SALA: A Skew-Avoiding and Locality-Aware Algorithm for MapReduce-Based Join

Ziyu Lin[1(✉)], Minxing Cai[1], Ziming Huang[1], and Yongxuan Lai[2]

[1] Department of Computer Science, Xiamen University, Xiamen, China
{ziyulin,caiminxing,zimінghuang}@xmu.edu.cn
[2] School of Software, Xiamen University, Xiamen, China
laiyx@xmu.edu.cn

Abstract. MapReduce is a parallel programming model, which is extensively used to process join operations for large-scale dataset. However, traditional MapReduce-based join is not efficient when handling skewed data, because it can lead to partitioning skew, which further results in longer response time of the whole join process. Additionally, some newly proposed methods usually involve large amounts of intermediate results over the network in the shuffle phase of Mapreduce-based join, which may consume a lot of time and cause performance degradation. Here a novel algorithm called SALA is proposed, which employs volume/locality-aware partitioning instead of hash partitioning for data distribution. Compared with other existing join algorithms, SALA has three typical advantages: (1) makes sure that the data is distributed to reducers evenly when the input datasets are skewed, (2) reduces the amount of intermediate results transferred across the network by utilizing data locality, and (3) does not make any modification of the MapReduce framework. The extensive experimental results show that SALA not only achieves better load balance but reduces network overhead, and therefore speeds up the whole join process significantly in the presence of data skew.

1 Introduction

MapReduce is an efficient programming model from Google for large-scale data processing in domains such as search engine, data mining and machine learning. MapReduce is extensively used to process the join operation for large-scale dataset, and various join algorithms have been proposed to implement join operation in MapReduce environment [3].

Traditional MapReduce-based join algorithms, however, are suffering performance degradation when handling skewed data , because they use hash partitioning to distribute data that can lead to partitioning skew. Partitioning skew will bring some problems. On one hand, join algorithms have to take longer time to deal with load imbalance caused by partitioning skew. On the other hand,

Supported by the Natural Science Foundation of China under Grant No. 61303004 and 61202012, and the Natural Science Foundation of Fujian Province under Grant No.2013J05099.

J. Li and Y. Sun (Eds.): WAIM 2015, LNCS 9098, pp. 311–323, 2015.
DOI: 10.1007/978-3-319-21042-1_25

large amounts of intermediate results have to be moved from mappers to reducers over the network, thus introducing extra network overhead. As a result, join processing upon skewed data consumes more time.

Some methods such as SAND [2] and LEEN [2], have been proposed to solve the problem of data skew in MapReduce-based join, which adopt partition schemes considering the key's frequency distribution. However, SAND does not take into account the network overhead. LEEN not only solves load imbalance but also reduces network transmission, however, it changes the internal implementation scheme of Hadoop and ignores the advantage of overlapping [1] between the shuffle and map phases.

To overcome the above deficiency, we proposed SALA (Skew-avoiding and Locality-aware) MapReduce-based join algorithm, which uses volume/locality-aware partitioning to distribute data and does not make any modification of the MapReduce framework. Our approach firstly obtains the distribution information of key's frequency and location through data sampling. Based on this distribution information, SALA is able to guarantee the uniform distribution of data even when skewed data exist, so as to effectively avoid partitioning skew. At the same time, SALA reduces the amount of data transferred over network by utilizing the data locality feature of MapReduce, i.e., assigning keys to the nodes on which most of the intermediate results are located. This significantly improves the efficiency of the whole join operation.

In summary, we make the following major contributions:

- A novel algorithm called SALA is proposed to handle skewed data in MapReduce-based join. It not only achieves better load balance but reduces shuffled data over the network, thus resulting in significantly performance improvement.
- Volume/locality-aware partitioning scheme is proposed to distribute data, which is easy to implement without any modification of the MapReduce framework.
- We carry out extensive experiments and the results show the efficiency of SALA in the presence of data skew.

