Proceedings of the 2023 International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/31/20230149

Elastic distributed dataset RDD (Resilient Distributed Datasets)

Yiwen Li

Taiyuan University of Technology, Shanxin, Taiyuan

1326518139@qq.com

Abstract. In recent years, the amount of data to be processed has become larger and larger, and it is difficult for a single processor to handle data, so we need to work hard to develop the expansion mode of cluster mode. Mapreduce, Storm, etc. are a solution for clusters to process data. However, we still have parallel processing, user resource sharing and other issues to be addressed. RDD data model is the key to solve these problems. In this paper, RDD is applied to Spark for detailed expansion, and project experiments are carried out to study the RDD data model. The programming language used in this article is python. The advantage of studying distributed iterative computing is that during the calculation process, the task calculation fails and can be recovered. The default is to allow four failures recoverable. If the stage fails, one can recover, and if the partition fails, one can also recover. Multiple partitions can improve parallelism and efficiency. After a shuffle, a partition failed because the many-to-one relationship needs to be recalculated.

Keywords: RDD, Spark, data base, bigdata, distributed iterative computing.

1. Introduction

RDD is the core concept of Spark and an elastic distributed dataset. Elastic distribution refers to that RDDs are horizontally multi partitioned. In the vertical direction, when the memory is insufficient during the calculation process, RDDs can be written to disk and other external storage, and flexible data exchange can be made with external storage. RDD is a virtual dataset that does not contain real data. It uses a fault tolerance mechanism to reconstruct the data model. In terms of spatial structure, it can be understood as a set of read-only and partitioned distributed data sets, which contain multiple partitions. Partitioning is to put data records with the same attributes together according to specific rules. Each partition is equivalent to a dataset fragment. RDD is a read-only dataset with attributes. Attributes are used to describe the state of the current dataset. The dataset is composed of data partitions and mapped (by blocks) to real data. RDD attributes include name, partition type, parent RDD pointer, data localization, data dependency, etc. It represents an immutable, partitioned set of elements that can be calculated in parallel. RDD has the characteristics of data flow model: automatic fault tolerance, location aware scheduling and scalability. RDD allows users to explicitly cache data in memory when executing multiple queries, and subsequent queries can reuse the data, which greatly improves the query speed [1].

The arrival of the big data era has made the original data mining methods and BI (Business Intelligence) tools unable to meet practical needs, and the computer science community urgently needs to seek new data mining solutions. In March 2012, the White House of the United States released its big data plan, which will take five years to establish an Extensible Data Management Analysis and Research Center (SDAV). On the basis of integrating the scientific research forces of six national laboratories and seven universities in the United States, a new data mining system will be developed to use supercomputers to manage and visualize big data. The traditional data mining technology has been difficult to meet such needs, and the most prominent contradictions are reflected in the scalability, efficiency, and processing ability of heterogeneous data of the system. Currently, the research on the analysis and processing of big data in a distributed environment is very necessary and has broad prospects for development [2].

In addition, the author would like to explain that RDD includes the following:

Dataset: a data set used to store data

Distributed: the data in RDD is stored in a distributed way and can be used for Distributed computing

Resilient: Data in RDD can be stored in memory or disk

All operations are based on the RDD data structure

RDD can be considered as a distributed list or array array, an abstract data structure, and an abstract class Abstract Class and generic Generic TypeYarn. Internal characteristics of RDD data structure:

1. Partition

RDD is logically partitioned, and the data of each partition is abstract. During calculation, the data of each partition is obtained through the compute function.

2. Read only

RDDs are read-only. To change the data in RDDs, one can only create new RDDs based on existing RDDs.

3. Dependence

RDDS maintain a certain correlation, also known as dependency.

4. Cache

If one uses the same RDD multiple times in an application, one can cache the RDD. The RDD will only get the partitioned data according to the correlation when it is first calculated.

5.checkpoint

As the iteration progresses, there will be more and more relationships between RDDs. Once errors occur in the subsequent iterations, reconstruction is required, which will inevitably affect performance and efficiency. To this end, RDD supports checkpoint to save data to persistent storage, so that previous relationships can be cut off.

