Optimization of Spark Data Skew in Big Data Environment

Huanshu Wang*
School of Computer & Communication Engineering,
Changsha University of Science & Technology
3043451759@qq.com

Huali Wang
School of Computer & Communication Engineering,
Changsha University of Science & Technology
2579704203@qq.com

ABSTRACT

In order to solve the problem of uneven data distribution in the shuffle stage of Spark distributed platform, this paper first analyzes the causes of data skew in Spark platform, and then establishes a skew model that can quantify the degree of data skew. Based on quantifiable skew model, a shuffle partition scheme is proposed to optimize data skew in Spark distributed platform. The working node of the partitioning scheme first samples the output data of the Map stage, summarizes the data of each working node to the master node to predict the data size of all working nodes, and then pre-partitions the intermediate data according to the load balancing partitioning algorithm and Hash partitioning algorithm to obtain a pre-partitioning table. The master node distributes the pre-partitioning table to each working node. Finally, the working node partitions all the intermediate data according to the pre-partitioning table. The experimental results under different skew conditions show that the shuffle partition scheme proposed in this paper is universal and efficient, and can effectively deal with the data skew problem of Spark distributed platform..

CCS CONCEPTS

• **Theory of computation**; • **Design and analysis of algorithms**; • **Distributed algorithms**;

KEYWORDS

Spark, data skew, shuffle, partitioning algorithm

ACM Reference Format:
Huanshu Wang* and Huali Wang. 2023. Optimization of Spark Data Skew in Big Data Environment. In *International Conference on Computer, Vision and Intelligent Technology (ICCVIT 2023), August 25–28, 2023, Chenzhou, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3627341.3630380

1 INTRODUCTION

Spark [1] is a popular memory computing based distributed computing platform, which is simple, universal and efficient. Because Spark has the memory computing feature, intermediate data is saved in the memory, reducing the disk disk landing times, reducing the disk I/O load, and improving the efficiency of data iterative calculation.. Data skew is a frequent problem in distributed computing. The Spark framework itself has not taken effective countermeasures against the data skew [1] problem, during the shuffle phase of Spark, data sets of tasks are unevenly distributed, and a large amount of data is pulled down for individual tasks. As a result, the execution efficiency of subsequent tasks is affected, the computing time is increased, and the efficiency of the whole Spark cluster is reduced. Therefore, the optimization of Spark 's shuffle mechanism is a very worthy topic to investigate.

2 ASSOCIATED STUDIES

MapReduce architecture has emerged for more than ten years, and academia and industry have had relatively in-depth research and mining of its various aspects, and have relatively in-depth and comprehensive optimization algorithms academically when faced with the situation of processing skewed data. While the concept of Spark was only proposed in 2012, with relatively short years and facing data skew problems on the Spark platform, although there are some accumulation and summary of processing and optimization methods in industry and academia, it is still limited compared with MapReduce. Therefore, when processing skewed data, Spark can learn from the research and methods of MapReduce for processing skewed data.

In MapReduce, to address data skewing issues, many investigators have proposed ways to improve partitioning strategies based on key 's frequency of occurrence[2, 3]. B. Gofler[4] et al proposed two load balancing Partitioning algorithms, Dynamic Fragmentation and Fine Partitioning, which repartition the output data of Map task so that the data sets of each Reduce task are roughly the same. To a certain extent, the data skew problem is alleviated. Son et al[5] designed MapReduce sampling operation to detect the frequency of internal data key , further predict the global data distribution, and then develop a more accurate and balanced partitioning strategy. Most sampling-based solutions, however, need to wait until all Map Task is completed before starting. Ramakrishnan et al[6] proposed the progressive sampling algorithm and Kwon et al proposed the SkewTune method[7] to reduce the waiting time of the above schemes, however the former algorithm requires an additional sampling stage before the job runs to arrange the partition plan, and the latter method introduces an additional overhead during the job runs, which detects the speed of task execution on each workstation, divides the slow task into multiple new tasks to process the remaining data, and improves the execution speed, But it uses Hadoop 's resource manager and is not suitable for the Spark platform. Q.Chen et al[8] used the LIBRAF method for migration of skewed data, in which a new data sampling method was used and the traditional Rang Partition was improved to cut large data clusters, making the data input at the Reduce stage more balanced and ensuring the order of intermediate data, and also considering the relevant scheduling problems in heterogeneous environments,

ICCVIT 2023, August 25–28, 2023, Chenzhou, China

ACM ISBN 979-8-4007-0870-1/23/08…$15.00
https://doi.org/10.1145/3627341.3630380

but this method requires completing the sampling task on the machine that completes the Map task first and is not suitable for the Spark platform that needs to wait until all Map tasks are completed before starting subsequent work.