The rest of this paper is organized as follows. Sect.2 briefly introduces MapReduce-based join, and then we investigate the problem of data skew in Sect.3. Sect.4 presents the detail of the SALA. Extensive experimental results are reported in Sect.5. Related work is reviewed in Sect.6. Finally, we conclude the paper and discuss our future work in Sect.7.

2 MapReduce-Based Join

MapReduce-based join algorithms can be classified into two categories: map-side join and reduce-side join. For map-side join, the smaller input dataset is placed on each mapper and join operation only needs to be executed in the map phase to get the final results. Instead, a reduce-side join is carried out on the reduce phase. First, the map function takes the input dataset from DFS(Distributed File

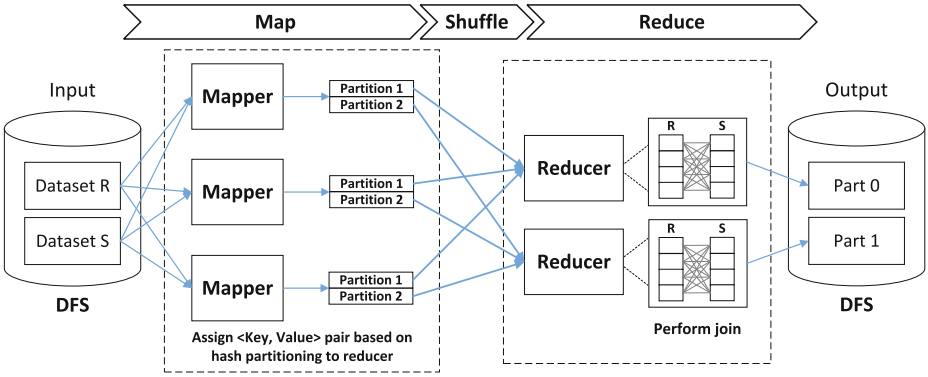


Fig. 1. The process of repartition join with the dataset R and S

System), and generates key-value pairs with the form of *(Key, Value)* as intermediate results, wherein *Key* represents join attribute. These intermediate results are to be assigned to reducers using hash partitioning. Second, in the shuffle phase, the reducers are notified to pull partitions across the network. Finally, the reduce function performs join operation with *(Key, list(Value))* pairs, wherein *list(Value)* is a list of values associated with the given key *Key*, and writes the final results to DFS. Fig.1 shows the process of a join operation between the dataset R and S with a typical reduce-side join algorithm, which is called repartition join[3].

Map-side join algorithms are more efficient than reduce-side join algorithms, because they produce the final results in map phase without shuffling data across the network. However, they can be used only in particular circumstances, i.e., one of the input datasets must be small enough to be buffered in memory of nodes. Reduce-side join algorithms are commonly used because they have fewer restrictions on input datasets. Therefore we focuse on the problem of data skew in reduce-side join.

3 The Problems in MapReduce-Based Join

MapReduce-based join algorithms sometimes suffer performance degradation from partitioning skew and heavy network overhead.

3.1 Partitioning Skew

Traditional join algorithms use hash partitioning to distribute data. Hash partitioning, the default partitioning function used in MapReduce model, is based on key hashing: *hash(Key) mod R*, wherein R is the number of reducers, which can allow data to be distributed uniformly when there are no skewness in the input datasets. In practice, however, partitioning skew tends to occur in processing skewed data and cause load imbalance, which means large amounts of data

are distributed on only a few nodes. Because the larger the volume of partition is, the longer time it takes to process data. In addition, the response time of a MapReduce job is dominated by the slowest reduce instance. So partitioning skew results in longer response time of MapReduce-based join on the whole.

According to the process of repartition join shown in Fig.1, to which node the intermediate results will be distributed, is determined by the partitioning function. Therefore, the key factor to achieve load balance in MapReduce-based join operation lies in whether or not the partitioning function can guarantee uniform distribution of data.

