Research on Software Architecture Optimization of Cloud Computing Data Center Based on Hadoop

Jun Yu\textsuperscript{a}, Guangxin Zhu\textsuperscript{b}, Haiyang Chen\textsuperscript{c}, Lin Wang\textsuperscript{d} and Mingjie Xu\textsuperscript{e}

State Grid Electric Power Research Institute, Nanjing, 210000, China
\textsuperscript{a}yujun@sgepri.sgcc.com.cn, \textsuperscript{b}zhuguangxin@sgepri.sgcc.com.cn, \textsuperscript{c}chenhaiyang@sgepri.sgcc.com.cn, \textsuperscript{d}wanglin18@sgepri.sgcc.com.cn, \textsuperscript{e}xumingjie@sgepri.sgcc.com.cn

Abstract. This paper introduces the architecture optimization design of high performance cloud computing data center (HCDC), and proposes an efficient parallel algorithm based on Hadoop system and MapReduce framework. The performance of the algorithm in big data analysis and processing, and the impact of the algorithm on energy efficiency are studied. Finally, the challenges of future work in this area of research are discussed.

1. Introduction
The data scale has been expanding from tuberculosis level to lead level with the rapid development of cloud computing, Internet of Things and social networks has greatly enriched the production channels of massive heterogeneous data. To enable the scalability of micro-module data center applications, the development and application of cloud computing and cloud services technologies will provide end-users with on-demand services through standard models. The service has the advantages of saving investment, improving business support capability, improving operation and maintenance efficiency, reducing investment risk and decision risk, and green energy conservation. For the processing of massive data in the micro-module data center, the cost of center point selection is higher. Through the improvement of intelligent algorithms such as ant colony algorithm [1] and genetic algorithm [2], these improved algorithms can obtain good convergence. However, algorithms for initial value improvement can achieve better initial values, but iterative center updates use mean replacement, which is susceptible to singular value points. Based on the above problems, this paper proposes an efficient parallel algorithm for HCDC based on MapReduce. After experimental research, the algorithm can effectively improve the data processing performance and contribute to the improvement of data center energy efficiency.

2. Related work
Hadoop is an open source implementation by Doug Cuitting based the MapReduce system of Google, including distributed file system named HDFS and the MapReduce which is a parallel framework for programming. Hadoop can run on large clusters composed of general commercial machines, and can run distributed applications to achieve efficient processing of large-scale data.

MapReduce [3] is a programming model based on Hadoop distributed platform. It is suitable for distributed computing of massive data sets. It uploads files to HDFS distributed file system and processes files in distributed file system through Map and Reduce functions. It provides an organic
combination for the processing of big data. According to the required information, the Map function splits the files in the distributed file system, and generates a series of intermediate key-value pairs <key, value>. The same key is merged and sorted in the shuffle stage. The Reduce processes the merged sorted data and traverses the value [4] of the same key value. Then it outputs the processed information by bar.

In addition, MapReduce has good scalability and fault tolerance to handle large-scale data processing. The underlying HDFS file system backup and recovery mechanism, machine heartbeat mechanism and task supervision mechanism ensure high reliability of program operation.

The workflow of MapReduce is shown in Fig.1. The Job Tracker is responsible for job scheduling, the TaskTracker is responsible for task scheduling, the JobTracker is assigned to Task Tracker, the TaskTracker is used to execute the job, and the job is processed by Map Task and Reduce Task to complete this job.

![Figure 1. Working flow of MapReduce model.](image)

The data warehouse has the characteristics of topic-oriented, integration, consistency, time-varying and non-volatility, the Hadoop distributed file system is very suitable for the underlying storage of data warehouse applications. Hive, an important sub-project of the Hadoop platform, is based on the data warehouse architecture of the Hadoop file system. It uses MapReduce programming technology to implement some SQL statements and provides a SQL-like programming interface. The main reason why it is called the data warehouse infrastructure is not only that it provides the conversion and parsing of SQL to MapReduce programs for data analysts, but more importantly, Hive provides the most important metadata service for the Hadoop distributed file system to be used in data warehousing.

3. Method Design

Selects Sorting Algorithm [5-8] is a heap sorting algorithm that selects the first elements to establish a binary tree. When a new element is added, it compares with the parent node of the binary tree. If it is larger than the replacement, it re-adjusts the binary tree. Until all elements are processed.

The commonly used text sampling method is progressive scan sampling, and the required sample is selected by traversing the global data, and the sampling method can retain the original data format; This approach is simple and easy when you take a small sample of data, and you can get data samples quickly. As the sampling amount increases, the method increases the system consumption, and the running time also increases linearly, which is not suitable for big data sampling operations.

