Query Optimization Algorithm of Replication Join Based on Sampling Partition

Xin Lu¹, Junchao Yang¹*, Jiao Yuan², Xun Wang², Kun Fu², Ke Yang²

¹School of software, University of Electronic Science and Technology of China, Chengdu, Sichuan, 610054, China
²China Railway Eryuan Engineering Group Co. Ltd, Chengdu, Sichuan, 610031, China
* 353605473@qq.com, 201821090208@std.uestc.edu.cn

Abstract. Aiming at the low efficiency of join query in MapReduce traditional partition join algorithm when data skew, a replication join optimization algorithm based on sampling partition is proposed. According to the sampled statistics of connection attribute data, the algorithm divides the datasets in connection relationship into skewed data subset and non skewed data subset. In order to optimize the query performance, join query processing is carried out on them respectively. For the join queries of non skewed data subsets, the improved consistency hash function is used to partition these subsets, so that the load of data connection query processing of each node is balanced. For the skewed data subset join query, the smaller skewed data subsets are distributed to each node, and the larger skewed data subsets are partitioned according to the non skewed fields. In the Reduce stage, these skewed data subsets are join queried. Experiments show that the algorithm can optimize the join query performance under different data skew rates, and achieve efficient join query processing of large datasets.

1. Introduction
With the wide application of Internet of things, cloud computing, artificial intelligence and other new technologies, big data is constantly generated, which brings difficulties to data processing and analysis. With its good scalability, high availability and fault tolerance ability, Hadoop is suitable for intensive computing tasks with massive data. MapReduce computing model, one of the core components of Hadoop, can process the massive data of TB level or even PB level stored on HDFS through user-defined map and reduce function. Join query is one of the core operations of massive data processing. It is frequently used in data management, OLAP and other intensive data computing fields. The idea of join query for large datasets comes from the join algorithm in traditional database, and improves the performance with the help of MapReduce parallel computing framework. In connection query of two large datasets, the default partition strategy may not distribute data evenly to each node because it is difficult to accurately estimate the data distribution. The data processing time of each node is quite different. Data skew is easy to occur, which will affect the processing efficiency of distributed computing platform.

As one of the typical join query methods, the replication join algorithm compares the sizes of two datasets in connection query and copies the smaller dataset to the nodes of larger dataset. However, when two large datasets are used for join query, distributing one dataset to another dataset node will result in large network transmission data cost, which will lead to the increase of join query response
time. In this paper, an improved replication join algorithm is proposed to solve this kind of join query performance problem. The algorithm divides the datasets in the join relationship into skewed data subset and non-skewed data subset by sampling statistics, and processes the join queries respectively, so as to optimize the performance of join queries for large datasets.

2. Related research
At present, there are two kinds of join query algorithm optimization of MapReduce, which are join query algorithm based on sampling data partition and join query algorithm based on load balance.

By sampling the datasets involved in the join operation, the join query algorithm based on sampling data division can understand the data distribution of the dataset in the whole connection relationship, and then adjust the data distribution on the reduce nodes, so as to reduce the cost of join query. Ya Zhou et al. [1] aimed at the performance bottleneck problem of traditional join query algorithm under data skew, a join query optimization algorithm based on statistical skewed data and polling partition was proposed. The algorithm distributes data to each computing nodes of Hadoop cluster by sampling statistics and polling partition strategy. Guipeng Liu et al. [2] proposed a partition method to balance the load of Reduce nodes for the partition data skew problem of Spark Streaming data stream. This method predicts the characteristics of the intermediate data by sampling the dataset samples, generates a reference table. According to the prediction results, the table guides the data distribute to each Reduce node evenly. M.Al Hajj Hassan et al. [3] proposed a frequency adaptive connection algorithm MRFA. According to the frequency of each value in the join attribute, the algorithm redistributes the highly skewed data based on the distribution histogram and random key redistribution method to reduce the data transmission cost and ensure the load balance of the data processing nodes. Wenxia Guo et al. [4] proposed a Reduce partition method to solve the data skew problem in Spark calculation of Reduce stage. It allocates tasks by using greedy strategy through the overall distribution of sampling datasets. These algorithms are only suitable for range partition, but have limitations for other partition methods. In addition, the sampling process of these algorithms is usually a single thread traversal of the dataset, and its efficiency is low. When a large amount of data will to be partitioned, a large amount of network overhead will produced.