3.2 Heavy Network Overhead

Apart from partitioning skew, network overhead is another non-negligible problem. Large amounts of intermediate results are produced and need to be transferred across the nodes through network, which may consume a lot of network resources and result in longer execution time. For Hadoop, it runs mappers on those machines where splits of input datasets are located, so as to avoid network overhead. Most existing MapReduce-based join algorithms, however, does not take full advantage of such data locality feature in the reduce phase, and as a result, lots of intermediate results have to be transferred over network during the shuffle phase. In addition, transmission for skewded partitions may also introduce extra network overhead, because they have more data to transfer than non-skewed partitions. What's more, the reduce phase only can start after the shuffle phase completes, so network overhead is to increase the response time of the whole join operation.

Therefore, with evenly distribution of partitions, data transmission time tends to be equal among various partitions. In addition, applying the data locality feature in the reduce phase, can also reduce the amount of the transferred data and further improve the performance of join operation.

4 SALA Join Algorithm

To solve the above problem, we propose SALA join algorithm to handle skewed data and reduce network overhead. In this section, we first present an overview of SALA, and then present the volume/locality-aware partitioning used in SALA in detail. Also a example is discussed to compare SALA join with repartition join. Finally, we propose a cost model to analyze the performance of our algorithm.

4.1 Overview

We propose SALA join algorithm to handle skewed data. The core idea of SALA join is to distribute intermediate results based on the distribution information of key's frequency and location. With volume/locality-aware partitioning scheme, SALA join is able to not only handle skewed data but also reduce network overhead.

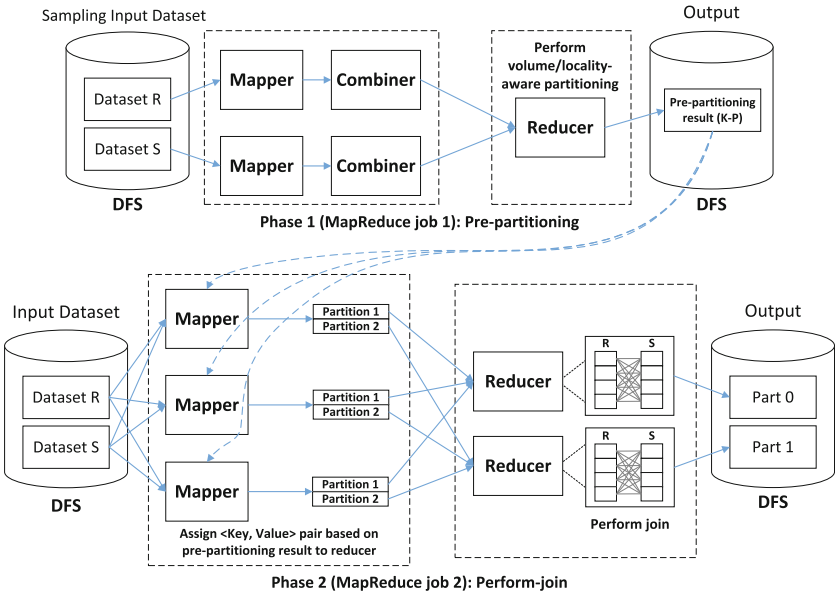


Fig. 2. The process of SALA join algorithm

Fig.1 shows the traditional process of MapReduce-based join, and the join process with SALA is shown in Fig.2. The main difference between the two algorithms is that SALA adds an additional MapReduce job to obtain key's distribution information. The SALA join includes two phases:

1. Phase 1: sample the input dataset and pre-compute the data to get the partitioning results, represented as *K-P*. *K-P* is a mapping array, and each of the array element is a map between a key and the partition that the key is assigned to.
2. Phase 2: perform the actually join operation. The join process is similar with repartition join, except that SALA join directly partitions the intermediate results according to *K-P* instead of using hash partitioning.

Since phase 2 of SALA is similar with repartition join, so we mainly focus on phase 1, i.e., the pre-partitioning process.

4.2 The Pre-partitioning Process

The pre-partitioning process is to pre-compute the sample input dataset to get *K-P*. and includes three phases - map phase, combine phase and reduce phase:

– Map phase: process the sample input dataset and take the join attribute as the *Key*. The output will be *(Key, (node,* 1)), wherein *node* represents the node on which the data is located and the number 1 represents the frequency of this key.