2. Main analysis of RDD

The main idea of Spark to realize iterative computing is RDD, which stores all the calculated data in distributed memory. Iterative calculation is usually performed on the same dataset repeatedly. The data in memory will greatly improve the IO operation. This is also the core of Spark: memory computing.

RDD operations include two types, namely Transformation and action operation. It includes the following operations:

Filter (func) filters out the elements that satisfy the function func and returns a new dataset.

Map (func) passes each element to the function func and returns the result as a new dataset.

FlatMap (func) is familiar with map(), but each input element can be mapped to 0 or multiple output results.

When groupByKey() is applied to the dataset of (K, V) key value pairs, a new dataset in the form of (K, Iterable) is returned.

When reduceByKey (func) is applied to the dataset of (K, V) key value pairs, a new dataset in the form of (K, K) is returned, where each value is the result after each key is passed to the function func for aggregation.

Action operations are the real trigger for computing. Only when the Spark program executes the action operation, it will perform the real calculation, load data from the file, complete the conversion operation again and again, and finally, complete the action operation to get the results.

Operation meaning:

Count () returns the number of elements in the dataset.

Collect () returns all elements in the dataset in the form of an array.

First() returns the first element in the dataset.

Take (n) returns the first n elements in the dataset in the form of an array.

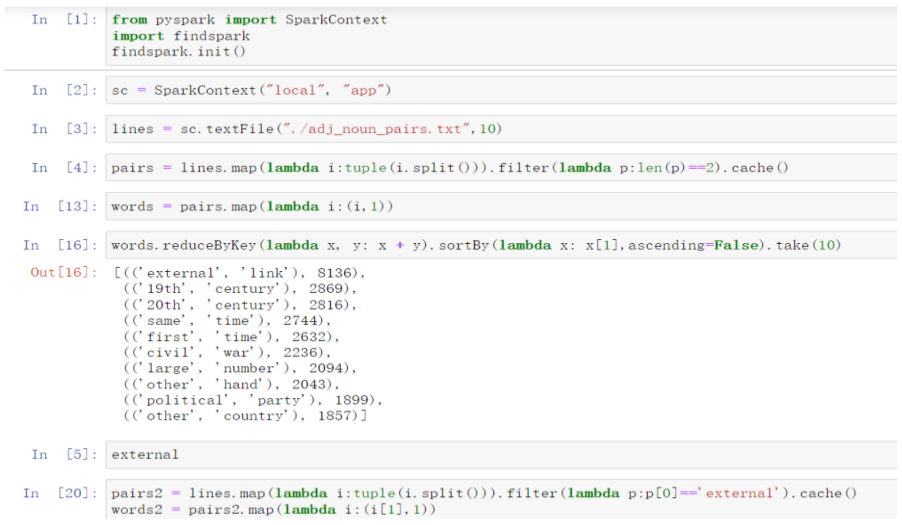
Reduce (func) aggregates the elements in the dataset through func (input two parameters and return a value)

Foreach (func) transfers each element in the dataset to the function func to run.

The next content is the design of iterative calculation using RDD.

In particular, Apache Spark has emerged as a widely used open-source engine. Spark is a fault-tolerant and general-purpose cluster computing system providing APIs in Java, Scala, Python, and R, along with an optimized engine that supports general execution graphs. Moreover, Spark is efficient at iterative computations and is thus well-suited for the development of large-scale machine learning applications [3].

Design:the code overview is as follows:



```
In  [1]: from pyspark import SparkContext
         import findspark
         findspark.init()

In  [2]: sc = SparkContext("local", "app")

In  [3]: lines = sc.textFile("./adj_noun_pairs.txt",10)

In  [4]: pairs = lines.map(lambda i:tuple(i.split())).filter(lambda p:len(p)==2).cache()

In [13]: words = pairs.map(lambda i:(i,1))

In [16]: words.reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1],ascending=False).take(10)
Out[16]: [(('external', 'link'), 8136),
          (('19th', 'century'), 2869),
          (('20th', 'century'), 2816),
          (('same', 'time'), 2744),
          (('first', 'time'), 2632),
          (('civil', 'war'), 2236),
          (('large', 'number'), 2094),
          (('other', 'hand'), 2043),
          (('political', 'party'), 1899),
          (('other', 'country'), 1857)]

In  [5]: external

In [20]: pairs2 = lines.map(lambda i:tuple(i.split())).filter(lambda p:p[0]=='external').cache()
         words2 = pairs2.map(lambda i:(i[1],1))
```

Figure 1. Code overview.