The join query algorithm based on load balancing is generally based on the load situation of each reduce node, so as to reasonably distribute the data in each reduce node, and ensure that the task execution time of each node is generally consistent. As a join query algorithm based on load balancing, replication join algorithm transfers smaller datasets to the node where another dataset is located, and then performs join query processing on them, and finally merges the result data. If the size of the two datasets is quite different, the traditional partition replication algorithm has higher efficiency. However, when the size of both two datasets is large, the network transmission cost of the algorithm is large and the response time of join query is long. Joanna Berlińska et al. [5] analyzed the influence of data skew in MapReduce computing, proposed four different types of load balancing methods, and compared their applicability under different data skew distribution. Zhu Wang et al. [6] proposed an incremental data allocation method to reduce partition data skew in MapReduce. Mohamad Amin irandoost et al. [7] proposed an adaptive partition algorithm LAHP (learning automata hash partitioner). When the algorithm detects that the running load exceeds the target seted in the Reduce node, the data exceeding the load will be distributed to other nodes through the automatic learning machine to ensure the dynamic load balance. Elaheh gavagsz et al. [8] proposed a fine-grained partitioning algorithm for data skewed. It proposes stream sampling processing according to the attributes of input dataset, and introduces a new method of allocating input data, which can effectively repartition the dataset and improve the load balance of running nodes. Foto affati et al. [9] proposed a multiple connection algorithm for processing highly skewed data in MapReduce. The algorithm allocates the data size of each node based on the weight value, which reduces the data transmission cost between nodes, and keeps the data size of each node relatively balanced. Although these algorithms can effectively reduce the impact of data skewed through the load balancing of each node, most of them are complex, and their operation takes up more system resources, and the network transmission cost is also high.
3. Dataset join query definition
Suppose that the two datasets participating in join query operation are A and B. The join query operation between them is shown in Formula 1.

\[ \sigma_{A \times B}(A \times B) \]  

(1)

In this formula, the connection property is \( x \in A \cap B \), and the connection condition is \( A.x = B.x \).

The unconnected attributes are \( y \in A \cup y \notin B, Z \in B \cup Z \notin A \).

If the join query has query conditions, the join query operation is shown in formula 2.

\[ \pi_{A \times B}(A \times B) \]  

(2)

In this formula, the query conditions are \( S_A \) and \( S_B \), and the projection attributes are \( A.y \) and \( B.z \).

Dataset skewed degree refers to the proportion of data skewed in the dataset. For the dataset of join query operation, we can judge the data skewed of the dataset by comparing the number of records of each connection value and the total number of connection records / (the number of connection values). If the number of records of each connection value is greater than the total number of connection records / (the number of connection values), the data is skewed.

The traditional replication join query algorithm calculates the time cost of join query between A and B datasets as shown in Formula 3.

\[ C_{pcj} = C_{\text{tran}}(B) + C_{\text{par}}(A) + \max\{C_{\text{proc}}(Ri)\} \]  

(3)

In this formula, it is assumed that the size of dataset A is larger than that of dataset B. \( C_{\text{tran}}(B) \) represents the time required to load dataset B into the distributed cache and distribute it to each node in the Map stage. \( C_{\text{par}}(A) \) indicates the time required for dataset A to be partitioned. \( \max\{C_{\text{proc}}(Ri)\} \) represents the longest time for connection calculation between dataset A and dataset B in each node.

The time cost calculation of standard repartition join query algorithm on A and B datasets is shown in formula 4.

\[ C_{\text{spj}} = \max\{C_{\text{filter}}(B), C_{\text{filter}}(A)\} + \max\{C_{\text{par}}(A), C_{\text{par}}(B)\} + \max\{C_{\text{proc}}(Ri)\} \]  

(4)

In this formula, \( \max\{C_{\text{filter}}(B), C_{\text{filter}}(A)\} \) is the longest time required for filtering intermediate result data in A and B datasets in Map stage. \( \max\{C_{\text{par}}(A), C_{\text{par}}(B)\} \) is the longest time required for A and B datasets to be repartitioned in Shuffle stage. \( \max\{C_{\text{proc}}(Ri)\} \) represents the longest time of connection calculation between dataset A and dataset B in each node of Reduce stage.