- Combine phase: the combine task will count the frequency of each key on the current node and the output will be *(Key, (node, sum))*, i.e., outputting the total frequency of each *Key* on the node.
- Reduce phase: volume/locality-aware partitioning is employed to get the pre-partitioning result *(Key, Partition)*, i.e., *K-P*.

Volume/locality-aware partitioning plays an important role in SALA join, so we discuss it in more detail below.

4.3 Volume/Locality-Aware Partitioning

Volume/locality-aware partitioning can not only deal with partitioning skew to achieve load balance, but also reduce the data transferred over network. Assuming that the data volume is M (which can be represented by the rows of the input dataset) and the number of nodes is N. In order to achieve load balance, the volume of data distributed to each node should be close to $\frac{M}{N}$. To reduce data transferred over network, volme/locality-aware partitioning makes full use of data locality feature by adopting greedy selection strategy as follows:

1. Each key value is distributed in higher priority to the node on which most intermediate results of this key are located.
2. First process the key value which has larger size of intermediate results.

Volume/locality-aware partitioning involves the following two steps:

1. Preparing step:
   (a) Compute the total rows of intermediate results of each key value in all nodes and write it as T_{key}.
   (b) Extract all *(Key, node, sum)* tuples from $(Key, list(node, sum))$ paris and store them in list L, meanwhile, put all key values into set K. After that, sequence all tuples in L in descending order based on the size of *sum*.
2. Partitioning step:
   (a) Traverse list L and process each tuple *(Key, node, sum)*. We use P_{key} to represent the partitioning result of each key value, which means to which node the key value should be distributed, and use V_{node} to record the volume of data that has been distributed to the node at present. If P_{key} is null, then determine whether or not $V_{node} + T_{key} \leq \frac{M}{N}$. If it is true, let $P_{key} = node$ and $V_{node} = V_{node} + T_{key}$.
   (b) Lastly, there may be some key values in K which have not been partitioned. In this case, find out the smallest V_{node}, to which the minimum volume of data is distributed at present, and then P_{key} will be the *node* that refers to V_{node}.

Algorithm 1 in Fig.3 formally describes volume/locality-aware partitioning. Due to the random sampling method used in the pre-partitioning process, there

Algorithm1: Volume/Locality-aware Partitioning
Input: pairs of $(Key, list(node, sum))$;
$M \leftarrow$ rows of input dataset; $N \leftarrow$ the number of nodes;
Output: partitioning results K-P

1. $T \leftarrow$ total rows of intermediate results in all nodes for each key value;
2. traverse the input and put all $(Key, node, sum)$ into L, put all Key into K;
3. initialize the list P and V;
4. **for** each $(Key, node, sum) \in L$ **do**
5. **if** $P[Key]$ is *null* and $V[node] + T[key] \leq \frac{M}{N}$ **then**
6. $P[Key] = node$;
7. $V[node] = V[node] + T[Key]$;
8. **endif**
9. **endfor**
10. **for** each $Key \in K$ **do**
11. **if** $P[Key]$ is *null* **then**
12. $node \leftarrow$ the node that refers to minimum $V[node]$ in V;
13. $P[Key] = node$;
14. $V[node] = V[node] + T[Key]$;
15. **endif**
16. **endfor**
17. return P as K-P;

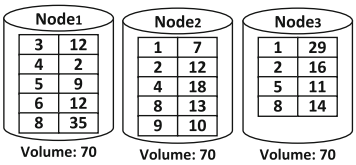
Fig. 3. The algorithm of volume/locality-aware partitioning

are some key values which may not be counted in. Therefore, when the intermediate results are partitioned in the perform-join process, key values which have been counted in will be partitioned according to the K-P, while key values which have not been counted in will still be partitioned by hash partitioning. Given that key values which have not been counted in only involve a small part of all key values, they will have negligible impact on the data distribution.

