920. https://doi.org/10.1109/ICDE.2017.139

[24] Xie, H., Lu, X., Tang, Z., *et al.* (2016) Detection of Entity Mixture in Knowledge Bases Using Hierarchical Clustering. *Proceedings of the National CCF Conference on Natural Language Processing and Chinese Computing*, Kunming, 2-6 December 2016, 288-299. https://doi.org/10.1007/978-3-319-50496-4_24

[25] Pouriyeh, S. (2017) Es-Ida: Entity Summarization Using Knowledge-Based Topic Modeling. *Proceedings of the Eighth International Joint Conference on Natural Language Processing*, Taipei, November 2017, 605-614.

[26] Zhang, A.Z., Men, X.Y., Wang, H.Z., Li, J.Z. and Gao, H. (2015) Hadoop-Based Inconsistence Detection and Reparation Algorithms for Big Data. *Journal of Frontiers of Computer Science & Technology*, **9**, 1044-1055. (In Chinese)