Load Balancing Mechanism Based on Linear Regression Partition Prediction in Spark

Zichun Huang, Wenguo Wei* and Guiyuan Xie
Guangdong Polytechnic Normal University, Faculty of Electronics and Information, 293 Zhongshan Avenue West, Tianhe District, Guangzhou.
Email: 980599602@qq.com

Abstract. With the rise of cloud computing, the internet of things, social networks and other technologies, information data is expanding rapidly, and traditional processing and storage systems have been difficult to deal with massive data. Spark is a fast and efficient MapReduce implementation developed after Hadoop. However, the shuffle operation in spark will cause the data set on some working nodes to be too large, while other nodes may be idle, which will affect the performance of spark jobs. This phenomenon is called data skew. Aiming at the problem of data skew in spark platform, this paper proposes a load balancing mechanism based on linear regression partition prediction: SP-LRP (Spark load balancing mechanism based on Linear Regression Partition). SP-LRP predicts the partition size of the reduce tasks at run-time, leverages the skew detection algorithm to identify the skew partition and adjusts the task resource allocation according to the fine-grained resource allocation algorithm. We use the benchmark dataset to evaluate SP-LRP, and compare the average execution time of two algorithms and the load balancing degree of reducers in different situations. The experimental confirmed the efficiency of SP-LRP in their respective usage scenarios.

1. Introduction
With the rapid development of mobile Internet and social networks, the data scale is growing explosively. Nowadays, various e-commerce platforms will generate a large number of hot spot data, which brings about the problem of unbalanced data distribution, that is, data skew. People have entered the era of big data, and the processing of big data is a hot spot of current research and application. Because the normal data distribution theoretically skewed, which is what we call the 20-80 principle: 80% of the wealth is concentrated in the hands of 20% of the people, 80% of the users only use 20% of the functions, 20% of users contribute 80% of hits, large amount of data leads to unbalanced data partition, resulting in data skew. Spark is widely used in search engine, database operation, machine learning, scientific computing and other fields due to its simplicity[1].

Compared with Hadoop[2] and other distributed computing frameworks, Apache Spark[3] provides a more effective implementation mechanism for large-scale data processing. Spark, as a distributed data processing framework, uses the memory computing method to introduce the concept of resilient distributed datasets (RDD)[15], load the data into memory, debase the access delay of data exchange, and achieve the ability of quasi real-time analysis of large datasets. Although Spark's RDD does not divide the job display area into map and reduce tasks like MapReduce[19], Spark also has a stage similar to the shuffle stage in MapReduce, and uses some of MapReduce's terms. The tasks before shuffle are called map tasks, and the tasks after shuffle are called reduce tasks. The main reason for data skew in Spark is its shuffle mechanism. As shown in Figure 1, because Spark has the shuffle mechanism, in the case of data skew, the shuffle operation pulls a large number of data with
the same key in different nodes to the same node to perform the reduce operation. A Spark task can only be executed in one partition, so the task running time of some key with huge data volume will be very slow. At present, the main idea of research on Spark task optimization is to avoid shuffle operation and kill data skew in the cradle. The other is to optimize resource allocation in task scheduling.

Figure 1. Map stage and Reduce stage in spark.

In order to solve the problem of performance and running time, this paper proposes a load balancing mechanism based on linear regression partition prediction, which can predict the partition size of each map task before the completion of map task, improve the load balancing of reduce task, and realize the payload balancing under Spark platform.

The rest of this paper is organized as follows: Section 2 investigates the related work of data skew in distributed processing environment; section 3 introduces a load balancing mechanism based on linear regression partition prediction; section 4 presents experiment results. The conclusion is in Section 5.

2. Related Works

Nowadays, the framework used to deal with a large number of data is based on MapReduce. The academic community has a more in-depth research and mining on all aspects of it. Data skew is one of the biggest threats to the performance of this system. The concept of Spark appeared in 2012, and the academic research on it is relatively limited. Therefore, to some extent, we can learn from the relevant research and methods in MapReduce to deal with data skew. When solving the data skew problem in map reduce, in order to optimize the performance of MapReduce framework, in recent years, people have conducted a lot of research on the data skew problem in MapReduce, there are also in-depth studies on data skew in distributed databases [20, 21]. The solution mainly includes the following aspects: solving the data skew in map stage; improving the data skew in reduce stage, solving the data skew in map stage and reduce stage.