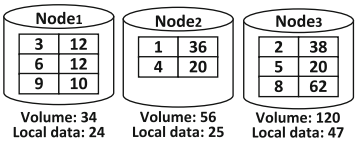
4.4 Example

Taking the following join operation for example: $R \overset{R.a=S.a}{\bowtie} S$. Assuming that there are 3 nodes in the cluster and the input data volume of each node is the same, i.e., 70, but with skewed data. Fig.4(a) shows the intermediate results produced in the map phase, in which, each row represents one key group *(Key, volume)*, wherein *volume* is the data volume of this key value on the present node.

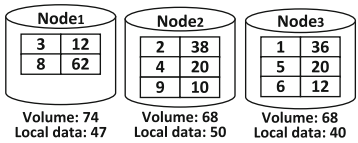
The partitioning results of repartitioning join and SALA join are shown in Fig.4(b) and Fig.4(c) respectively. According to Fig.4(b), partitioning skew happens in repartition join. Too much data are distributed to $Node_3$, almost four times of that distributed to $Node_1$. Therefore, load imbalance appears. However, as Fig.4(c) shows, SALA join algorithm has achieved better load balancing, and at the same time, the overall network overhead has reduced by 36% compared with repartition join.



(a) Distribution of intermediate results



(b) Partitioning with repartition join



(c) Partitioning with SALA join

Fig. 4. Partitioning results of various methods

With SALA join algorithm, the volume of data distributed to each node will tend to be equal and load balance is therefore achieved. Further, because each key value is first distributed to the node on which most of its intermediate results are located, the overall volume of data to be transferred over network is remarkably reduced and the performance of join operation is improved.

4.5 Cost Model

As shown in Fig.1, the whole processing time of the traditional reduce-side join algorithm includes three parts: processing time of map phase, transmission time of shuffle phase and processing time of reduce phase. For convenience, we use the following notations in Table 1:

Table 1. Table of notations

Notation	Meaning
t_m	*processing time for a record of input datasets in map phase*
t_s	*transmission time for a record in shuffle phase*
t_r	*processing time for a record in reduce phase*
M	*total rows of input dataset*
M_s	*total rows of sampling input dataset*
N	*the number of nodes*
B	*average available bandwidth of nodes*
L	*data locality of partitions*
s	*skewness of input dataset*

Because the response time of a MapReduce job is determined by the slowest reduce instance, we can estimate the response time by the reducer which is allocated the most volume of data, represented as R. Therefore, the cost model for a traditional reduce-side join algorithm is as follows:

$$T_{tra} = t_m \cdot \frac{M}{N} + \frac{R \cdot (1 - L)}{B} + t_r \cdot R \tag{1}$$

With uniform distribution of data, R_e tends to be $\frac{M}{N}$. In the case of partitioning skew, however, R_s tends to be:

$$R_s = M \cdot s + \frac{M \cdot (1-x)}{N} \tag{2}$$

The key values are $K=\{k_1, k_2, \ldots k_n\}$, and F_k represents the frequency of key value k on nodes. With hash partitioning, the data locality L_{tra} is $\frac{\sum_{k=k_1}^{kn} mean(F_k)}{M}$. Therefore, the cost model for a traditional reduce-side join algorithm in the case of partitioning skew can further be written as:

$$T_{tra} = t_m \cdot \frac{M}{N} + \frac{R_s \cdot (1 - L_{tra})}{B} + t_r \cdot R_s \tag{3}$$

With SALA join, the data locality L_{sala} tends to be $\frac{\sum_{k=k_1}^{kn} max(F_k)}{M}$. SALA join guarantees the uniform distribution of data, but needs an additional pre-partitioning process, and the required time of pre-partitioning process is represented as T_{pre}. Therefore the cost model for SALA join is:

$$T_{sala} = T_{pre} + t_m \cdot \frac{M}{N} + \frac{R_e \cdot (1 - L_{sala})}{B} + t_r \cdot R_e \tag{4}$$