In Spark, Yu et al[9] adopted an Adaptive strategy called Spark Adaptive Skew Mitigation (SASM), which alleviates the problem of data skew through data migration. The SP-Partitioner scheme adopted by Liu et al[10] is more suitable to solve the problem of data skew in the case of computing stream data, but its sampling method is interval sampling, which lacks flexibility. When the data volume becomes huge, the partition table will become huge, resulting in difficulties in maintaining the partition table. Tang et al[11] proposed a SCID(Splitting and combination) algorithm for intermediate data. The algorithm first uses pond sampling to sample the input data, and then calculates the output data in the Map stage according to the SCID algorithm to obtain a corresponding prediction partition table. In the shuffle phase, partitions are performed according to the predicted partition table obtained in the Map phase, so that the data in the next REduce phase is more average. However, in the worst case of SCID algorithm, each data cluster must be split at least once, and there may be a large number of intermediate data needs to be transferred in the cluster, which will waste valuable network I/O resources, increase the burden of the cluster and reduce the efficiency of data processing.

Gavagsaz et al [12] also proposed a slant data fine-grained partitioning algorithm for common join operations in data analysis. This algorithm uses stream sampling to obtain important attributes of input and output data, and then uses a multistage algorithm to segment input tuple data and propose the degree of load balancing among various tasks. This improves the execution efficiency of Map Reduce when processing skewed data. By using the relationship measured by Shannon entropy, Chen et al[13] obtained the information changes of data sets at different stages of Map Reduce, and then proposed a new Map Reduce connection algorithm based on dynamic partitioning strategy to improve the efficiency of connection operations in the case of data skew. Li et al[14] proposed a new parallel programming model to deal with uneven data in Map Reduce process, and further designed adaptive scheduling and a new pre-processing scheduling scheme to realize the model. Singh et al[15] proposed a data skew algorithm in Map Reduce programming framework that can balance the tasks in each mapping phase of the load. The algorithm can analyze the expected time required by tasks in the mapping phase, and realize load balancing of the data volume of each task by migrating large tasks and slow tasks to fast tasks. Reduces the completion time of the slowest tasks, thereby optimizing the problem of data skew.

3 DATA SKEW PROBLEM INTRODUCTION

Data skew[16] is a common problem in distributed systems . When the size of some data clusters with the same key is very large, it will lead to reduced efficiency of cluster parallel computing, and most tasks are quickly executed, while a small number of tasks will be performed for a very long time due to overload, and even memory overflow problems will occur, which also violates the original intention of parallel computing and fails to make full use of the computing resources of clusters.

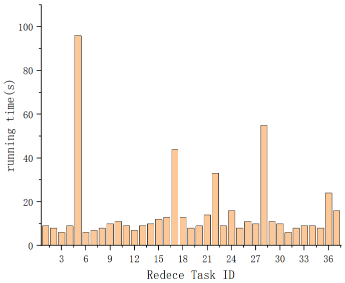

Figure 1: Reduce Performing Time Diagram for Each Task with Data skew Problem

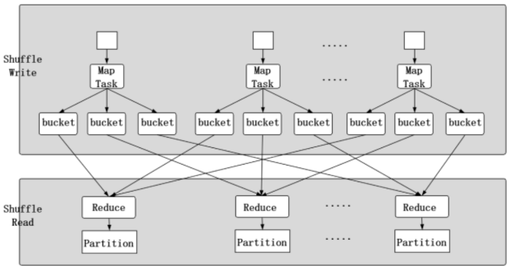

Figure 2: Spark shuffle process

In data processing platforms such as MapReduce and Spark, the problem of data skew is caused by the operation of the shuffle stage, which in turn affects the execution speed of the Reduce task and eventually delays the execution time of the entire job. When performing shuffle, the data with the same key on each working node is concentrated in a task on a certain node for processing. Most keys have only a few hundred pieces of data, and when individual keys have millions of pieces of data, most Reduce tasks will then be divided into only a small amount of data, while individual Reduce tasks will be assigned to a large amount of data that needs to be processed. As shown in Figure 1, this is the case of data skew, and the data volume of the four groups of tasks with Reduce Task id of 5, 17, 22, and 28 is much greater than the average data volume of other tasks, while the execution time of the entire Spark job is determined by the task with the longest running time. So in the case of data skew, Spark jobs tend to have very long stays.