4. Replication join optimization based on sampling partition
Aiming at the performance bottleneck of join query in MapReduce data skewed scenario, this paper proposes a replication join optimization algorithm based on sampling partition (SPCJ). Based on the existing MapReduce replication join query algorithm, the algorithm improves three aspects: data sampling statistics, consistent hash partition method and replication join mode.

4.1. Sampling statistics
For large-scale datasets, the general distribution of various types of data in the dataset can be mastered through sampling statistical method [10]. In the map stage, it is necessary to sample the dataset of join query, so as to determine whether there is skewed data in the dataset. When the size of the dataset is very large, the whole dataset cannot be directly loaded into the memory for sampling operation. To solve this problem, this paper proposes a multi-thread random sampling skew statistics method for massive data.

The multi-thread random sampling statistics method of massive data refers to the reservoir sampling idea [11]. The \( n \) pieces of data need to be randomly selected from the dataset with the total number of data \( N \), and the skew statistics is carried out. The process is as follows:

Step 1: divide the dataset into \( k \) sub datasets, and the data volume of each sub dataset is \( N/k \).

Step 2: select the first \( n/k \) data in each sub dataset and save them in the set \( S_j \) (\( 1 \leq j \leq k \)).
Step 3: start from the i-th \((n/k+1 <= i <= n/k)\) data of each subset, and calculate the probability \(P=n/(k*i)\) whether the data is left or not through multi-thread calculation. If the i-th data is left, a piece of data is randomly selected from \(S_j\) and replaced with the i-th data, otherwise the i-th data is eliminated.

Step 4: repeat step 3 until the data from \(n/k+1\) to \(N/k\) in each sub dataset are traversed. Each set \(S_j\) is merged into \(S\), and the data in set \(S\) is n random data extracted by the algorithm.

Step 5: traverse the n pieces of data obtained by sampling, update the values of the following three counters, and record whether the data in the set \(S\) is skewed or not.

Three global counters: \(\text{joinkeynum, joinrecordnum, joinrecord < key, values >}\). \(\text{joinkeynum}\) records the number of connection values. \(\text{joinrecordnum}\) records the number of data pieces participating in the connection. \(\text{joinrecord < key, values >}\) records the number of records per connection value. Where values represents the number of times a key value appears. If values > \(\text{joinrecordnum / joinkeynum}\), then the record data is skewed data and stored in the skewed dataset \(R\).

Other sampling methods need random sampling according to the total data. In this sampling statistical method, n data are randomly sampled with \(O(N/k)\) time complexity. Compared with the conventional reservoir sampling method which traverses the whole dataset by single thread, this algorithm improves the efficiency by using multi-thread parallel processing. According to the sampling samples, this algorithm can count the frequency of each key value in the dataset, so as to record whether the key value is skewed or not.

4.2. Improvement of consistent hash partition function

MapReduce uses the consistent hash partition function to modulus \(2^{32}\) by default to realize data distribution to nodes [12]. In principle, the consistent hash partition function distributes data to the ring composed of cluster nodes. Its structure is shown in Figure 1 (a).

For example, there are four nodes A, B, C and D. According to their unique identification, they are mapped to the hash rings through hash functions. The input data is mapped to the ring according to the same hash function, and the nearest node by clockwise is found as its storage node. Due to the fact that the hash values of nodes can not be evenly distributed throughout the hash ring, the data skew phenomenon will appeared between cluster nodes in the case of large amount of data storage. Especially when one of the nodes fails, the data distributed on it will be redistributed to the node nearest to the node, which will cause the load of the next node to increase instantaneously and cause the phenomenon of downtime. In turn, a avalanche will occurred in the cluster [13].

In order to solve the above problems, this paper introduces the virtual node to subdivide the hash space of the ring. The hash value is calculated by adding different serial numbers to a physical node ID, which is used as the virtual node mapped on the hash ring. As long as the number of virtual nodes is enough, the distribution of virtual nodes is more uniform than that of physical nodes. The improved consistent hash function is mapped to the hash ring, and its data distribution is shown in Fig. 1 (b).

