A strategy to load balancing for non-connectivity MapReduce job

Huaping Zhou, Guangzong Liu, Haixia Gui
College of Computer Science and Engineering, Anhui University of Science and Technology, Huainan 232000, Anhui, China

Abstract. MapReduce has been widely used in large scale and complex datasets as a kind of distributed programming model. Original Hash partitioning function in MapReduce often results the problem of data skew when data distribution is uneven. To solve the imbalance of data partitioning, we propose a strategy to change the remaining partitioning index when data is skewed. In Map phase, we count the amount of data which will be distributed to each reducer, then Job Tracker monitor the global partitioning information and dynamically modify the original partitioning function according to the data skew model, so the Partitioner can change the index of these partitioning which will cause data skew to the other reducer that has less load in the next partitioning process, and can eventually balance the load of each node. Finally, we experimentally compare our method with existing methods on both synthetic and real datasets, the experimental results show our strategy can solve the problem of data skew with better stability and efficiency than Hash method and Sampling method for non-connectivity MapReduce task.

1 Introduction

With the popularity of mobile Internet, networking and other information technology, people's lives become more convenient and the amount of data generate at TB level every day, the traditional centralized data processing has been unable to adapt to the growth of data. The distributed processing model proposed by Google, MapReduce, shows good parallelism and scalability in massive data processing[1]. The whole distributed computing process is divided into two stages map and reduce, it hides information communication and data transmission among nodes, users can process data in the cluster only need to complete the design and implementation of the Map and Reduce functions. MapReduce has been built based on the HDFS which has high fault tolerance in Apache’s open source project Hadoop, and it has been widely used and rapidly developed into the leading analysis platform in big data processing[2].

In the parallel processing of MapReduce tasks’ completion time is determined by the slowest Reduce, and the data skew is an important factor that affects the performance of the task. The existing partitioning methods can be roughly divided into two categories: one is to avoid data skew by preprocessing; the other repartitions the data before data skew appeared.

In literature[6], a first sampling method is proposed to set the equilibrium partitioning function, then users partitioning the data according to the distribution. The method obtains the approximate distribution of the data by sampling, so MapReduce can balance the data partitioning according to the sampling results. Gufler[7,8] has proposed a partitioning method based on sampling, this method increases the sampling function in Map phase, so it can reduce consumption of system communication resource. When stage Map is completed after a certain proportion, users can split and merge data to
balance allocation of partitioning according to the sampling results. Jiashuai Zhou\cite{9} has saved the intermediate data when mapper is running, so the number of reducer can be determined by analyzing the sampling results. But the method begins to transmit intermediate data after stage Map is completed, the data transfer times increase more than Hadoop’s method transmit while processing. An incremental partitioning strategy was proposed in\cite{10}, MapReduce reduces the partitioning size when overflow writing appeared in the Map phase, until total micro data partitioning can be the number of reducer integer times, then these micro data partitioning can be partitioned into nodes even according to the greedy algorithm, but this method cannot get good result when some key has a substantial ratio. Hangchen Li\cite{11} has calculated the pressure of each reduce nodes in stage shuffle, then system can schedule data of heavy load nodes to other nodes according to the data distribution, so the cluster’s load can be balanced without requiring users to provide extra input. Kwon\cite{12} has proposed a method to balance the data of each node in the process of reducer operation. This method establish the cost model of remaining data processing for all reducer, then those reducers which have not completed will transmit a certain proportion of the remaining data to the other reducers according to this cost model, so that the data or task of skew node can be distributed to other nodes evenly, the load of cluster can be balanced on MapReduce. All of the above two methods make reducers retransmit the data of large load nodes to the other nodes after they get data partitioning, however, they add extra data transmission costs in the process of reducer operation.

2 Data Skew of MapReduce
This section first introduces how the data skew generated on the MapReduce, and then describes the specific process to load balancing through the index shift.

2.1 Data Skew on MapReduce
The running process of distributed computing model MapReduce can be divided into two stages: Map and Reduce.

Stage Map: Hadoop system divides the input file according to the block size setted by users, then it starts a corresponding number of mapper through one-to-one or many-to-one way to obtain data partitioning; the mapper convert the data to <key₁, value₁> according to the logic defined in stage Map, then mapping it to <key₂, value₂>, another group of key values; later keys’ value are calculated by Hash the default system partitioning function, and the array which have the same value will be partitioned into the same reducer to do related calculation.