Thus, SALA join algorithm is superior to traditional reduce-side join algorithm when satisfying the following condition:

$$T_{sala} - T_{tra} < 0 \Rightarrow T_{pre} < \frac{R_s \cdot (1 - L_{tra}) - R_e \cdot (1 - L_{sala})}{B} + t_r \cdot (R_s - R_e) \tag{5}$$

As can be seen from Eq.(5), SALA join performs better when the decreased of time results in from avoiding solving the partitioning skew is greater than the time used to process pre-partitioning. We can therefore employ Eq.(5) in optimal query plan selection. Here, according to many experiments, T_{pre} tends to be $0.23 \times t_m \times \frac{M}{N}$ and t_r tends to be $0.69 \times t_m$. We take N=5 and s=10%, then $L_{sala} = 2.94 \times L_{tra}$ and $R_s = 1.4 \times R_e$, so the Eq.(5) is satisfied, as is shown in Eq.(6). Also with the case of greater data skewness and the lower available bandwidth, SALA join will performs much better.

$$\begin{aligned} T_{pre} = 0.23 \cdot t_m \cdot \frac{M}{N} &< \frac{0.4 + 1.54 \cdot L_{tra}}{B} \cdot \frac{M}{N} + 0.28 \cdot t_m \cdot \frac{M}{N} \\ \Longrightarrow \quad -0.05 \cdot t_m - \frac{0.4 + 1.54 \cdot L_{tra}}{B} &< 0 \end{aligned} \tag{6}$$

5 Empirical Study

In this section, we conduct experiments to verify the efficiency of our approach. We mainly use the response time of join operation to demonstrate performance difference in the case of data skew. We compare SALA join with the repartition join algorithm [3] and SAND join algorithm [2], because repartition join is extensively used, and SAND join is a typical join algorithm to deal with skewed data.

5.1 Environmental Setup

Our experiments run on AliCloud (Alibaba Cloud Computing) with a 6-node cluster running native Hadoop 2.4.1, where there are 1 master node scheduling the task and 5 slave nodes taking charge of both storage and computation. Each node has two Xeon 2.3Ghz CPUs, 4GB memory and 60GB disk drive. HDFS block size is set to be 128MB and each node is configured to run one reducer task.

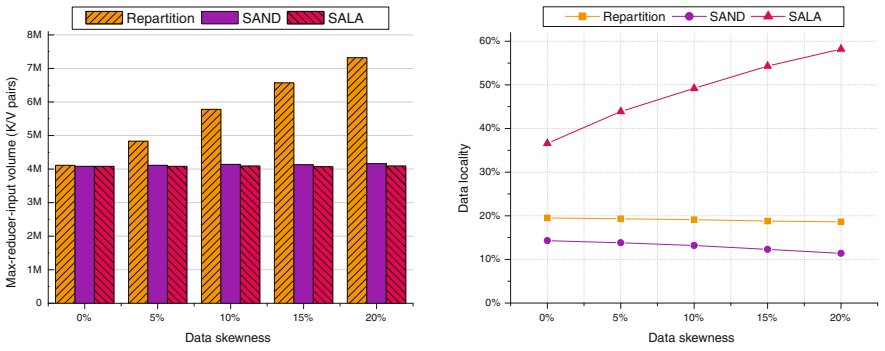
We use TPC-H to generate the input dataset and take the following query in our experiments:

$$\text{select} * \text{from CUSTOMER C join ORDER O on C.CUSKEY} = \text{O.CUSKEY}$$

In order to control data skewness, we randomly choose a portion of the input dataset *ORDER* and change its *CUSKEY* to the same value. For example, if the skewness is 10%, it means that we change 10% rows of the input dataset *ORDER* to have the same value in the join attribute *CUSKEY*. Finally, we generate 20 million records for query with various degree of data skewness.

5.2 Partitioning Effectiveness

Firstly, our concern is whether or not SALA join can effectively solve the partitioning skew problem. As analysis in Sect.2 has suggested that the key factor of load balancing is uniform distribution of data, we can therefore evaluate the capability of a join algorithm to handle skewed data by the value of *max-reducer-input*, i.e., the maximum volume of data distributed to any reducer. According to Fig.5(a), as the degree of skewness increases, repartition join concentrates a large amount of data on hot nodes, while both SALA join and SAND join can guarantee the uniform distribution of data.