For the defects of progressive scan sampling, based on the MapReduce framework, this paper uses selection sorting algorithm for parallel random sampling.

The specific implementation process is as follows:

Input: sample size K, random number range X (0 < X < K), Reduce number is R.

Output: K data samples.

Step 1: In the Map phase, each data is randomly generated with an integer from 0 to X as its key value, and the data content is used as the value to form a <key, value> key-value pair output.
Step 2: Combines the randomly generated same key value data to form <key, list<value1, value2, valueN>>, and internally sort according to the key value.

Step 3: Each Reduce outputs the previous K / Rn data.

The algorithm is described as follows:

The core code is as follows:

Map stage:
Random rd = new Random();
int tmp = rd.nextInt(X);
Context. Write (new IntWritable (key), new Text (value.toString));
For (Text val: value) {
    i = 0;
    If (i < K / Rn) {
        Context. Write (null, new Text (val.toString));
        i++;
    }
}

In view of the memory overflow of the original algorithm in the face of massive data, we use the MapReduce computational model to improve the parallel algorithm to better adapt to the processing of massive data of HCDC. The choice of the initial value of the mean algorithm is that randomness leads to unstable clustering results and accuracy.

4. Experimental and Results

The experiment consists of 6 PCs, one of which acts as the master node for resource scheduling and allocation, and the remaining 5 as slave nodes, responsible for the task. The machines use the same configuration: 14-core CPU, 4G memory, 500G hard drive, CPU clocked at 2.9HZ, model Pentium(R) Dual-Core E6600, operating system Ubuntu 14.04LTS, JDK 1.7.0, The cluster is built using the Hadoop 2.2.0 version.

1. Comparison of convergence

As can be seen from the Fig.2, the experiment uses 6 machines as the cluster structure, and the master acts as the master node to allocate data and does not participate in the operation. The remaining five machines act as slave nodes, they receive data assigned by the Master, and perform calculations. In this pseudo-distributed environment, the P parallel algorithm in the literature and the number of iterations of the algorithm in the same-scale data set clustering are compared. The experimental data is 4D data with 10000 data objects, divided into 3 categories, and the data set size is 0.48M.
Table 1. Comparison of convergence.

| Number | PK-Means algorithm | Algorithm of this paper |
|--------|--------------------|------------------------|
| 1      | 8                  | 6                      |
| 2      | 10                 | 5                      |
| 3      | 6                  | 5                      |
| 4      | 7                  | 6                      |
| 5      | 10                 | 7                      |
| 6      | 6                  | 5                      |

It can be seen from Table 1 that under the pseudo-distribution, the algorithm has fewer average iterations and better convergence. This is because the PKmeans algorithm solves the problem that the parallel algorithm deals with insufficient memory of massive data, but the initial value selection is random and the clustering result is unstable. The algorithm of this paper has better convergence by selecting better initial values.

(2) Test of the cluster accuracy

In the pseudo-distributed environment, the experiments are compared with Canopy-Kmeans, BCKmeans algorithm and the proposed algorithm in the literature respectively, and each algorithm runs 6 times and counts the average clustering accuracy. Due to the small amount of data, so it processes global data without sampling. The data and results of experiments are shown in Table 2 and Table 3.

Table 2. Standard data set.

| Type of data | Data collection size/KB | Dimension | Number of categories |
|--------------|-------------------------|-----------|---------------------|
| Iris         | 4.58                    | 4         | 4                   |
| 4k-far       | 6.56                    | 2         | 3                   |
| wine         | 10.5                    | 13        | 3                   |

Table 3. The comparison of Clustering Accuracy.

| Type of data | Canopy-Kmeans | BCKmeans | Algorithm of this paper |
|--------------|---------------|----------|------------------------|
| Iris         | 84.46         | 87.62    | 88.45                  |
| 4k-far       | 90.37         | 93.26    | 94.86                  |
| wine         | 65.46         | 67.03    | 68.49                  |

It can be seen from Table 3 that the Canopy-Kmeans algorithm has relatively low accuracy under the test of three sets of standard data sets, and the algorithm has better accuracy than the former two algorithms. Although the Canopy-Kmeans algorithm improves the initial value, the selection of the initial value is easy to fall into the local optimum, affecting the clustering effect; although the BCKmeans algorithm improves on this problem, it avoids local optimality and makes it have better clustering accuracy. However, the center point iterative update uses the method of averaging, so the center point is easily affected by the singular value and affects the final clustering result; In this paper, the algorithm not only optimizes the initial value, but also uses the weight replacement strategy to update the center point in order to reduce the influence of the singular value, so the algorithm has better clustering precision.