The design specification is as follows:

•Download the adj-noun pair data.

• wget http://www.cse.ust.hk/msbd5003/data/adj_noun_pairs.txt.

• Load it into spark.

• lines = sc.textFile("adj_noun_pairs.txt", 10).

• pairs = lines.map(lambda l: tuple(l.split())).filter(lambda p: len(p)==2).cache().

• 1) Find the most frequent adj-noun pair.

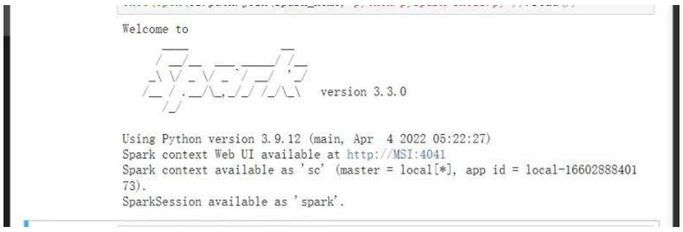
• 2) For this adj in (1), find the 3 most common noun’ s paired with it.

The project mainly solves two problems: one is to find frequent adj none and the other is to find an adjective and the three nouns that match it the most times. In the map function, the text content is divided by the space symbol and is recorded. The reduce function performs the final merge statistics on the incoming < K2, V2 > to form the final statistical result.

```
    from pyspark import SparkContext
import findspark
findspark.init()
sc = SparkContext("local", "app")
```

By default, the Spark interactive environment has created a variable named sc for SparkConhtext. Creating a new environment variable will not work. However, in a standalone spark application, one needs to create a SparkContext object yourself. So one needs to create the object first.

After initialization, one can use various methods contained in the SparkContext object to create or operate distributed data sets and shared variables.



```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.3.0
      /_/

Using Python version 3.9.12 (main, Apr  4 2022 05:22:27)
Spark context Web UI available at http://MSI:4041
Spark context available as 'sc' (master = local[*], app id = local-16602888401
73).
SparkSession available as 'spark'.
```

Figure 2. Install.

Next,using the textfile method of sparkcontext to create RDDS of text files. This method accepts the URI of the file (local path on the computer, or URI), and reads it as a collection by line.

Next, the author will briefly describe the RDD partition. The amount of RDD data is large, and it will be divided into many partitions, which are stored on different nodes.

The primary function of partitioning RDDs is to increase parallelism.

Start multiple threads on these different work nodes to process the data of these multiple partitions in parallel, so as to increase the parallelism of tasks. In addition, it can make communication overhead. The requirement for partitions is to make the number of partitions equal to the number of CPU cores in the cluster as much as possible. For different Spark deployment modes, one can configure the default number of partitions by setting the value of the parameter Spark.default.parallelism. More importantly, Spark provides its own HashPartition and RangePartition. Next, traverse each tuple, split the text by line according to the space, and filter the words with length of 2.

Line is org.apache.spark.rdd. An instance in the RDD class. For RDD, each conversion operation will generate different RDDs for the next operation. The conversion process of RDD is lazy evaluation. Generally speaking, the whole conversion process only tracks records, and no real calculation occurs. When an action operation is encountered, the real calculation of the whole process will be triggered, and lines will be executed Map() operation, the input parameter of map(). Lambda is essentially an anonymous function that can greatly optimize the code structure, and can implement the interface itself. Lambda line: line. split () is a lambda expression. The meaning of map (lambda line: line. split ()) is to take out each element in the RDD lines in turn. For the currently obtained element, assign it to the line in the lambda expression, and then execute the line. split () of the lambda expression.