Shuffle[17] is a bridge connecting Map and Reduce. The output of upstream Map task must pass through Shuffle process to be transmitted to downstream Reduce task. Spark, as an implementation of MapReduce framework, also implements Shuffle logic. Spark divides the partition of Map task output data into Shuffle Write process, while the process of pulling data and aggregating data from Reduce task is called Shuffle Read process.

Early Spark versions default to a hash-based Shuffle writing mechanism, a schematic of which is shown in Figure 2 , where each Map task creates the corresponding number of Buckets based on the number of downstream Reduce tasks, so the total number of Buckets created is MR, where M is the number of Map upstream tasks and R is the number of downstream Reduce tasks .

4 SPARK SHUFFLE DATA SLOPE MODEL

Assuming there are m different keys, the dataset M of all keys at Mapper stage is denoted as

$$M = \{k_1, k_2, k_3 ..., k_i, ...k_m\} \tag{1}$$

The intermediate dataset I is the dataset of each partition < key, value > key-value pair at Mapper end and each partition < key, value > key-value pair "Cartesian product" at Reduce stage , which can be expressed as

$$I = \left\{ \begin{array}{c} \left(k_1, v_1^{k_1}\right), \left(k_1, v_2^{k_1}\right), ..., \left(k_1, v_i^{k_1}\right), ... \left(k_1, v_{n_{k_1}}^{k_1}\right) \\ \left(k_i, v_1^{k_i}\right), \left(k_i, v_2^{k_i}\right), ..., \left(k_i, v_i^{k_i}\right), ... \left(k_i, v_{n_{k_i}}^{k_i}\right) \\ \left(k_m, v_1^{k_m}\right), \left(k_1, v_2^{k_m}\right), ..., \left(k_1, v_i^{k_m}\right), ... \left(k_1, v_{n_{k_m}}^{k_m}\right) \end{array} \right. \tag{2}$$

Where i is the j th value of m , is the number of arrays with $V_i^{k_i}$ as key for intermediate data, that is, the number of k_i corresponding values, j is the < key, value > key pairs in each barrel of Reduce terminal , can be expressed as

$$R = \left\{r_1, r_2, ..., r_j, ...r_n\right\} \tag{3}$$

Where i takes a value between 1 and the size of Reduce stage partitions . r_j is the amount of data of the Reduce terminal j th bucket, which is the sum of data consumed at j th Reduce and can be expressed as

$$r_j = \sum_{i=1}^{m} A_{i,j} \tag{4}$$

Where i takes values ranging from 1 to the size of Mapper stage partitions, $A_{i,j}$ is the data that produces the j Reduce consumption in the i Mapper, the cluster of Mapper stage of intermediate data $\mathrm{C(k_i)}$ is the data set of the same key in all Mapper partitions, which can represent

$$C\left(k_i\right) = \left(k_i, V_1^{k_i}\right), \left(k_i, V_2^{k_i}\right), \left(k_i, V_3^{k_i}\right), ..., \left(k_i, V_m^{k_i}\right) \tag{5}$$

Where i takes a value between 1 and the number of Mapper stage partitions . The data size of the i key of Mapper terminal is represented by $|C(k_i)|$, where the number according to $RC(j)$ of j Reduce terminal is the sum of the data size of Mapper 's data cluster from 1 to i key, which can be represented as:

$$RC\left(j\right) = \sum_{i=1}^{k_i} \left|C\left(k_i\right)\right| \tag{6}$$

Where i takes values between 1 and i th key. The mean value avg in each Reduce stage partition is the sum of the data volumes from Reduce stage 1 to m partitions divided by the total Reduce stage number of partitions, which can be expressed as:

$$avg = \frac{\sum_{j=1}^{m} RC\left(j\right)}{m} = \frac{\sum_{j=1}^{m} \sum_{i=1}^{k_i} \left|C\left(k_i\right)\right|}{m} \tag{7}$$

The standard deviation is used to calculate the data skew value of each Reduce partition, and the data skew value of all partitions is calculated to uniformly quantify the Reduce partition.