For example, A1, A2, B1, B2, C1, C2, D1, D2 are virtual nodes. Each input data is distributed to a location on the ring according to its hash value. As shown in Figure 1 (b), k1 is located between A1
and B1. The first node is found to be B1 in clockwise direction. Therefore, k1 data will be stored in virtual node B1, and the actual storage node is B.

In the improved consistent hash ring, when one of the physical nodes is down, the data stored in the node will migrate clockwise to the nearest virtual node of the next physical node according to the hash ring. Due to the balanced distribution of virtual nodes, the data will be uniformly stored in the physical nodes, so it is not easy to appear avalanche phenomenon.

4.3. Partition replication connection optimization

The existing partition replication join query algorithm can not effectively solve the problem of join query between two large datasets. However, in this algorithm, two datasets of join query are sampled and statistically processed, and they are decomposed into skewed data subset and non skewed data subset. Then partition replication join query is used for skewed data subset, and regular partition join query is used for non skewed data subset.

Because the A and B datasets of join query are unified when marking skewed data, the skewed data subset and the non skewed data subset between them will not generate join intersection. Therefore, we only need to consider the skewed data subset join processing and the non skewed data subset join query processing between them.

For the skewed data subset join queries of dataset A and B, we first compare their skewed data subset sizes. The smaller skewed data subset is copied to the larger skewed data subset node, and the larger skewed data subset is partitioned according to its non skewed fields. In the reduce stage, the skewed data subset of B in each node is connected with that of A.

For the join queries of non skewed data subsets of datasets A and B, the non skewed data subsets of datasets A and B are hash partitioned according to the value of join fields. In the reduce stage, join queries are performed on A and B for non skewed data subsets in each node.

Finally, the join query results of A and B are obtained by merging the join query result set of skewed data subset and the join query result set of non skewed data subset in A and B.

4.4. Algorithm implementation

Based on the algorithm processing ideas in Sections 4.1, 4.2 and 4.3, the specific implementation process of replication connection optimization algorithm based on sampling partition is shown in Figure 2.

4.4.1. Map stage.

Based on the sampling statistical method in Section 3.1, the datasets A and B are sampled. The number of join values, the number of join records, the number of records participating in the connection, and the number of records corresponding to each connection value are calculated. Determine the skew threshold of the dataset and mark the data whether skewed or not. Tag and dataskew are added to the connection attributes of the dataset to form a combinekey (key, tag, dataskew), where tag is the identifier distinguishing the dataset, such as A or B. The dataskew indicates whether the data is skewed. Finally, the key value pair data < combinekey (key, tag, dataskew), value > are output as the result of Map processing. Where combinekey (key, tag, dataskew) is the tag combination of the join key, and value is the non join key attribute value of the data. In the
map stage, the datasets are sampled and marked with data skewed. The pseudo code is described as follows:

**Map stage processing: data sampling and data marking processing**

**Input:** TableA, TableB {datasets A and B}

**Output:** Map < combinekey (key, tag, Datasheet), value > {output dataset with dataset ID and skew identifier}

1. Init joinrecord < key, values >, joinrecordnum, joinkeynum, R {initialization variable}
2. Reservoirsampling (tableA, tableb, joinrecord < key, values >, joinrecordnum, joinkeynum) {sampling statistical dataset information}
3. Isskew = joinrecordnum / joinkeynum {determines the skew threshold}
4. R [] = putskewkey (joinrecord < key, values >, isskew) {values determined to be skewed are saved in the set R}
5. Init map < key, value > {initialization map table is used to store intermediate result key value pairs}
6. For data in tableA and tableB
7. If R.contains (key) then {if the join key value of the dataset exists in R, it is skewed, otherwise it is non skewed}
8. dataskew=false
9. else
10. dataskew=true
11. map.put (combine key (key, tag, Datasheet), value) {stores the marked key attribute and non key value in the key value pair}
12. End for
13. Return map < combinekey (key, tag, Datasheet), value > {output intermediate result set}