The original data of Program:

```
lines = sc.textFile("adj_noun_pairs.txt",10)
# [word1, word2 .....]
pairs = lines.map(lambda i: tuple(i.split())).filter(lambda p: len(p) == 2).cache()
```

```
# temp1 = lines.map(lambda i: tuple(i.split()))temp1 = [(word1,word2,
word3,word4),(word_11,word_12)....
# temp
```

RDD operations include two types, namely, transformation operations and action operations

(1) Conversion operation

Operation meaning:

Filter (func) filters out the elements that meet the func function and returns a new dataset.

Map (func) passes each element to the function func and returns the result as a new data set.

FlatMap (func) is familiar with map(), but each input element can be mapped to 0 or more output results.

When groupByKey() is applied to the dataset of (K, V) key-value pairs, a new dataset in the form of (K, Iterable) is returned.

When reduceByKey (func) is applied to the dataset of (K, V) key-value pairs, a new dataset in the form of (K, K) is returned, where each value is the result of passing each key to the function func for aggregation.

Action operation is the place that really triggers calculation. The Spark program will only perform the real calculation when it executes the action operation, load data from the file, complete the conversion operation again and again, and finally complete the action operation to get the result.

Operation meaning

Count() returns the number of elements in the dataset.

Collect() returns all elements in the data set as an array.

First() returns the first element in the dataset.

Take (n) returns the first n elements of the dataset in the form of an array.

Reduce (func) aggregates the elements in the dataset through the function func (enter two parameters and return a value).

Foreach (func) passes each element in the dataset to the function func to run.

Most popular distributed computing frameworks due to its memory-based computing methods. To improve ease of use, Spark provides users with about 200 configurable parameters that control the operation of the task. However, Spark has a huge parameter space, and users often cannot reasonably choose which parameters to configure [4].

The original data of program:

```
    words = pairs.map(lambda i: (i, 1))
# [((word1, word2), 1), ((word_21, word_22), 1) .....]
print(words)
words.reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], ascending=False).take(10)
# [1-10] [(words, x+y)]
```

First, the first line of code outputs a three-dimensional tuple, still uses the lambda function, and sets the value of the latter to 1, rdd.reduceByKey (lambda x, y: x+y) Consolidated statistics, sortBy (lambda x: x [0]), and arranges the columns in descending order through ascending=false in sortby(). If you want to arrange in ascending order, change the parameter to ascending=True to represent ascending order. Take (10) means to list ten groups of data. The reduceByKey can aggregate the data in pairs with the same Key. The aggregation method needs to be specified. Both reduceByKey and groupByKey have shuffle operations, but reduceByKey can combine data sets with the same key in the partition before shuffle, which will reduce the amount of data dropped. GroupByKey only groups, and there is no problem of data reduction. reduceByKey has high performance.

RDD is a data source that contains (word, 1) data in two partitions. Before grouping, pre-aggregate the data in the partition.

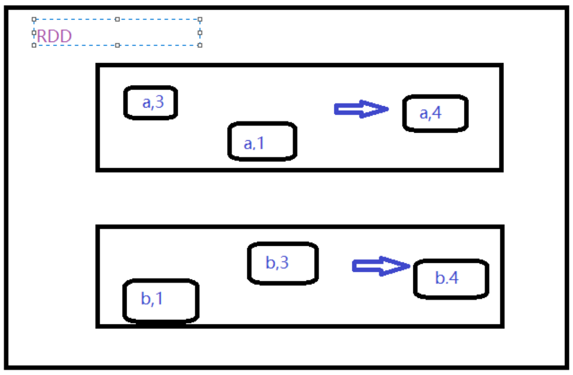


Figure 3. Principle.

Shuffle operation:

1. The data is pre-aggregated before grouping, and the amount of data participating in grouping becomes smaller, that is, the amount of data participating in Shuffle becomes smaller.

2. Because the amount of data participating in Shuffle is smaller, the number of disk IO during Shuffle will be smaller.

3. The number of quantity calculation becomes less during polymerization calculation.

GroupByKey can group the data of the data source into values according to the key.

RDD of the result of pairwise aggregation of Value according to the specified aggregation formula. The same key exists in one partition of the RDD, and the value can be aggregated. In the implementation process of groupbykey, because groupby key has no aggregation function, aggregation calculation is performed after all data is grouped.So there is prepolymerization.