$$sd = \sqrt{\frac{\sum_{j=1}^{m} \left(avg - RC\left(j\right)\right)}{m}} \tag{8}$$

sd it can represent the fluctuation range of Reduce terminal dataset. In big data processing, for different datasets, the total amount of datasets varies greatly. If only the standard deviation is used, it will be flawed to use the standard deviation to measure the skew value.

$$Aox = \frac{std}{avg} \tag{9}$$

Aox The value reflects the degree of load balance of Reduce Aox. a smaller value indicates that the amount of data in each bucket is closer, the data inclination degree is smaller, and conversely, it means that the data volume of bucket is more uneven Aox. When value is zero, it reaches the ideal value, and the data volume in each bucket is exactly the same.

5 SPARK SHUFFLE PARTITION OPTIMIZATION ALGORITHM

In order to solve the data tilt problem in the process of Spar shuffle, based on the data tilt model proposed in section 3, this paper proposes a shuffle partitioning algorithm that can solve the data tilt problem in the process of shuffle. The partitioning algorithm is divided into the following four steps. In the third step, data partitioning policies include load balancing policies and Hash partitioning optimization policies:

1)Intermediate data sampling; 2)Overall dataset prediction; 3)Formulating data partitioning strategies; 4) completing data partitioning.

5.1 Intermediate Data Sampling

First, the impoundment optimization algorithm[18] is used to perform efficient equal-probability sampling of intermediate data distributed at various working nodes in the cluster. This algorithm only needs to scan the intermediate data once to complete the sampling, and removes the sample space and does not need to occupy additional space.Following data sampling, the following information can be obtained:

Sample data for each Task, which can be expressed in Equation 10 :

$$RD = \{(k_1, V\left(v_1^{k_1}\right)), ..., (k_i, V\left(v_i^{k_i}\right)), ..., (k_1, V\left(v_1^{k_1}\right))\} \tag{10}$$

$V(v_i^{k_i})$ is the space occupied by k_i.

2)Total number of middle keys per Task *num*.

3)the total space occupied by the intermediate data for each Task S.

5.2 Overall dataset prediction

The features of all intermediate data are drawn from the pooled reservoir optimization algorithm into the master node.These data can be expressed in Equation 11 :

$$KC = \{(k_1, S_1, V_1), .., . (k_i, S_i, V_i), ..., (k_m, S_m, V_m)\} \tag{11}$$

Among them, takes values ranging from 1 to key, k_i represents the i th data cluster, represents k_i the average space occupied by the data cluster, and V_i represents k_i the space occupied by the data cluster.

In addition, by summarizing the space occupied by each data cluster , the total space TS occupied by all data and the number n of user-defined partitions are obtained, and the average capacity in each bucket can be expressed by Equation 12:

$$avg = \frac{TS}{n} \tag{12}$$

5.3 Developing Data Partitioning Strategy

According to the relevant data predicted in the previous step, this step has to place the data cluster in a reasonable bucket in a relatively optimal manner, which is similar to a binning problem. Because the binning problem has been proved to be an NP-Hard problem, it is impossible to find the optimal solution in polynomial time, and can only find the approximate optimal solution in a short time as far as possible, so the prepartition algorithm is designed, that is, load balance partition algorithm and Hash partition optimization algorithm.

5.3.1 Load Balancing Partitioning Algorithm.

The load balancing algorithm is presented in Algorithm 1.

Algorithm 1 Load Balancing Partitioning Algorithm

```
The initial sample data is average
Loop Judge if any of the data clusters are greater than the average
Calculate amount of data already in bucke
Filled bucket plus one
Record Partition Dataset
Calculate the remaining space for this data cluster
End loop
Sort data clusters
Loop Traversal Data Cluster
Found data cluster not full
Judge if the current bucket remaining space is larger than the data cluster size
Directly into the partition
Judge if there is bucket remaining space greater than this data cluster,
Find the largest ID for the remaining bucket
Else Find the ID with the largest bucket space
End if
Record Partition Dataset
Calculate the space remaining in the bucket
End loop
```

The input parameter data cluster information *KC* and average bucket capacity *agv* of the algorithm are described in Equations 11) and (12), respectively, and the output result *PC*of the algorithm indicates the pre-partition set, which contains the information of key and the id that the key should be assigned to the corresponding bucket as well as the corresponding number of assignments.