4.4.2. **Shuffle stage.**

Traverse the output data < combinekey (key, tag, Datasheet), value > in the map stage, and classify them into the following subsets:

- Dataset A skewed data subset {< combinekey (key, a, true), value >}
- Dataset B skewed data subset {< combinekey (key, B, true), value >}
- Dataset A non skewed data subset {< combinekey (key, a, false), value >}
- Dataset B is a non skewed subset {< combinekey (key, B, false), value >}

For skewed data subsets, compare the number of {< combinekey (key, a, true), value >} and {< combinekey (key, B, true), value >}. Load the skewed data subset with a small amount of data into the cache, and distribute it to the nodes where the large amount of data is located. At the same time, the skewed data subset with large amount of data is processed by hash partition according to its non skewed fields.

For the non skewed data subset, the consistent hash partition algorithm in Section 3.2 is used for partition processing. At the same time, the non skewed data in the same partition is sorted according to tag and key to ensure that the data of a dataset is always together, and the data of the same key value of a dataset are also together. Output to the Reduce node in the form of < combinekey (unskewkey, tag), value >. The same is true for skewed data subsets. In the shuffle stage, the dataset is divided into subsets and partitioned. The pseudo code is described as follows:
Shuffle stage processing: data subset partition and partition process

Input: Map \textless\text{combinekey (key, tag, Dataskew)}, value\textgreater\ {intermediate dataset identified in map stage}

Output: \textless\text{combinekey (unskewkey, A), value\textgreater, \textless\text{combinekey (unskewkey, B), value\textgreater, \textless\text{skewkeyA, value\textgreater, \textless\text{skewkeyB, value\textgreater} \ {skewed data subset, skewed data subset}

1: For each \textless key, value\textgreater \text{in Map \textless\text{combinekey (key, tag, Dataskew)}, value\textgreater \ {traversing the intermediate dataset returned in the map stage}
2: If (dataskew = \text{false}) \ {if it is non skewed data}
3: unskewset.put (\textless key, value\textgreater) \ {divide it into unskewset, a non skewed data subset}
4: Else if (dataskew = \text{true} \& \& \text{tag = A}) \ {if it is skewed data and it is data in A}
5: skewsetA.put (\textless key, value\textgreater) \ {divide it into the skewed data subset of A}
6: else
7: skewsetB.put (\textless key, value\textgreater) \ {divide it into the skewed data subset of B}
8: End For
9: If (sizeof (skewsetA) < sizeof (skewsetB)) \ {compare the size of skewed data subset}
10: DistributedCache (skewsetB) \ {store smaller subset of data in distributed cache}
11: Hash (skewsetB) \ {partition large data subsets according to non skewed fields}
12: OrderByTagAndKey (\textless\text{skewkeyB, value\textgreater} \ {sort the skewed data subset in the partition}
13: else
14: DistributedCache (skewsetB) \ {store smaller subset of data in distributed cache}
15: Hash (skewsetA) \ {partition large datasets according to non skewed fields}
16: OrderByTagAndKey (\textless\text{skewkeyA, value\textgreater} \ {sort the skewed data subset in the partition}
17: ConsistentHash (unskewset) \ {performing consistent hash partition on non skewed data subset}
18: OrderByTagAndKkey (\textless\text{combinekey (unskewkey, A, value\textgreater, \textless\text{combinekey (unskewkey, B, value\textgreater, \textless\text{skykeyA, value\textgreater, \textless\text{skykeyB, value\textgreater} \ {output data of each subset}

4.4.3. Reduce stage.
Join query is performed on the subset of \textless\text{combinekey (unskekey, A), value\textgreater, \textless\text{combinekey (unskekey, B), value\textgreater} \ {subsets, and the output result is the join query result of non skewed data. Join query the subset of \textless\text{skewkeyA, value\textgreater, \textless\text{skewkeyB, value\textgreater} \ {skewed data subset join query}, and the output result is the join query result of skewed data. The final join query result data is obtained by combining the join query result data of non skewed data subset and the join query result data of skewed data subset. The pseudo code is described as follows:

Reduce stage processing: data result integration process

Input: \textless\text{combinekey (unskekey, A), value\textgreater, \textless\text{combinekey (unskekey, B), value\textgreater, \textless\text{skykeyA, value\textgreater, \textless\text{skykeyB, value\textgreater} \ {subsets of join query}