On the map stage, Jiong et al.[4] proposed a mechanism to distribute load in heterogeneous environments according to the capacity of machines. The purpose of this mechanism is to enable map nodes to complete their tasks at the same time. The specific implementation of this mechanism is that when processing files of the same size, the machines in the cluster first sort according to their response time, and then allocate data blocks according to the capacity of the machines on HDFS. After data initialization, if new data is added or deleted, or nodes are removed from the cluster, or new nodes are added to the cluster, the data will be reallocated among nodes according to their computing power. Wang[18] proposed an incremental data allocation approach to reduce partition skew among reducers on MapReduce. The proposed approach divides mapped data into many micro-
partitions and gradually gathers the statistics on their sizes in the process of mapping. The micropartitions are then incrementally allocated to reducers in multiple rounds. Guo et al.[5] put forward a strategy to shorten the execution time of MapReduce task, which can shorten the execution time of map phase by better using mapper. In this strategy, if the number of map tasks is less than the number of mappers, many map tasks will be split into several smaller tasks so that all mappers can be used for processing. In another strategy, if the number of map tasks is greater than mapper, multiple map tasks are merged. During execution, if some map tasks last longer than others, the released mapper can be used to process the remaining data to overcome the calculation deviation.

At the reduce stage, Gufler B and others[6] proposed a distributed monitoring system top cluster for capturing data skew in MapReduce system. They solved the problem of estimating the task cost assigned to the reducer according to the given cost model. Xu et al.[7] Proposed two algorithms to overcome the data partition skew of reduce side. In both algorithms, they use a separate MapReduce job running before the original job, whose task is to estimate the frequency of the cluster, and the sampling method is random sampling with uniform distribution. Tang et al.[8] Proposed a segmentation and composition algorithm of skew intermediate data block (SCID), which can improve the load balance of various reduce tasks. Once a data cluster exceeds the remaining capacity of the current bucket, it will be split. SR Ramakrishnan et al.[9] Proposed a static load balancing algorithm, which evenly distributes the work on the reduce of MapReduce jobs, thus significantly reducing the running time. Chen et al.[10] introduced a strategy called MRSIM (reducing reducer skew in MapReduce). According to the strategy, two algorithms are designed: MRSIM-LB and MRSIM-LBF. In the two algorithms in the shuffle phase, the load size on each reducer is recorded by the reducer load monitor, which allows calculating the average load on the reducer. If in the unordered broadcast stage, the reducer will unordered play all part of its data among the remaining reducers, and the unordered play of the reducer has the maximum load stop, then divide the remaining load of the reducer into two equal parts, and distribute between the two reducers.

When dealing with data skew in the reduce and map side, Dhawalia et al.[11] proposed a method to overcome the calculation skew in the map phase and the partition skew in the reduce phase. Ahmad et al. Proposed three methods to optimize the operation of map reduce jobs on heterogeneous clusters: CALB, CAS and PLB. Kwon et al. [12] Introduced skewtune. If the remaining time of the task is more than 1 minute and there are idle nodes in the cluster, the system will dynamically partition the unprocessed data of the task. The scope partition is already used to distribute unprocessed data. Zhang et al.[13] Considered the impact of heterogeneous environment on MapReduce operation. An algorithm named mrheter (to improve the performance of MapReduce in heterogeneous environment) is proposed. The algorithm divides the MapReduce implementation process into two sub phases: mapshuffle and reduce. In this paper, assuming that there is no skew in the data, the reducer is only on the heterogeneous machine. The algorithm tries to load data according to the capacity of the reducer. In the mapping phase, data is provided for each mapper based on its capabilities.