Stage Reduce: Each reducers obtain their corresponding partitioning from the mapper through the Hyper Text Transfer Protocol, then they operate the Reduce() function on these data partitioning in there own nodes, and write the final result of the operation to respective output file at last.

Cause the overall task completion time depends on the slowest node on MapReduce, data skew in Reduce will extend the execution time of individual reducer, so data skew on MapReduce will delay the overall execution time of cluster. Fig.1 is data processing procedure of MapReduce frame.

![Fig. 1 Data processing procedure of MapReduce frame.](image)

The reasons of the data skew can be divided into two kinds: first, the data quantity which divides into the nodes is not balanced; the two is the difference of processing cost of the node to the
intermediate results\cite{3}. The processing cost has more effect in the connection calculation, and this strategy mainly consider the data distribution of non-connectivity problems in calculation. When data skew appeared, the node’s execution time which get more data is higher than average, while other nodes must remain idle to wait skew nodes complete their tasks, cause the extension of execution time, and reduce the efficiency of cluster. literature\cite{4} shows 92% tasks appeared data skew in Reduce, and the running time of these reducers is higher than the default Hash method about 22% to 38% through a lot of experiments on the Hadoop platform.

In view of the above problems, we proposed a method based on index shift. The method calculates the amount of data transmitted to each reducer in the process of Mapper, and then the data is collected to be managed and monitored by Job Tracker. When there is a Reduce’s amount of data achieved from Map exceed a certain proportion of the total data, the Partitioner changes the index of data partitioning that will be sent to skew reducer to rest Reduce, and each node’s data achieves even distribution ultimately.

2.2 Index Shift to Eliminate Data Skew on MapReduce Tasks
In the MapReduce operations, each mapper increase R partitioning counters to record the results of the mapper partitioning into each reducer’ records. The value of mapper partitioning counter will be sent to the Job Tracker through heartbeat mechanism, then Job Tracker will collect these values to find skew reducer partitioning through the global information and skew model, next it selects the minimum load Reduce from all non skew reducer to receive the shift partitioning, and calculates the difference between skew reducer’s index and shift reducer’s. Job Tracker will send shift value to the Partitioner through the heartbeat to modify the partitioning function dynamically, next all the data that would be sent to skew reducer will be partitioned into the lighter load Reduce by adding an corresponding shift. Job Tracker continue to the statistical partitioning information and deal with the skew data after a round of shift operation, finally, all the reducers in this work get the average load until the dataset is completed in stage Map. Figure 2 is operation process for index shift strategy to solve the data skew in the MapReduce:

![Fig. 2 Data balancing in MapReduce based on index shift.](image)

In this paper, our partitioning method no need to obtain the key value proportion before inputting data, but the same keys’ values are sent to a different reducer due to these partitioning’ index shift in the process of load balance, so the results of operation should be merged in the Reduce task.

3 Cost Model and Specific Algorithm of Index Shift Strategy
In this section, we first introduce the three important models of index shift, and then give the concrete realization and the cost evaluation of the algorithm.

3.1 Cost Model of Index Shift Strategy
The original partitioning function is: \( f(\text{key}) = \text{Hash}(\text{key}) \mod R \), \( R \) is the number of reducer; partitioning function after modifying: \( f'(\text{key}) = f(\text{key}) + P(\text{key}) \cdot P_{\text{f(\text{key})}} \) is shift value of next
partitioning which will be sent to skew reducer, All the initial shift values are 0, they change along the skew partitioning, \( f(\text{key}) = \{ 0, 1, \ldots, R-1 \} \).

**Global partitioning** is the sum of M mappers’ partitioning information which is obtained by adding a counting method \( \text{Counter}() \) into Map() function, this counting method contains an array of counters \{\( \text{count}_0 \), \( \text{count}_1 \), \ldots, \( \text{count}_{R-1} \)\} to count the number of records for R reducers’ partitioning. Dynamic partitioning method calculates the output of mapper \(<\text{key}, \text{value}>\) with shift results by Hash partitioning, which reducer partitioning the results belong to, the counter of this reducer plus 1.

\[
\text{count}_i = \sum_{f(\text{key})=i} (\text{key}, \text{value})
\]

\( i =\{ 0, 1, \ldots, R-1 \} \), \( CR_i \) is total number records of each reducer,

\[
CR_i = \sum_{i=0}^{M} \text{count}_i
\]

\( M \) is the number of mapper, all mapper will send these counter information to Job Tracker through heartbeat mechanism, then Job Tracker collects the number of records which are included in each reducer, these global records are the basic conditions to find data skew.