Output: resultFile \ {final join query result file}
1: New resultfile \ {new data file}
2: For each partitions \ {traverse each partition}
3: Resultunskew = connectunkew (\textless\text{combinekey (unskekey, A), value\textgreater, \textless\text{combinekey (unskekey, B), value\textgreater} \ {A, B non skewed data subset join query}
4: Resultskew = connectskew (\textless\text{skykeyA, value\textgreater, \textless\text{skykeyB, value\textgreater} \ {A, B skewed data subset join query}
5: Resultp = merge (Resultunskew, Resultskew) \ {integrate the skewed data join query result set and the non skewed data join query result set in a single partition}
6: Addresulttofile (Resultp, resultFile) \ {append the result set to the final result file}
7:   End For
8:   Return resultFile {return connection query result file}

5. Algorithm experiment and result analysis
In order to measure and evaluate the performance of our algorithm (SPCJ), standard repartition join query algorithm (SRJ) [14] and traditional replication join query algorithm (PCJ), we will carry out join query experiments based on standard datasets. This experiment uses different datasets, different sizes and different inclinations to test the response time of the above algorithms, and compares the running costs of these algorithms, such as CPU utilization and memory consumption.

5.1. Experimental environment and scheme
In the experiment, build Hadoop cluster. The cluster consists of two master nodes and three slave nodes. The processor model of each node is 12core 2.3GHz Intel Xeon Cold5118, and the memory size is 16GB. Each node is installed with Ubuntu 16.04lts operating system.

Select Orders and Customer in TPC-H [15] as the two datasets of join query. The customer dataset contains fields such as custkey (primary key) and name. The orders dataset contains fields such as orderkey (primary key), custkey (foreign key), and cleak. The two datasets are connected by the field custkey. The dataset related parameters of A (orders) and B (customer) used in the experiment are shown in Table 1.

| Data volume of dataset (Ten thousand pieces) | Dataset A | Dataset B | Dataset skewed degree |
|---------------------------------------------|-----------|-----------|-----------------------|
| Dataset A                                   | 1000      | 1000      | 0.2                   |
| Dataset B                                   | 2500      | 2500      | 0.4                   |
| Dataset A                                   | 5000      | 5000      | 0.8                   |
| Dataset B                                   | 10000     | 10000     | 0.8                   |

5.2. Analysis of experimental results
Each algorithm performs join query in different scale and different skewed degree datasets, and its response time result data is shown in Figure 3. The average CPU consumption of each algorithm is shown in Figure 4. The average memory consumption of each algorithm is shown in Figure 5.

![Figure 3. Join query response time of each algorithm.](image)
As can be seen from the result data in Fig.3, when the skewed degree of dataset is low, the response time of join query of each algorithm is roughly the same. However, when the skewed degree of the dataset become higher, the processing efficiency of this algorithm is higher and the response time is the fastest.

![Figure 4. CPU cost of each algorithm program.](image)

From the result data of Figure 4, we can see that the CPU utilization of each algorithm increases with the increase of dataset size and skewed degree. The CPU consumption of SPCJ algorithm is higher than that of the other two algorithms, but the algorithm improves the utilization of computing resources of each node.

![Figure 5. Memory cost of each algorithm program.](image)

From the result data of Figure 5, we can see that the memory cost of SPCJ algorithm is between PCJ algorithm and SRJ algorithm. Because the dataset files in SRJ algorithm are distributed to each node by file transmission, the memory cost of SRJ algorithm is small, but its disk I/O operations are the most and the read-write speed is slow. PCJ algorithm will load all the dataset into the memory processing, will occupy a lot of memory space, prone to node memory overflow. The SPCJ algorithm only loads the skewed data in the dataset into the memory, which reduces the memory overhead of the algorithm. Especially, the larger the dataset size is, the more significant the memory consumption of SPCJ algorithm is compared with PCJ.