**3. SP-LRP General Framework**

In this section, we propose a load balancing mechanism based on linear regression partition prediction, which includes three modules: partition monitor, partition size predictor and resource broker. First, the operation information is counted according to the partition monitor, then the intermediate operation information partition is predicted according to the partition size predictor, and finally the dynamic resource adjustment is carried out by using the resource scheduler.

**3.1. SP-LRP Summary**

The spark job execution process based on SP-LRP is shown in the following figure, with the basic steps as follows:

- After starting the map task, the partition monitor in SP-LRP starts to analyze the operation statistics, such as the percentage of partition completed by the map task and the sum of sub partitions generated by the completed map task for reduce task, and the amount of intermediate data generated.
After getting the operation statistics, the size of each sub partition after completing the 100% map task is calculated by the partition predictor, and the current maximum partition is obtained by sorting.

- After predicting the size of each partition, the skew detection algorithm is used to identify the skew partition.
- Shuffle pulls and merges data from mapper nodes. When the resource broker recognizes the skew partition, it arranges the data in the partition in descending order and dynamically adjust the partition.

**Figure 2.** SP-LRP frame diagram.

### 3.2. Partition Monitor

Partition monitor is an important component of SP-LRP. Each worker node sends heartbeat information to master regularly to ensure its availability and update the status of running tasks of a given application. The existing research work focuses on analyzing the information of (key, value) pairs of intermediate results, and predicting the load of reduce tasks through sampling. This type of method will lead to a lot of overhead. Different from the existing research work, this paper extends the heartbeat mechanism during the map task operation.

Monitoring information includes operation information \( O \) and Intermediate data \( I_k \), which consists of the percentage of processed data in the total data set \( D_i \) and the total number of sub partitions \( X_j \) generated by reduce task \( j \), where \( D_i \) will change with the progress of spark job. Therefore, we use \( I \) to distinguish the data values measured at different times. \( X_j \) can be expressed in \( \sum_{i \in C_j} P_i \). \( C \) refers to the set of completed map tasks, and \( P \) represents the reduce task partition.

\[
U^{(j)} = \{O_1, O_2, \ldots O_j\}; \forall j \in [1, |U|]
\]  

The operation statistics \( U^{(j)} \) as in equation (1)’ of each partition \( j \) is a set of observation values O of a specific partition. Where \( U^{(j)} \) and \( K_j \) is tracked by the partition monitor, which can forward these statistics and heartbeat messages to the master.
3.3. Partition Size Predictor

The relationship between the two information in \( U^{(i)} \) is linear. When the map tasks is completed, \( X^i_j \) is the load of reduce task \( j \). Therefore, we use linear regression technology to determine the correlation coefficient between \( D^i \) and \( X^i_j \). For any reduce task \( j \in [1, M] \), its linear equation can be expressed by the following:

\[
X^i_j = \alpha_j + \beta_j \cdot D^i \quad l=1,2,...,k
\]  

(2)

Where, \( \alpha_j \) and \( \beta_j \) are correlation coefficients, and \( k \) is the total number of data tuples \( (D^i, X^i_j) \) measured. Once a map task is completed, a new tuple of training data can be generated. In order to control the size of training data set, we introduce a control factor \( \delta \). When the percentage of map tasks completed to the total number of map tasks exceeds \( \delta \) (for example, set to 5%), regression training stops immediately, and the correlation coefficient of the linear equation is determined by the least square method. When the correlation coefficients \( \alpha_j \) and \( \beta_j \) have been determined, it can be obtained \( X^i_j \) by substituting \( D^i = 100\% \) into the linear equation, which is the linear estimate of the load of reduce task \( j \).