The operation results are shown in the figure 9.

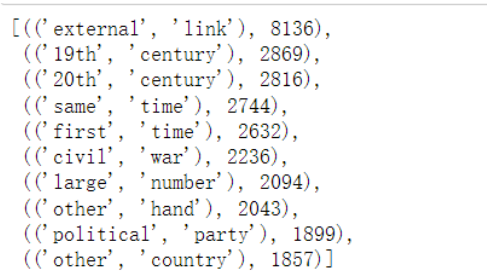

```
[(('external', 'link'), 8136),
 (('19th', 'century'), 2869),
 (('20th', 'century'), 2816),
 (('same', 'time'), 2744),
 (('first', 'time'), 2632),
 (('civil', 'war'), 2236),
 (('large', 'number'), 2094),
 (('other', 'hand'), 2043),
 (('political', 'party'), 1899),
 (('other', 'country'), 1857)]
```

Figure 4. Result.

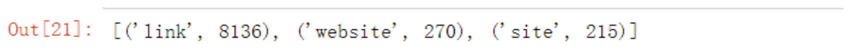
The original data of program:

```
    words.reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], ascending=False).take(10)
# [1-10] [(words, x+y)]
pairs2 = lines.map(lambda i: tuple(i.split())).filter(lambda p: p[0] == 'external').cache()
# pairs2 = [("external", word) ....]from pyspark import SparkContext
# import findspark
# findspark.init()

# sc = SparkContext("local", "app")
words2 = pairs2.map(lambda i: (i[1], 1))
```

```
# words2 = [(word_2, 1)]
words2.reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], ascending=False).take(3)
```

Similar to the above, the difference is that given an adjective, it starts from the second, that is, a noun.Consolidated statistics. The operation results are shown in the figure 10. The first map uses the lamda function to traverse the epoch group, and there is another filter() function. It is used to filter a sequence. Its function is to filter out elements that meet the conditions from a sequence. The filtering condition is two-dimensional tuple. The second dimension is “external”. The reducebykey groups are listed in descending order.



```
Out[21]:  [('link', 8136), ('website', 270), ('site', 215)]
```

Figure 5. Result2.

MapReduce is a software framework for parallel computing and running. It provides a huge but well-designed parallel computing software framework, which can automatically complete the parallel processing of computing tasks, automatically divide computing data and computing tasks, automatically allocate and execute tasks on cluster nodes and collect computing results, and hand over many complex details of the underlying system involved in parallel computing such as data distribution storage, data communication and fault tolerance processing to the system for processing, which greatly reduces the burden of software developers.

The mapreduce computing framework moves the code to the data side. It mainly deals with the problem through two stages: map stage and reduce stage. In the map phase, many map processes are parallel at the same time. Because the input of the map phase is determined by the inputformat object, one can use the default segmentation method or customize the segmentation method. In the map phase, the input is partitioned (the default is the hash method, and the purpose is: one partition corresponds to one reduce), sorted, merged, and so on.

In the reduce phase, many reduce processes are parallel at the same time. The input is the output of the map function. The user-defined reduce function determines the output of reduce, but the output format must be in the form of<key, value>. The specific calculation process of MapReduce is Input ----- Map ----- Shuffle ----- Reduce ----- Output.

After a certain proportion of Map tasks are completed, the Reduce node will copy the output data of these Map tasks, and the actual Reduce operation will not begin until all data has been copied. This process of generating data from the Map side to copying the data from the Reduce side is called Shuffle. The Shuffle process is the core of MapReduce, also known as the place where miracles occur, and focuses on the most critical parts of the MapReduce process. The Shuffle process contains many details and spans both sides of Map and Reduce. Understanding the Shuffle process is conducive to optimizing MapReduce job performance [5].