### 6. Conclusion

Data skew is common in massive data analysis and processing. This paper studies the join query algorithm of two large datasets under the data skew scene. On the basis of replication join algorithm, massive datasets are processed by data sampling statistics method, and the datasets are divided into skewed data subset and non skewed data subset. In order to improve the performance of join query of dataset, this algorithm processes the skewed data subset and non skewed data subset respectively. Experimental results show that the proposed replication join optimization algorithm based on sampling partition has obvious speed advantage compared with the other two algorithms in join query of two large datasets.

### Acknowledgments

Thanks to the China Railway Eryuan Engineering Group.Co.Ltd for providing research project support, as well as colleagues from the Academy of Science and Technology for their encouragement and help.
References

[1] Ya Zhou,XiaFei Wei,Han Xiong ,CaiLin Hu ,Ling Li. (2018) CSPRJ: MapReduce Join Query Algorithm based on Data Skew. J. Mini-Micro Systems.,39:367-371.

[2] Guipeng Liu,Xiaomin Zhu,Ji Wang,Deke Guo,Weidong Bao,Hui Guo. (2018) SP-Partitioner: A novel partition method to handle intermediate data skew in spark streaming.J.Future Generation Computer Systems.,86: 1054-1063.

[3] M. Al Hajj Hassan,M. Bamha,F. Loulergue. (2014) Handling Data-skew Effects in Join Operations Using MapReduce.J.Procedia Computer ence.,29:145-158.

[4] Wenxia Guo,Chaojie Huang,Wenhong Tian. (2020) Handling data skew at reduce stage in Spark by ReducePartition.J. Concurrency and Computation: Practice and Experience.,32:1-19.

[5] Joanna Berlińska,Maciej Drozdowski. (2018) Comparing load-balancing algorithms for MapReduce under Zipfian data skews. J. Parallel Computing.,72:14-28.

[6] Zhuo Wang,Qun Chen,Bo Suo,Wei Pan,Zhanhuai Li. (2019) Reducing partition skew on MapReduce: an incremental allocation approach.J. Frontiers of Computer Science.,13: 960 – 975.

[7] Mohammad Amin Irandoost,Amir Masoud Rahmani,Saeed Setayeshi. (2019) A novel algorithm for handling reducer side data skew in MapReduce based on a learning automata game.J.Information Sciences.,501:662-679.

[8] Elaheh Gavagsaz,Ali Rezaee,Hamid Haj Seyyed Javadi. (2019) Load balancing in join algorithms for skewed data in MapReduce systems.J. The Journal of Supercomputing.,75: 228–254.

[9] Foto Afrati,Nikos Stasinopoulos,Jeffrey D. Ullman,Angelos Vassilakopoulos. (2018) SharesSkew: An Algorithm to Handle Skew for Joins in MapReduce.J.Information Systems.,77:129-150.

[10] Hon Hwang,Louise Ryan. (2020) Statistical strategies for the analysis of massive datasets.J. Biometrical Journal.,62:270-281.

[11] Leenen Lies,Vergote Valentijn,Vanmechelen Bert,Tersago Katrien,Baele Guy,Leem Philippe,Leirs Herwig,Dellicour Simon,Vrancken Bram,Maes Piet. (2019) Identifying the patterns and drivers of Puumala hantavirus enzootic dynamics using reservoir sampling.J. Virus evolution.,5:132-141.

[12] Lei Yu,Xiao-fang Zhao,Yan Jin,Heng-yi Cai,Bo Wei,Bin Hu. (2019) Low powered blockchain consensus protocols based on consistent hash.J. Frontiers of Information Technology & Electronic Engineering.,20:1361-1378.

[13] J. Vinothkumar. (2018) Decoupling the Ethernet from Systems in Consistent Hashing.J. Networking and Communication Engineering.,10:49-51.

[14] Xiaofeng Ding,Wanlu Yang,Kim-Kwang Raymond Choo,Xiaoli Wang,Hai Jin. (2019) Privacy preserving similarity joins using MapReduce.J. Information Sciences.,493:20-33.

[15] Mohammad Sultan Mahmud,Joshua Zhexue Huang,Salman Salloum,Tamer Z.Emara,Kuanishbay Sadatiyinov.(2020) A Survey of Data Partitioning and Sampling Methods to Support Big Data Analysis.J.Big Data Mining and Analytics.,3:85-101.