Suppose map task \( i \) is the sub partition \( P_{i,j} \) generated by reduce task \( j \), \( N \) is the number of map tasks, and \( M \) is the number of reduce tasks. Then, the load of the reduce task \( j (j \in [1, M]) \) can be expressed in the following ways:

\[
\forall j, RL = \sum_{i=1}^{N} P_{i,j}
\]  

(3)

If at this time \( D^i > \delta \), then trigger the load estimation: for each reduce task \( j \), first determine the correlation coefficient \( (D^i, X^i_j) \) of its linear equation according to the collected training data set, then predict the load \( V_{RL}[j] \) of reduce task \( j \) according to the linear equation, and finally return the load vector \( V_{RL}[j] \) of all reduce tasks.

\[
\text{Max}\{V_{RL}[1], V_{RL}[2],...,V_{RL}[M]\}
\]  

(4)

Through this formula, we get the maximum load reducer node under the current state, which is recorded as \( \text{Max} (V_{RL}[M]) \), and then we use the skew detection algorithm to determine whether the current reduce node is overloaded, and finally feedback it to the resource manager for dynamic adjustment in real time.

Algorithm 1: reduce task load estimation algorithm based on linear regression

Given: Control factors of training data volume \( \delta \)

Figure Out: load vector \( V_{RL} \) of all reduce tasks \( j \in [1, M] \)

Initialization: \( l=0, F^l, S^l_j = 0, V_{RL} = \{0\} \)

1: When the master receives the heartbeat message from the worker:
2: for all reduce tasks \( j=1,2,...,M \) do
3: \hspace{1cm} Get the training data collected \( D^i, X^i_j = 0 \).
4: end for
5: if \( F > \delta \) then
6: \hspace{1cm} for all reduce \( j=1,2,...,M \) do
3.4. Data Skew Detection Algorithm

After getting the current maximum partition, in order to better measure the skew degree of data, this paper defines the uniformity degree of data distribution in data set, and puts forward the concept of data skew. The calculation of data inclination refers to the concept of average absolute deviation in classification statistics, counts the times of each key in a data set, and then calculates the average of the absolute value of the deviation of each observation value and arithmetic average value. At the same time, in order to regularize and standardize the results, we introduce the calculation method of relative mean absolute deviation, that is, divide the mean absolute deviation by the arithmetic mean, and finally the definition of data inclination takes the relative mean absolute deviation of one-half.

$$G = \frac{\sum_{a=1}^{n} \sum_{b=1}^{n} |x_a - x_b|}{2 \sum_{a=1}^{n} \sum_{b=1}^{n} x_b} = \frac{\sum_{a=1}^{n} \sum_{b=1}^{n} |x_a - x_b|}{2n \sum_{a=1}^{n} x_b}$$

(5)

G represents the data inclination, and a represents the number of times each key in the data set appears. The range of data inclination G is from 0 to 1. The closer G is to 1, the greater the data inclination is. The closer G is to 0, the average data distribution is.

3.5. Resource Broker

By predicting the partition size statistics, the waiting time in the reduce phase can be minimized. The main idea of resource scheduler is to distribute the identified skew data to the current least loaded reducer. The keys in the partition are sorted in descending order, with the largest key marked as 1 and the rest marked as 0. After the allocation of each cluster is completed, according to the current remaining capacity of all reducers, sort them in descending order again, and repeat the above cluster allocation process. The dynamic allocation of resource broker is shown in algorithm 2.

Algorithm 2: dynamic allocation of resource broker algorithm

Import: source dataset R
Output: Result
Initialization: $L=\phi$;
1. receive the load vector $V_{RL}$ passed by algorithm 1;
2. for each key=0 do
3. assign directly to the specified reducer node;
4. end for
5. Map();
6. While receive the new key do
7. If key=0 then
8. key assign directly to the specified reducer node;
9. else
10. $\text{Min}(V_{RL}[M]) = \text{Min}(V_{RL}[1], V_{RL}[2], V_{RL}[3], ..., V_{RL}[M])$;
4. Experiment and Result Analysis
This section mainly carries out WordCount and Tera-Sort benchmark tests on the linear regression prediction partitioning mechanism proposed in the third section, and analyzes the test results.

4.1. Description of Experimentation
In this experiment, four VM virtual machines are built, one of which is the master node of spark and the other three are the worker node; the operating system of each node is CentOS 7.2, and the experimental environment is based on spark 1.6.3 and Hadoop 2.6.5.