The idea of algorithm 1 is to first initialize the remaining capacity of each bucket with avg and then obtain an initial target bucket with Hash 's method. If the bucket is full, find the bucket with the smallest remaining capacity in the bucket of the data cluster from the remaining bucket; if the remaining capacity of all buckets is less than the size of the data cluster, find the bucket with the largest remaining capacity to place the data cluster. The algorithm uses the advantage of hash partition method to complete the data partition under the time complexity of O(nlogn). Here n is the number of data clusters. Generally, the number of data clusters will be many orders of magnitude smaller than the intermediate data. Theoretically, the algorithm has excellent performance and good pre-partition results. The algorithm finally outputs a pre-partition table that records the data bucket corresponding to each key value and the number of buckets it is to be placed in.

5.3.2 Hash Partition Optimization Algorithm.

The Hash partition optimization algorithm is shown in Algorithm 2.

Algorithm 2 Hash Partition Optimization Algorithm

```
The initial sample data is average
Loop Traversal Data Cluster
Get a partition randomly, Judge if the current bucket remaining space is larger than the data cluster size
Directly into the partition
Else if judge that the remaining space of two adjacent partitions is larger than the space of this data cluster
Find the largest ID for the remaining space bucket
Else find bucket space max
End if
Record Partition Dataset
Calculate the space remaining in the bucket
End loop
```

The idea of Algorithm 2 is mainly to initialize the remaining capacity of each bucket with *avg* and then generate a random number as the default bucket in n partitions using a random function. Judge whether the bucket data volume where the random function is located reaches the average value, put the current bucket if it fails to reach the average value, if it has reached the average value, obtain the residual capacity of two adjacent buckets, compare the capacity size of three buckets, and place the current data cluster into the bucket with the smallest capacity. The algorithm uses a random function to disrupt the initial bucket 's position so that the initial bucket' s position is as evenly distributed as possible. Data partitioning can be accomplished at the time complexity of O(nlogn), where n is the number of data clusters, which are generally many orders of magnitude smaller than intermediate data. Theoretically, the algorithm has excellent performance and good pre-partition results. The algorithm finally outputs a pre-partition table that records the data bucket corresponding to each key value and the number of buckets it is to be placed in.

5.4 Completion of data partitioning

This step will partition the full amount of intermediate data according to the pre-partition table obtained in the previous step. First, query the table for keys with current intermediate data, if any, place them according to the bucket indicated in the partition table; if not in the pre-partition table, partition them according to the method of the Hash partition. In the implementation, the pre-partition table is saved by HashMap, which only requires O(1) time complexity for the process of each data partition, and the partition process is very efficient. With such a partition scheme, each reduce task after shuffle receives approximately the same amount of data.

In order to facilitate setting the skew of the raw data, the raw data with different skew was generated experimentally using the zipf 's law[19] to simulate the real skew data. The execution time

Table 1: Spark Cluster Configuration Table

Node Type	Quantity	Memory/GB	Hard disk	Deployment Mode	System Environment
Master	1	4	100	Standalone alone	JDK1.8, Scala-sdk-2.12.11, Apache Hadoop3.0
Slave	3	4	100		

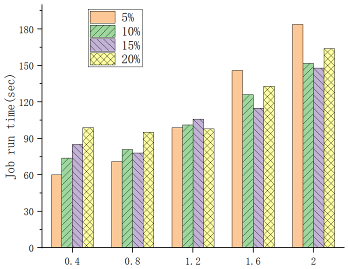

Figure 3: Execution time of partition algorithm under different sampling probabilities

of the Spark job under sloping data and the degree of data equalization after shuffle Aox were selected as performance measures for the shuffle mechanism. In order to verify the performance of the shuffle optimization scheme, two control partition experiments were selected.

1) Hash partition. This is the default method used by the Spark platform. The algorithm is simple and uses the key hash value of intermediate data directly to modulate the number of partitions to obtain the partition id of the key.

2) Range partition. This is another partition method provided in Spark, which is used to map a certain range of data into a partition and try to keep the number of data in each partition relatively uniform. This mechanism also requires that intermediate data be sampled first.

6 EXPERIMENTAL VALIDATION

The experimental environment is a Spark and HDFS cluster built for four virtual machines[20] , and the hardware and configuration of each node are shown in Table 1 .