**Skew model** is established based on global partitioning, if \( CR_i \) the number of some reducer’s partitioning record exceed \( 1/R \) of SC the total number of all reducers’ records before the stage Map completed, this reducer is skew reducer. Skew reducer \( R_i \) should be added into the skew reducer set \( R_S \), we need only to consider the skew reducer not contain in reducer set \( R_S \) in the next skew judgment, the initial \( R_S \) is empty.

\[
CR_i \geq \frac{S_C}{R}, R_i \notin R_S
\]

But we can only get the total amount of input data obtained in the \( S_D \) operation before the end of Map phase, and a built-in counter of original MapReduce framework contains the statistics of processed bytes for all mapper \( O_D \) and output record number \( O_C \), the built-in counter is maintained by the Job Tracker. In the operation of task, multiple mapper process input data slice together, the processed data can be regarded as a random sampling, namely there is a certain proportion between the overall number of records and processed data records, so the total number of records need to be processed by all reducers is:

\[
S_C = \frac{S_D \times O_C}{O_D}
\]

Job Tracker will recalculate the total number of records \( SC' \) when processing a certain proportion of data,

\[
S_C' = \max\{S_C, S_C'\}
\]

In the operation of Map tasks, with the data processing increased, the estimated overall records’ amount tends to close to the actual value. In the data skew model, we began to judge data skew when the amount of data processing has been more than the \( 1/R \) of total amount of data, because all Reduce are not skewed before exceeding this limit, and the solution will have a greater error according to the proportional relationship when processed data is too little.

**3.2 Algorithm Implementation and Analysis of Index Shift Strategy**

**Algorithm 1:** Partitioning dynamically by shift index

**Input:** Summary of all mappers’ data record in each partitioning

**Output:** Shift values of reducers’ partitioning

1. WHILE \(((\text{OutData}/\text{SumData})>1/R)\&\&!(\text{Map.Complete})\) /*Processed data in Map has exceeded 1/R of total data and Stage Map was not completed*/
2. FOR \( i=0 \) TO \( R-1 \) /*R reducers*/
3. IF !ReduceSkewSet.Contain\{Reduce\_i\}; /*Do not consider the skew reducer which Has
processed */
4.  \( \text{CountRecord}_i = \text{Sum}\{ \text{Reduce}_i\text{.AllValue} \} \); /*The total amount of data records of reducer \( n \)*/
5.  \( \text{SumCount}=\max\{ \text{SumCount}, \text{SumData}\times\text{OutCount} / \text{OutData} \} \); /*The total number of records of this task*/
6.  IF \( \text{CountRecord}_i \geq \text{SumCount} / \text{R} \) /* Skew reducer appeared*/
7.  \( \text{ReduceSkew}\text{.Add}\{ \text{Reduce}_i \} \); /*Adding skew reducer into skew set*/
8.  \( \text{MinLoad}=\text{Min}\{ \text{Reduce}\text{.Except}\{ \text{ReduceSkewSet} \} \} \); /*Finding the minimum load node in non skew reducer as shift reducer*/
9.  \( \text{SkewShift}=\text{Reduce}_i\text{.Index} - \text{MinLoad}\text{.Index} \); /*Shift value of skew reducer*/
10. FOR \( \text{k}=0 \) TO \( \text{R}-1 \)
11. IF \( \text{Reduce}_k\text{.ShiftSet}\text{.Contain}\{ \text{Reduce}_i \} \) /*Finding those skew reducers which contain \( \text{R}_i \) in shift reducer set*/
12. \( \text{Shift}_{\text{k}}+=\text{SkewShift} \); /*Changing all those shift values which have pointed to these skew reducers*/
13. \( \text{Reduce}_k\text{.ShiftSet}\text{.Add}\{ \text{MinLoad} \} \); /* Adding this shift reducer into the shift set of these skew reducers */
14. ENDIF
15. ENDFOR
16. ENDIF
17. ENDFOR
18. ENDFOR
19. RETURN \( \text{Shift} \); /*Output the array of index shift values*/
20. ENDFOR