This experiment uses PUMA benchmark data[12] and divides the experimental data into two categories: small dataset of 6GB and large dataset of 20GB, respectively (table 1).

| Table 1. Benchmark dataset of different data volume sizes |
|-------------------------------|-------------------------------|-----------------|-----------------|
| Application                   | Dataset                       | Category        | Size(GB)        |
| Tera-Sort                     | Wikipedia                     | Small           | 6              |
|                               |                               | Large           | 20             |
| WordCount                     | Wikipedia                     | Small           | 6              |
|                               |                               | Large           | 20             |

Tera-Sort: Sort load is implemented using sortByKey operator.
WordCount: The wordcount application counts the number of occurrences of each word in a given file. The map function generates an intermediate tuple (word, 1). Reduce aggregates intermediate tuples from all map tasks and provides a combined output (key, value) pair. The implementation of the WordCount load changed from using the reduceByKey operator to the groupByKey operator, because reduceByKey does map-side local aggregation by default, the amount of data transferred to each Executor is small, and the skew is not obvious.

Application makespan and load balancing degree are often used as the classical metrics [8, 16, 17] to quantitatively measure the impact of Spark application. There are two main evaluation indexes of the experiment, one is the makespan of the application, the makespan is the total execution time of the benchmark test, that is, the total time consumed from the first task in the first stage of the first job to the last task in the last stage of the last job; the other is the load balancing degree of the cluster, reflecting load balancing degree of spark cluster system.

The experimental comparison algorithm is HashPartitioner algorithm and SCID algorithm[6]. HashPartitioner is the default method used by the spark platform. In this method, the hash value of the key in the middle data is directly used to get the partition ID of the key; SCID first uses the reservoir sampling algorithm to sample, then estimates the size of the data through sampling, identifies the data skew block, and then splits the data skew block to maintain the load balance of the reducer.

4.2. Experimental Result
We present next the results of applying the proposed SP-LRP on real Spark workloads so as to evaluate its effectiveness.

In order to address the above partition skew problem, we propose an SP-LRP based on linear regression partition to create containers. In Spark, they used default resource container of uniform specification irrespective the different partition size which can degrade the applications and the
clusters performance. In order to highlight our contribution, we made a comparative experiment on HashPartitioner, SCID and SP-LRP.

A comparison of smaller dataset shows that Word Count applications in HashPartitioner, SCID, and SP-LRP take up 218, 221, and 192 seconds; TeraSort applications take up 171, 133, and 112 seconds, respectively. SP-LRP completes tasks 20-30 seconds before SCID. SP-LRP is significantly better than HashPartitioner and SCID in the comparison chart of another large dataset.

In this experiment, the standard deviation ‘as in equation (6 and 7)’ of each reducer node load is used to measure the system load balance (LB). The smaller the LB, the more balanced the load of each reducer node. The larger the LB, the more uneven the load of the reducer node.

\[
\bar{L} = \frac{1}{M} \sum_{j=1}^{M} L_j
\]

(6)

\[
LB = \sqrt{\frac{1}{M} \sum_{j=1}^{M} (L_j - \bar{L})^2}
\]

(7)

A comparison chart of smaller datasets shows that the load balances of WordCount applications in HashPartitioner, SCID, and SP-LRP are 0.285, 0.263, and 0.251, respectively; TeraSort applications occupy 0.261, 0.249, and 0.243, respectively. The load balance of SP-LRP is significantly better than that of HashPartitioner and SCID.

5. Conclusion
This paper proposes a load balancing mechanism SP-LRP based on linear regression prediction partition to solve reduce data skew problem in spark platform, builds heterogeneous spark standalone cluster, compare SP-LRP with spark’s native partition algorithm HashPartitioner and SCID algorithm in WordCount and Tera-Sort benchmark tests, analysis of SP-LRP performance in different data size;
multiple tests verify that the algorithm proposed in this paper can effectively reduce the impact of data skew on the makespan of big data applications.

6. Acknowledgements
This work was supported in part by the Science and Technology Project of Guangzhou City (201802020019, 201806040010).