The word count[21] job is used to verify the performance of the partition option in the key skew case, where Hash is the default Hash partition algorithm of Spark, G-Hash is the Hash partition optimization algorithm, LDP is the load balance partition algorithm, and Range is the Range partition algorithm of Spark. First, five sets of data with key inclination of 0.4, 0.8, 1.2, 1.6, and 2.0 were generated by the zipf method, and the size of each text data was 1 GB. Because LDP and G-hash in the shuffle optimization option involve sampling, the execution time of each job is first calculated with different sampling rates. Figure 3 shows job execution times for the four partitioning algorithms for sampling rates of 5%, 10%, 15%, and 20%. In order to avoid the influence caused by different sampling rate, the average value of the four sampling rate results is selected to calculate the execution time and Aox value of the four partition algorithms.

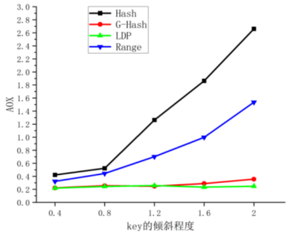

Figure 4: Aox Values for four partition schemes

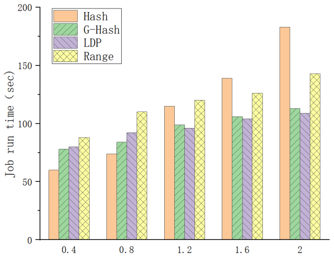

Figure 5: Execution time of four partition algorithms

Figure 4 shows Aox changes at slopes of 0.3, 0.7, 1.2, 1.6, and 2.0 for the four partitioning schemes Hash, G-Hash, LDP, and Range. At slopes of 0.3 and 0.7, their Aox values are close and not significantly different. When the skew is above 1.2, the Aox growth of Hash method is very fast, of Range algorithm has been relatively stable growth, LDP option and G-Hash option have very good load balancing ability, has been maintained at a very low level.

Figure 5 shows job execution time comparisons for the four partitioning schemes. When the slope of key is 0.8, Hash 's option is the fastest because it doesn' t take any extra time to partition and can handle very low slope data better. However, with the increase of inclination, the execution time of Hash option increases rapidly, and the operation execution time is the longest at inclination 2.0. LDP option and G-Hash option execution time above 1.2 inclination has been the lowest, and execution time is very stable. Range has a stable execution time but a long execution time. From the experimental results, it can be seen that LDP option and G-Hash option can effectively respond to the data skew situation, with good load balancing ability and efficient execution efficiency.

7 SUMMARY

According to the problem of data skew caused by shuffle stage in Spark, a skew model that can uniformly quantify <key, value> data skew is proposed in this paper, and then load balance partition option and Hash optimization partition option are proposed based on this skew model . Finally, the verification experiment shows that the model and option have universality and high efficiency to reduce the total completion time of the application program and improve the load balance of Spark. The main contributions of this paper are as follows:

1) The research status of data skew problem is analyzed, and the previous solutions of MapReduce and Spark for data skew are summarized and analyzed .

2) A < key, value > data skew model is established; In the Spark platform, intermediate data always appears in the form of key pairs such as < key, value>. According to the space occupied by intermediate data < key, value > , this paper comprehensively evaluates the skew of data and uniformly and objectively quantifies the skew of intermediate data in Spark, providing a theoretical basis for the subsequent development of partition strategies. The objective function in the model serves as an important indicator for assessing the degree of load balance in Spark and Reduce jobs.

3) The load balance partitioning and Hash partitioning optimization algorithm are proposed; the load balance partitioning algorithm can adopt different strategies to efficiently and rationally complete the data allocation according to different inclination degrees of intermediate data. The load balancing partitioning algorithm has a time complexity of O(nlgn) , which can effectively solve the data skew problem . The Hash partition optimization algorithm uses random function to distribute data approximately equally from the source, and selects smaller buckets from adjacent buckets to distribute data when the amount of data in the bucket is greater than the average. The Hash partition optimization algorithm time complexity is also O(nlgn).

The main innovation of this paper is that 1) presents a mathematical model that can quantify the degree of data skew in big data processing, which has the ability to uniformly evaluate data skew. 2) In Spark platform, load balance partition algorithm and Hash partition optimization algorithm are proposed. Load balance partition algorithm can select suitable partition strategy according to the inclination degree of processed data and efficiently complete the data partition work. Hash partition optimization algorithm can select smaller capacity buckets from adjacent buckets and then efficiently complete the data partition work , improving the execution efficiency of Spark job under the condition of data inclination.