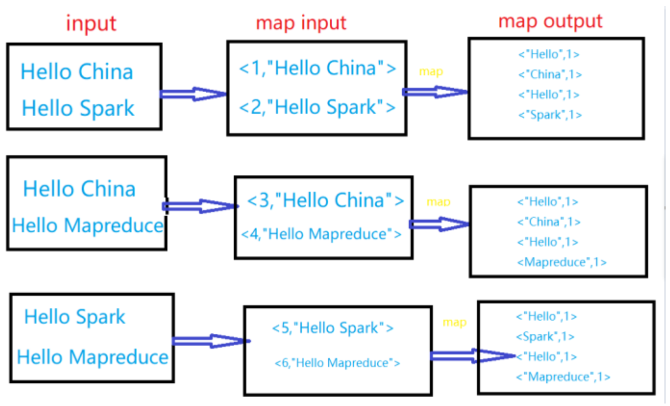


Figure 6. Principle2.

The specific steps are as follows.

Generally speaking, each data block corresponds to a map calculation. When the map side reads the data in the data block in HDFS, it will call the inputformat interface in Hadoop and use the key-value pair as the map input. After that, the user can operate on the data through the user-defined map function, and still output the data in the form of key-value pairs. The shuffle stage describes the process from the map function output to the reduce function input. Partition: MapReduce will first call the partition interface to partition the different key-value of the map output. The default is to hash the key value, which can also be achieved by overloading the partition interface. Sort: sort the key-value pairs of each partition according to the key value.

Merge: merge the key-value pairs of each partition to reduce the amount of data transfer between map and reduce. Merge: multiple files written to the disk will be merged after the map process is completed to form a large file. Merge is to divide the key-value of the same partition of different files into the same partition, and then sort and merge them to form the<key, value-list>. Read all the data on the map side into the cache and write it to the disk. After reading all the data, merge all the file data again. The task of the reduce stage is to execute the function for the input key-value according to the user-defined reduce function, and output the final result to the file system. The parallelism of the map phase of a job is determined by the client when submitting the job. The basic logic for the client to plan the parallelism of the map phase is: perform logical slicing of the data to be processed (divide the data to be processed into logical multiple splits according to a specific slice size), and then allocate a mapTask parallel instance for each split.

My running examples mainly represent wordcount. There are two kinds of running modes of MAPREDUCE program:

In local operation mode, the mapreduce program is submitted to LocalJobRunner to run locally as a single process.

The processed data and output results can be in the local file system or on hdfs. Write a program without a cluster configuration file (the essence is whether there is mapreduce. framework. name=local and yarn. resourcemanager. hostname parameter in the conf of the mr program).

In cluster operation mode, the mapreduce program is submitted to the yarn cluster resourcemanager and distributed to many nodes for concurrent execution. The processed data and output results should be located in the hdfs file system.

MapReduce provides three methods to improve usability for developers:

The first is to provide a stand-alone MapReduce library. After receiving the MapReduce task, this library will execute the map and reduce tasks locally. In this way, people can debug their MapReduce task locally by taking a little data, whether it is a debugger or a log.

Second, an HTTP server is embedded in the master, and then various statuses of the master are displayed to developers. In this way, one can see how many tasks have been completed, how many tasks are still in the process of execution, how many input data it has processed, how many intermediate data it has, how many output result data it has, and the percentage of task completion. Similarly, there is log information for each task. In addition, through the HTTP server, one can also see which worker's task failed and what the corresponding error log is.

Finally, MapReduce framework provides a counter mechanism. As a developer, people can define several counters themselves, and then call this counter in the Map and Reduce functions to auto-increment. All map and reduce counters will be summarized on the master node and displayed in the HTTP server above.

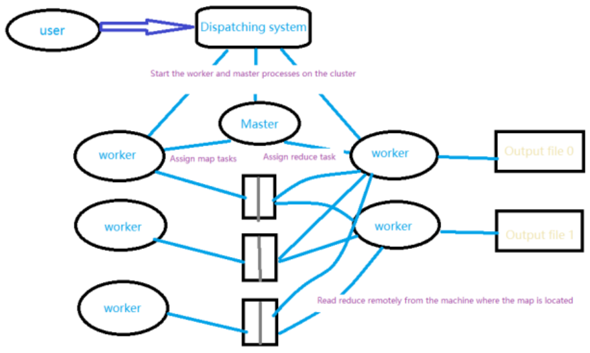


Figure 7. User and worker.