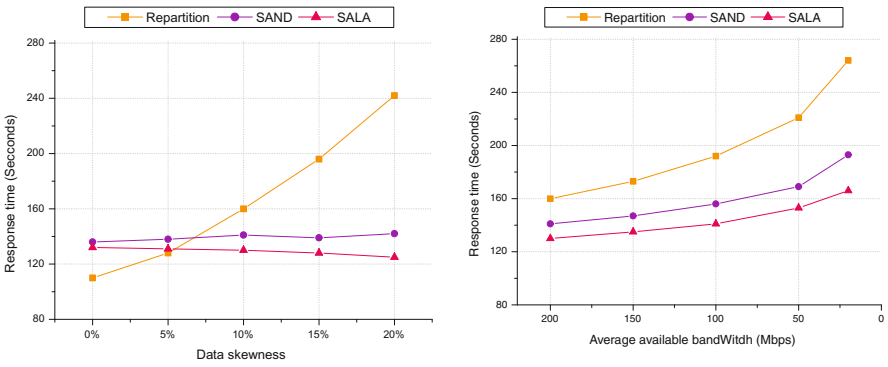
(a) Max-reducer-input (b) Data locality

Fig. 5. Partitioning with three join algorithms

Meanwhile, we use data locality to represent the volume of intermediate results that do not need to be transferred over network. From Fig.5(b), we can see that the data locality for SALA is much larger than the data locality for both repartition and SAND methods. Because the larger the data locality is, the less the volume of data required to be transferred across the network, thus less data needs to be transferred in SALA join algorithm than in both repartition and SAND methods.

5.3 Response Time

Fig.6(a) shows comparison between response time used to complete the given join operation under different degree of data skewness. The performance of repartition join is the best in the case of no or little data skewness, and the reason is that both SALA join and SAND join require additional MapReduce job to obtain frequency distribution of key values. However, with the increase of data skewness degree, the response time of repartition join increases almost linearly. The reason is that as the degree of data skewness increases, data will concentrates on hot nodes as Fig.5(a) shows, which increases the completion time of the overall join operation. However, both SALA and SAND can guarantee load balance, and therefore the response time remains steady with the increase of skewness degree. Most importantly, SALA join algorithm performs better than others when skewed data exist, because SALA not only achieves load balance but also reduces network overhead, thus speeding up the join operation process with the increase of data skewness degree.



(a) when data skewness changes (b) when average bandwidth changes

Fig. 6. Response time for three join algorithms

Fig.6(b) shows the variation of response time under different bandwidths when the degree of data skewness is 10%. It can be seen that as the average available bandwidth reduces, the problem of network overhead becomes prominent. It is because that the lower the bandwidth is, the longer time it will cost

to complete network transmission. By taking full advantage of data locality feature, the minimum volume of data is transferred with SALA join algorithm, and therefore SALA is preferable in the case of low bandwidth.

6 Related Work

In recent years, various approaches have been proposed to deal with skewed data in MapReduce-based join, such as [3,11,10,2,6,4,9,8]. The research work in [12] has demonstrated that the default hash partitioning function in Hadoop is not efficient for the skewed data and may lead to load imbalance of reducers.

The partitioning skew problem due to data skew can be solved by making a good partition scheme based on the key's frequency distribution, while sampling is a common way to obtain key's frequency [12,2]. The SAND join alogoritm [2] uses simple range partitioning for data distribution to achieve load balancing. Yujie Xu *et al.* proposed two partition schemes, namely cluster combination optimization and cluster partition combination based on random sampling results, to handle slight skew and heavy skew respectively.