REFERENCES

[1] Ping L. Research on University Campus Big Data Platform Based on Hadoop and Spark [J] .Software Engineering, 2018:2-34.

[2] Midoun K, Loudini M, Hidouci K W, *et al.* LoEM: Improving Load Balancing for MapReduce-based Matching [J]. International Journal of Artificial Intelligence, 2019, 17 (2): 217-235.

[3] Martha V S, Zhao W, Xu X .h-MapReduce: A Framework for Workload Balancing in MapReduce [C]//IEEE Computer Society.IEEE Computer Society, 2013:637-644.

[4] Gufler B, Augsten N, Reiser A, *et al.*HANDLING DATA SKEW IN MAPREDUCE [C]//CLOSER 2011 - Proceedings of the 1st International Conference on Cloud Computing and Services Science, Noordwijkerhout, Netherlands, 7-9 May, 2011:1-7.

[5] Son J, Choi H, Chung Y D .Skew-Tolerant Key Distribution for Load Balancing in MapReduce [J] .Ieice Trans.inf . & Syst, 2012, 95 (2): 677-680.

[6] Ramakrishnan S R, Swart G, Urmanov A. Balancing reducer skew in MapReduce workloads using Computsampling [C]//Third Proceedings of the Progressive ACM Symposium on Cloud Computing.ACM, 2012. 1-4.

[7] Kwon Y C, Balazinska M, Howe B, *et al.*SkewTune: Mitigating Skew in MapReduce Applications [J] .Proceedings of the Vldb Endowment, 2012, 5 (12): 1934-1937.

[8] Chen Q, Yao J, Xiao Z. LIBRA: Lightweight Data Skew Mitigation in MapReduce [J] .IEEE Transactions on Parallel & Distributed Systems, 2015, 26 (9): 2520-2533.

[9] Yu J, Chen H, Hu F .SASM: Improving Spark Performance with Adaptive Skew Mitigation [C]//2015 IEEE International Conference on Progress in Informatics and Computing , 2015:102-107.

[10] Liu G, Zhu X, Wang J, *et al.*SP-Partitioner: A novel partition method to handle intermediate data skew in spark streaming [J] .Future Generation Computer Systems, 2017, 86 (3): 1054-1063.

[11] Tang Z, Zhang X, Li K, *et al.* An intermediate data placement algorithm for load balancing in the Spark environment [J] .Future Generation Computer Systems, 2016, 78 (1): 287-301.

[12] Elaheh Gavagsaz, Ali Rezaee, Hamid H. S. Javadi. Load balancing in join algorithms for skewed data in Map Reduce systems. The Journal of Supercomputing, 2019, 75(1): 220-254.

[13] Donghua Chen, Runtong Zhang. Map Reduce-based dynamic partition join with shannon entropy for data skewness. Scientific Programming, 2021, 175-209.

[14] Jianjiang Li, Yajun Liu, Jian Pan, Peng Zhang, Wei Chen, Lizhe Wang. Map-Balance-Reduce: An improved parallel programming model for load balancing of Map Reduce. Future Generation Computer Systems, 2020, 105: 973-998.

[15] Balraj Singh, Harsh K. Verma. IMSM: An interval migration based approach for skew mitigation in Map Reduce. Recent Advances in Computer Science and Communications, 2021, 14(1): 61-81.

[16] Kotoulas S, Oren E, Harmelen F V. Mind the data skew: Distributed inferencing by speeddating in elastic regions [C]/Proceedings Conference of the 19th International dating on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010.ACM, 2010. 1-33.

[17] Rana N, Deshmukh S. Shuffle Performance in Apache Spark [C]//International Journal of Engineering Research & Technology.ESRSA Publications, 2015:10-44.

[18] Aggarwal C C. On Biased Reservoir Sampling in the Presence of Stream Evolution. [C]//Very Large Data Bases Conference.2006:33-46.

[19] Adamic, Lada A.. Zipf 's law and the Internet.' glottometrics (2002): 3 (1): 143-150.

[20] Li-Jie X U .Construction and Research of Big Data Processing Platform Based on Spark [J] .Computer Knowledge and Technology, 2016:10-23.

[21] Pennebaker J W, Francis M E, Booth R J. Linguistic inquiry and word count (LIWC) [J] .Lawrence Erlbaum Associates Mahwah Nj, 2001; 24-56.