In the algorithm, cycle of lines 2 to 18 is the core part of the index shift strategy, we do not consider skew reducer have been processed in each operation cycle from the third row, so the process of data skew will be executed \( \text{R}-1 \) times at most. And in lines 4 to 6, Job Tracker can find skew reducer from the global Reduce partitioning information, total amount of data and the processed data records, thereinto, global partitioning is the sum of M mappers’ statistical information; In lines 7 to 9, this strategy calculates the shift value between skew reducer’s index and the reducer’s which has minimum load, the minimum load node can be find by traversing all reducers; It updates these index shift values of existing skew reducer in the cycle of lines 10 to 15, at most it will change \( \text{R}-1 \) reducers’ index. So, the time complexity of index shift strategy is \( \text{O} (\text{MR}) \), that means it will be effected by the number of mapper and reducer, furthermore, \( \text{M} \times \text{R} \) generally. This strategy no need to maintain statistical information for each key, it need only for each reducer, so it can greatly reduce the consumption of memory and other resources.

4 Experimental Results and Analysis
Our Hadoop cluster is consists of 4 nodes, including 1 Master nodes, 3 Slave nodes, all of there nodes are connected by fast network. Each node has the same configuration: a 500 GB hard disk memory, a 4 GB memory, a processor with dual core Inter (R) Pentium (R) CPU@3.4GHz, the operating system is Ubuntu13.0. We installed the version Hadoop 2.0.0, and compile source code with Java JDK1.7.

The experimental dataset includes two categories, real datasets and synthetic datasets. The synthetic dataset is consist of 200M positive integer from 1 to 1000 which satisfies the standard Zipf distribution, the proportion of the kth elements is \( k^{-\alpha} / \sum_{i=1}^{N} i^{\alpha} \), \( N \) is the total number of elements, \( \alpha \) is the degree of data skew, the larger the \( \alpha \), the more skew the dataset, the dataset satisfies even distribution standardly when its \( \alpha \) is equal to 0;

4.1 Performance of loading balancing
This experiment compares index shift method IS with Hadoop default method Hash and cluster
segment-combine method CSC[15] at performance of processing data by the WordCount program based on the standard Zipf datasets. The Hash partitioning method sends these keys which have the same value that is calculated by hash to the same reducer; CSC will get the sampling of general data distribution firstly, then segment those large partitioning to small part, and combine them to the reducer according to the greedy algorithm; IS method modifies the partitioning function dynamically according to the processed records and data in the Map phase, so the data will be assigned to each reducer equilibrium.

In figure 3, we compare the execution time of these three methods at different skew degree, the reducer number is set to 6, the horizontal axis represents skew value $\alpha$ of dataset in the map, the vertical axis represents the running time of task. When the data distribution is completely even, means $\alpha = 0$, Hash method can deal this job with the shortest time, because other two methods increase some processing for data skew; When the data distribution is generally even, means the skew degree $\alpha$ is lower than 0.5, the execution time is little difference among these three methods; And when the data distribution trends to uneven, means $\alpha$ is more than 0.5, with the increment of $\alpha$, CSC and IS can significantly reduced the execution time relative to Hash method. To Hash method, with the degree of data skew increases, the overall running time grow rapidly, it means data skew have great affect on Hash; But data skew have less influence on execution time of CSC and IS method. And in each value of $\alpha$, IS’s execution time is less than CSC’s, because IS reduce a round of statistical operation than CSC. We can see that IS method has shown a better balance effect for different skew degree of dataset.

![Fig. 3 Running time contrast tests on standard Zipf datasets.](image1)

### 4.2 Running time for MapReduce clustering jobs

This experiment adopts the K-means++ clustering algorithm based on MapReduce to cluster and analyze these 10010 micro-blogs. In order to guarantee the single variable, K of all the experiments in this paper is fixed, and we select the same initial clustering center to ensure that the round of cluster and distribution of each round is same to other experiment. In order to obtain more exact results in the distributed cluster, we increased the real data which is pretreated 100 times than before.

![Fig. 4 Running time on MapReduce for each round.](image2)
Figure 4 compares program’s execution time of three methods in each stage of clustering iteration based on MapReduce, reducer number is set to 6 in this experiment, the horizontal axis represents the round of graph clustering iterative, the vertical axis represents the running time of these jobs. To Hash method, execution time of each round has more fluctuations than the other two methods, because the different result of each clustering round change the skew degree of nodes’ load; the average execution time for each round of CSC method is lower than the Hash method’s, and each round of execution time is stable, but the relative increase in the sampling and merge operation; The execution time for each round of IS method is less than CSC’s, because it omits sampling and statistic in the procedure of processing data, and it only increases an merging at the end of cluster, its efficiency is higher than that of CSC method.