Reducing the volume of data transferred across the network is an efficient way to further improve the performance of data-intensive join operation. Based on the priori knowledge of skewed key values, PRPD join geography proposed in [11] keeps all skewed rows locally to reduce the data volume transferred among nodes over network. LEEN scheme presented in [6] partitions the intermediate results based on key's frequency and the fairness score that is calculated after the shuffle phase. However, LEEN scheme changes the internal implementation of Hadoop and overlooks the advantage of overlapping between the shuffle and map.

An alternative strategy to mitigate skew is dividing the workload into fine-grained computation tasks and then scheduling them dynamically at runtime [5,7,9]. SkewTune [7] first identifies the task with the greatest expected remaining processing time when a node in the cluster becomes idle. The unprocessed data of this unfinished task is then repartitioned in a way that fully utilizes the computing power of cluster nodes.

7 Conclusion

In this paper, we propose SALA join algorithm, using volume/locality-aware partitioning to distribute intermediate results. On one hand, SALA guarantees the uniform distribution of data based on key's distribution information and therefore avoids partitioning skew problem. On the other hand, SALA takes full advantage of the data locality feature to reduce the volume of data transferred across the network. Experiments show that SALA is efficient to deal with skewed data. Our future work includes further improving performance of SALA join and extending volume/locality-aware partitioning to other MapReduce applications.

References

1. Ahmad, F., Lee, S., Thottethodi, M., Vijaykumar, T.N.: Mapreduce with communication overlap (marco). J. Parallel Distrib. Comput. **73**(5), 608–620 (2013)
2. Atta, F., Viglas, S.D., Niazi, S.: Sand join - a skew handling join algorithm for google's mapreduce framework. In: 2011 IEEE 14th International Multitopic Conference (INMIC), pp. 170–175, December 2011
3. Blanas, S., Patel, J.M., Ercegovac, V., Rao, J., Shekita, E.J., Tian, Y.: A comparison of join algorithms for log processing in mapreduce. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6–10, 2010, pp. 975–986 (2010)
4. Bruno, N., Kwon, Y.C., Wu, M.-C.: Advanced join strategies for large-scale distributed computation. PVLDB **7**(13), 1484–1495 (2014)
5. Dhawalia, P., Kailasam, S., Janakiram, D.: Chisel: a resource savvy approach for handling skew in mapreduce applications. In 2013 IEEE Sixth International Conference on Cloud Computing, Santa Clara, CA, USA, June 28 – July 3, 2013, pp. 652–660 (2013)
6. Ibrahim, S., Jin, H., Lu, L., Wu, S., He, B., Qi, L.: LEEN: locality/fairness-aware key partitioning for mapreduce in the cloud. In: Proceedings of the Cloud Computing, Second International Conference, CloudCom 2010, November 30 – December 3, 2010, Indianapolis, Indiana, USA, pp. 17–24 (2010)
7. Kwon, Y.C., Balazinska, M., Howe, B., Rolia, J.A.: Skewtune in action: Mitigating skew in mapreduce applications. PVLDB **5**(12), 1934–1937 (2012)
8. Kwon, Y.C., Ren, K., Balazinska, M., Howe, B.: Managing skew in hadoop. IEEE Data Eng. Bull. **36**(1), 24–33 (2013)
9. Lynden, S.J., Tanimura, Y., Kojima, I., Matono, A.: Dynamic data redistribution for mapreduce joins. In: IEEE 3rd International Conference on Cloud Computing Technology and Science, CloudCom 2011, Athens, Greece, November 29 – December 1, 2011, pp. 717–723 (2011)
10. Yu, X., Kostamaa, P.: Efficient outer join data skew handling in parallel DBMS. PVLDB **2**(2), 1390–1396 (2009)
11. Xu, Y., Kostamaa, P., Zhou, X., Chen, L.: Handling data skew in parallel joins in shared-nothing systems. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10–12, 2008, pp. 1043–1052 (2008)
12. Xu, Y., Zou, P., Qu, W., Li, Z., Li, K., Cui, X.: Sampling-based partitioning in mapreduce for skewed data. In: ChinaGrid Annual Conference (ChinaGrid), 2012 Seventh, pp. 1–8, September 2012