(1) Data collection test

The experimental content is two-dimensional data with a data set size of 1.8G. The parallel environment architecture consists of six Hadoop clusters, one as the master and the other five as the slave. Under the same data set, the efficiency of the two text sampling methods is tested, that is the sampling time variation of the test progressive scan sampling and the K selective sort parallel sampling with the change of the sampling amount. The test result is shown in Fig.3.
Figure 3. Comparison of data sampling

Figure 4. Node efficiency

Figure 5. Comparison of speedup

It can be seen from Fig.3 that when the sampling amount is small, the progressive sampling efficiency is the highest, but as the sampling amount increases, the operating efficiency decreases, and the parallel sampling time tends to be stable. Therefore, the time of parallel sampling in this paper tends to be stable. So, the parallel sampling in this paper is more suitable for sampling work in big data environment.

(2) Cluster performance test

The experiment used Hadoop clusters built by 6 PCs, one of which was used as the Name Nodes, and the other 5 were used as Data Nodes to participate in the calculation one by one. The calculation efficiency and the acceleration ratio of the different scales of the data were compared. The experimental data is shown in Table 4.

| Data set size | Number of rows | Data dimension | Number of categories |
|---------------|----------------|----------------|---------------------|
| 0.62          | 25071024       | 3              | 4                   |
| 1.2           | 52649151       | 3              | 4                   |
| 1.8           | 78973726       | 3              | 4                   |

As the number of nodes increases, the operation time of different big data sets is compared. The comparison is shown in Fig 4.

It can be seen from Fig.4, in the face of the processing of these three sets of large data sets, with the increase of the number of nodes, the operating efficiency is improved, and the convergence time is also significantly reduced, indicating that the algorithm is suitable for clustering operations under big data.
The speedup ratio is the ratio of the time consumed by the stand-alone system and the parallel system when processing the same task. It is used to measure the scalability and parallelization effect in parallel computing.

As the number of nodes increases, the comparison of the acceleration ratios of different large data sets is compared. The comparison results are shown in Fig. 5.

It can be seen from Fig. 5, as each number of computing nodes increases, the speedup ratio increases continuously. When the dataset is large, increasing the number of computing nodes can improve the parallel execution process. Therefore, the implementation of this algorithm under Hadoop distributed computing platform can effectively improve the efficiency of the algorithm.

5. Conclusion
In this paper, we study an efficient parallel algorithm for the micro-module data center. The experimental comparison of the algorithm has better clustering accuracy. In the cluster performance test, the algorithm is applied to big data by adjusting the number of cluster nodes and calculating the acceleration ratio. The analysis and processing, by adjusting the number of mappings and the way of cluster memory, further improve the performance of big data processing, based on the parallel algorithm of cloud platform technology, at the same time, it also contributes to the energy efficiency of cloud data centers.

Acknowledgments
This work was supported by the National Key Research and Development Program of China under Grant No.2017YFB1010000.

References
[1] YU Qianqian, DAI Yueming, LI Jingjing. Parallel clustering algorithm ACO-K-means based on MapReduce [J]. Computer Engineering and Application, 2013, 49 (16): 117 - 120.
[2] JIA Ruiyu, GUAN Yuyong, LI Yalong. Parallel genetic K-means clustering algorithm based on the MapReduce [J]. Computer engineering and design based on MapReduce, 2014 (2): 657 - 660.
[3] Chen k, Zheng WM. Cloud computing: System instances and current research [J], Journal of Software, 2009, 2009: 1337 - 1348.
[4] Gates A F, Natkovich O, Chopra S, et al. Building a high-level dataflow system on top of MapReduce: the Pig experience [J], Proceedings of the VLDB Endowment, 2009, 2 (2): 1414-1425.
[5] Thusoo A, Shao Z, Anthony S, et al. Data warehousing and analytics infrastructure at facebook [A], Proceedings of the 2010 ACM SIGMOD International Conference on Management of data [C], 2010, 1013 - 1020.
[6] Bialecki A, Cafarella M, Cutting D, et al. Hadoop: a framework for running applications on large clusters built of commodity hardware [J], Wiki at http://lucene.apache.org/hadoop, 2005, 11.
[7] Thusoo A, Sarma J S, Jain N, et al. Hive: a warehousing solution over a map-reduce framework [J], Proceedings of the VLDB Endowment, 2009, 2 (2): 1626 - 1629.
[8] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters [A], In the Proceedings of the 6th Symposium on Operating System Design and Implementation [C], 2004, 137 - 150.