7. References
[1] J Ekanayake, S Pallickara, G Fox. Mapreduce for data intensive scientific analyses[C]. IEEE Fourth International Conference on eScience. IEEE, 2008:277-284(2008).
[2] Hadoop. https://hadoop.apache.org/ 2019.
[3] Spark. https://spark.apache.org/2019.
[4] Jiong, X., Shu, Y., et al. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In: 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and PhD Forum (IPDPSW), 19 – 23 April 2010, pp. 1 - 9 (2010).
[5] Guo, Z., Pierce, M., Fox, G., Zhou, M. Automatic task re-organization in MapReduce. In: Cluster Computing (CLUSTER), 2011 IEEE International Conference on 2011, pp. 335 – 343(2011).
[6] Gufler B., Augsten N., Reiser A., et al. Load Balancing in MapReduce Based on Scalable Cardinality Estimates. IEEE International Conference on Data Engineering, 2012:522-533(2012).
[7] Xu, Y., Zou, P., Qu, W., Li, Z., Li, K., Cui, X. Sampling-based partitioning in MapReduce for skewed data. In: 2012 Seventh ChinaGrid Annual Conference, 20 – 23 Sept. 2012, pp.1 – 8 (2012).
[8] Tang Z., Zhang X. An intermediate data placement algorithm for load balancing in Spark computing environment. Future Generation Computer Systems, 2016, 78:287-301(2016).
[9] S.R. Ramakrishnan, G. Swart. Balancing reducer skew in mapreduce workloads using progressive sampling. ACM Symposium on Cloud Computing, 2012, pp. 1 – 14(2012).
[10] Chen, L., Lu, W., Che, X., Xing, W., Wang, L., Yang, Y. MRSIM: mitigating reducer skew In MapReduce. In: 2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA), 27 – 29 March 2017, pp. 379 – 384 (2017).
[11] Dhawalia, P., Kailasam, S., Janakiram, D. Chisel: a resource savvy approach for handling skew in mapreduce applications. In: Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on 2013, pp. 652 – 660. IEEE(2013).
[12] Kwon, Y., Balazinska, M., Howe, B., Rolia, J. SkewTune: mitigating skew in mapreduce applications. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data,Scottsdale, Arizona, USA 2012, pp. 25 – 36. ACM, 2213840(2012).
[13] Zhang, X., Wu, Y., Zhao, C. MrHeter: improving MapReduce performance in heterogeneous environments. Clust. Comput. 19(4), 1691 – 1701 (2016).
[14] PUMA Datasets [Online]. Available: https://engineering.purdue.edu/~puma/datasets.htm(2016).
[15] Wiese J, Stuhl L, Kliewer N. Mathematical models and solution methods for optimal container terminal yard layouts[J]. OR Spectrum, 2010, 32(3):427—452(2010).
[16] Xu Y., Qu W., Li Z., et al. Balancing reducer workload for skewed data using sampling-based partitioning. Computers & Electrical Engineering, 2014, 40(2): 675-687.(2014).
[17] Gavagasz E., Rezaee A., Javadi H H S. Load balancing in reducers for skewed data in MapReduce systems by using scalable simple random sampling. Journal of Supercomputing, 2018(3):1-26.(2018).
[18] Wang, Zhuo, et al. "Reducing partition skew on MapReduce: an incremental allocation approach." Frontiers of Computer Science 13.5 2019: 960-975.(2019).
[19] Glushkova, Daria, Petar Jovanovic, and Alberto Abelló. "Mapreduce performance model for Hadoop 2. x." Information Systems 79 2019: 32-43.(2019).

[20] Phinney M, Lander S, Spencer M, et al. Cartesian Operations on Distributed Datasets Using Virtual Partitioning[C]. IEEE Second International Conference on Big Data Computing Service and Applications. 2016:1-9.(2016).

[21] Xu Y, Kostamaa P. Efficient Outer Join Data Skew Handling in Parallel DBMS[J]. Proceedings of the Vldb Endowment, 2009, 2(2):1390-1396.(2009).