Figure 5 mainly expresses the impact of reducer’s number which is setted by user to the execution time of overall MapReduce clustering while total processing resources are not changed. With the change of reducer’s number, Hash method’s processing time shows greater volatility, the overall execution time extend especially the settings of reducer’s number is unreasonable, it reduces the efficiency of whole cluster because the maximum load node has serious data skew; CSC method’s average running time is less than Hash’s in different settings of reducer’s number, but in some cases, its running time is higher than the Hash partitioning method, because it increases consumption of sampling, its efficiency is lower than that of the Hash method when the setting of reducer’s number is reasonable; To IS method, the number of reducer defined by user affect overall execution time little, and the overall execution time is better than the CSC method, it is more close to Hash partitioning which is even, shows a better load balancing effect.

5 Conclusion
We proposed an index shift strategy to solve the data skew problem of non-connectivity MapReduce task, this method no need to know the distribution of the data, but it adjusts the partitioning dynamically in the stage of Map according to the partitioned data by Job Tracker to achieve load balancing on Reduce, and verifies the validity and efficiency of this method in the standard Zipf and real datasets.

Acknowledgements
This work was supported by natural science foundation of China (51174257), project of safety mangement research center of mining enterprises of Anhui university of science and technology (SK2015A084) and Anhui provincial department of education outstanding young talent fund.

References
[1] Dean J,Ghemawat S. MapReduce:simplified data processing on large clusters[J]. Communication of the ACM,2008,51(1):107-113
[2] Wang Shan, Wang Hui-ju , Qin Xiong-pai al. Big data: Challenges, studies and forecasts. Chinese Journal of Computers, 2011, 34(10): 1741-1752

[3] Zhang Chang-chun. Design and Optimization of Big Data Connection Algorithm Base on MapReduce[D]. University of Science and Technology of China, 2014.

[4] Lin J, et al. The curse of Zipf and limits to parallelization: A look at the stragglers problem in MapReduce/Proceedings of the 7th Workshop on Large-Scale Distributed Systems for Information Retrieval. Boston, USA, 2009

[5] Geng Yu-Jiao. The Research of Skew with Counter Technique in MapReduce[M. S, dissertation]. Dalian Maritime University, Dalian, 2013.

[6] Racha S C. Load Balancing Map-Reduce Communications for Efficient Executions of Applications in a Cloud[M. S, dissertation]. Indian Institute of Science, Bangalore, India, 2012.

[7] Gufler B, Augsten N, Reiser A, Kemper A. Load balancing in MapReduce based on scalable cardinality estimates //Proceeding of the 2012 IEEE 28th International Conference on Data Engineering (ICDE), Washington, USA, 2012: 522-533

[8] Gufler B, Augsten N, Reiser A, Kemper A. Handling data skew in MapReduce //Proceeding of 1th International Conference on Cloud Computing and Service Science. Noordwijkerhout, the Netherlands, 2011, 146: 574-583

[9] Zhou Jia-Shuai, Wang Qi, Uao Jun. An approach for load balancing Computer in MapReduce via dynamic partitioning. Journal of Computer Research and Development, 2013, 50(Suppl.): 369-377

[10] Wang Zhuo, Chen qun, Zhanhuai Li al. An Incremental Partitioning Strategy for Data Balance on MapReduce[J]. Chinese Journal of Computers, 2016, 39(01): 19-35.

[11] Li Hang-chen, Qin xiao-lin, Shen xiao. Pressure Feedback Strategy to Load Balancing on MapReduce [J]. Journal of Computer Science, 2015, 42(4): 141-146

[12] Kwon Y, Balazinska M, Howe B, Rolia J. Skew Tune: Mitigating skew in MapReduce applications//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. Scottsdale, USA, 2012: 25-36

[13] Li Qiu-hong. Large-scale Data Mining Based on MapReduce[D]. Fudan University, 2013.

[14] Fu jie, Du Zhi-hui. An Periodic Strategy to Load Balancing on MapReduce Job. Journal of Computer Science, 2013, 40(3): 38-40

[15] Xu Yu-jie. Parallel Cluster Algorithm for Mass Data Based on Cloud Computing[D]. Dalian Maritime University, 2014.