3. Conclusion

The main reasons for using mapreduce are summarized as follows: the processing of massive data on a single computer is not competent due to hardware resource constraints. Once the stand-alone program is extended to the cluster for distributed operation, it will greatly increase the complexity and development difficulty of the program

After the introduction of mapreduce framework, developers can focus most of their work on the development of business logic, and leave the complexity of distributed computing to the framework to handle.

It can be seen that Spark, as a supplement to Hadoop, has advantages in handling iterative work and interactive data analysis. The two begin to show a trend of convergence. From Hadoop 0.23 to make MapReduce a database, Hadoop's goal is to support more parallel computing models including MapReduce, such as MPI and Spark. It is difficult to predict who will be replaced with the development of technology in the future. We should learn from each other and complement each other. New requirements will lead to new platforms. The overall consolidation of a large amount of temporary result data generated by multiple Map tasks of the same job on a Map node replaces the original MapReduce architecture's mechanism of merging the result data of a single Map task, and solves the problems of the original multiple and large amount of result data on a Map node, as well as the time-consuming and high failure rate of copying these data on the Reduce side. This improvement scheme reduces the amount of output result data from the Map node to significantly reduce the amount of network transmission data for the entire cluster, and reduces the data transmission failure rate, reducing the execution time of MapReduce jobs to a certain extent, thereby improving the execution performance of MapReduce [6].

Large data has the characteristics of low value density, which means that valuable information can be purified quickly. Large data applications have penetrated into all aspects of life, such as pushing videos of interest to users on short video websites and recommending products of interest to users on e-commerce platforms. There is also the application of correlation algorithm in large data, such as retailers' placement of retail products and placing orders for things that customers may buy at the same time. In the measures to effectively solve the problems of big data application and operation, the big data analysis system based on cloud computing is the main strategy. Cloud computing application technology is an important part of big data analysis. It has high data processing efficiency for big data and enhances the sensitivity of system response [7].

The distribution of warehouse materials in logistics warehousing saves a lot of costs for enterprises. Large data can also carry out massive data mining and risk prediction. It also plays an important role in the insurance and financial industries.

During the implementation of parallel recommendation algorithms, it was found that there was an unreasonable problem of task scheduling in heterogeneous Spark cluster nodes. An adaptive task scheduling strategy for heterogeneous Spark clusters, HSATS, was proposed. Based on the optimization of neighborhood recommendation algorithms, it proposed to vectorize the hidden tag attributes of users or items, and ultimately fused them with similarity calculations [8].

Efficient data development one-stop solution tool, which can complete the development of real-time computing and offline computing tasks through IDE, connect all the above platforms and provide one-stop solutions. The data development ide provides all-round product services such as data integration, data development, data management, data quality and data services. The one-stop development management interface completes the operations of data transmission, conversion and integration through the data ide. Introduce data from different data stores, transform and develop them, and finally synchronize the processed data to other data systems. Through the efficient big data development IDE, big data engineers can basically shield all kinds of pain points and combine the above platform capabilities, so that big data development can be as simple as writing SQL.

References

[1] Meng X, Bradley J, Yavuz B, et al. MLlib: machine learning in apache spark[J]. Journal of Machine Learning Research, 2015, 17(1):1235-1241.
[2] Beijing University of Aeronautics and Astronautics,Design and Implementation of Distributed Microblog Data Processing System Based on Spark,2014
[3] 3.Armbrust M, Xin R S, Lian C, et al. Spark SQL: Relational Data Processing in Spark[C]// ACM SIGMOD International Conference on Management of Data. ACM, 2015:1383-1394.
[4] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets[C]// Usenix Conference on Hot Topics in Cloud Computing. USENIX Association, 2010:10-10.
[5] Zhu Peipei. Research on big data analysis and optimization technology based on cloud computing [J]. Modern information technology, 2019 (14): 69
[6] Guo ming,MapReduce source code analysis and performance improvement,2015
[7] Guo ming,MapReduce source code analysis and performance improvement,2015
[8] Beijing University of Aeronautics and Astronautics,Design and Implementation of Distributed Microblog Data Processing System Based on Spark,2014
[9] Yang Zhiwei Tutor: Zheng Zheng Master's Thesis, University of Science and Technology of China, May 6th, 2015