Research Article

MapReduce-Based Dynamic Partition Join with Shannon Entropy for Data Skewness

Donghua Chen¹ and Runtong Zhang²

¹*School of Information Technology and Management, University of International Business and Economics, Beijing 100029, China*

²*School of Economics and Management, Beijing Jiaotong University, Beijing 100044, China*

Correspondence should be addressed to Runtong Zhang; rtzhang@bjtu.edu.cn

Received 3 May 2021; Revised 4 September 2021; Accepted 18 October 2021; Published 24 November 2021

Academic Editor: Michele Risi

Copyright © 2021 Donghua Chen and Runtong Zhang. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Join operations of data sets play a crucial role in obtaining the relations of massive data in real life. Joining two data sets with MapReduce requires a proper design of the Map and Reduce stages for different scenarios. The factors affecting MapReduce join efficiency include the density of the data sets and data transmission over clusters like Hadoop. This study aims to improve the efficiency of MapReduce join algorithms on Hadoop by leveraging Shannon entropy to measure the information changes of data sets being joined in different MapReduce stages. To reduce the uncertainty of data sets in joins through the network, a novel MapReduce join algorithm with dynamic partition strategies called dynamic partition join (DPJ) is proposed. Leveraging the changes of entropy in the partitions of data sets during the Map and Reduce stages revises the logical partitions by changing the original input of the reduce tasks in the MapReduce jobs. Experimental results indicate that the entropy-based measures can measure entropy changes of join operations. Moreover, the DPJ variant methods achieved lower entropy compared with the existing joins, thereby increasing the feasibility of MapReduce join operations for different scenarios on Hadoop.

1. Introduction

Join algorithms between two data sets in stand-alone relational databases have been optimized for years; meanwhile, the increasing needs of big data analysis result in the emergence of various types of parallel join algorithms [1]. In the era of big data, such join operations on large data sets should be performed in existing distributed computing architectures, such as Apache Hadoop; that is, efficient joins must follow the scheme of programming models and require the extended revision of conventional joins for architectures [2]. At present, the Hadoop system is able to process big data in a rapid way [3]. Joining two data sets with MapReduce requires a proper design of the Map and Reduce stages for different scenarios. The factors affecting MapReduce join efficiency include the density of the data sets and data transmission over clusters like Hadoop. Join algorithms should be revised to utilize the ability of parallel computing in Hadoop and data set sizes being joined in performing joins for large data sets in Hadoop. However, only a few of

these algorithms have considered the data skewness of scientific data sets to optimize join operations. There exists difficulty in estimating the amount of information within heterogeneous data sets for each join. The skewness causes the occurrences of uneven distributions and a severe problem of load imbalance; the use of join operations on big data even makes the skewness occur exponentially [4]. Since Shannon's theory indicates that information is a measurable commodity, it is possible that MapReduce stages and the transmission of data sets over Hadoop clusters can be treated as a message channel between senders (mappers) and receivers (reducers). A data set to be joined is considered as a memoryless message source that contains a set of messages with its own probabilities to send to receivers.

This study aims to improve the efficiency of MapReduce join algorithms on Hadoop by leveraging Shannon entropy to measure the information changes of data sets being joined in different MapReduce stages. Our study highlights that leveraging the changes of entropy in the partitions of data sets during the Map and Reduce stages revises the logical

partitions by changing the original input of the reduce tasks in Hadoop MapReduce jobs. Such an action is particularly beneficial when different operations are required to handle different customized join operations for data with the same key on Hadoop.

The remainder of this paper is organized as follows. The next section summarizes the related work in this research field. Section 3 firstly leverages Shannon entropy to model the data intensity of data sets in a MapReduce join operation, and then it proposes a novel join algorithm with dynamic partition strategies for two-way joins by optimizing the partitions of data sets with skewed data intensity for reduce tasks. Section 4 conducts a series of experiments on different settings of data set sizes to compare the proposed method with the methods in literature. Then, Section 5 discusses our work in practice. Finally, the work is concluded in Section 6.

2. Related Work

MapReduce is the most widely used programming model in Hadoop and uses the Map, Combine, Shuffle, and Reduce stages to divide numerous data sets into small partitions and reduces the merged and sorted partitions on a cluster of computer nodes [5]. The Hadoop MapReduce architecture has contributed immensely to the existing big-data-related analyses. Joining large data sets, which is complex and time consuming, is one of the most important uses in Hadoop [6]. Applications of join operations include privacy preserving similarity [7], parallel set-similarity [8], and fuzzy joins [9], which focus on similarity estimation.

Join operations in MapReduce are time consuming and cumbersome. During a join operation in a Hadoop cluster, a mapper reads a partition from data sets and writes pairs of join keys and their corresponding values to intermediate files. Thereafter, the intermediate results are merged and sorted to be fed into numerous reducers. Eventually, each reducer that performs multiple Reduce tasks joins two sets of records with the same key. Join optimization has encountered considerable bottlenecks in reducing time cost and increasing the feasibility of dynamic partitioning data for different purposes. The main reason is that the shuffle stage on Hadoop is performed with a cluster network. With hardware upgrade, GPU is also very promising to improve the performance of join operations in a hybrid framework [10]. Existing join processing using the MapReduce programming model includes k-NN [11], Equi- [12], Theta- [13], Similarity [14], Top-K [15], and filter-based joins [16]. To handle the problem that existing MapReduce-based filtering methods require multiple MapReduce jobs to improve the join performance, the adaptive filter-based join algorithm is proposed [17]. The MapReduce framework is also suitable to handle large-scale high-dimensional data similarity joins [18]. For example, spatial join is able to perform data analysis in geospatial applications, which contain massive geographical information in a high-dimensional form [19].

The various types of join algorithms in accordance with the actual sides of performing MapReduce join operations on Hadoop include the map-side and reduce-side joins. Such algorithms are also known for repartition, broadcast, and replicated join in accordance with their features [20]. Join

algorithms utilize the ability of parallel computing in Hadoop and data set sizes being joined in performing joins for large data sets in Hadoop. In general, Bloom filter is able to reduce workload such as minimizing nonjoining data and reducing the costs of communication [21]. However, only a few of these algorithms have considered the data skewness of scientific data sets to optimize join operations. The reason is the difficulty in estimating the amount of information within heterogeneous data sets for each join. Chen et al. [22] promoted metric similarity joins by using two sampling-based partition methods to enable equal-sized partitions. In general, a data set contains different amounts of relations among data fields, thereby causing difference in workloads of Reduce tasks fed into a reducer, which differs from traditional joins. To reduce the complexity of constructing the index structure by using the see-based dynamic partitioning, a MapReduce-based kNN-join algorithm for query processing is proposed [23]. The partition phrase is also focused on in existing research to produce filter balanced and meaning subdivisions of the data sets [24]. Without effectively estimating such a difference of workload in two data sets with proper measures, joining data sets with skewed data intensity affects join performance and its feasibility.

Shannon entropy in information theory, which initially measures the information of communication systems, characterizes the impurity of arbitrary collections of examples [25]. The entropy of a variable is used to estimate the degree of information or uncertainty inherent in the scope of the variable outcomes. It is originally used to estimate the information value of a message between different entities. However, the entropy is also useful to measure diversity. At present, this theory has shaped virtually all systems that store, process, or transmit information in digital forms (e.g., performing knowledge reduction on the basis of information entropy using MapReduce) [26]. Chaos of data is the complete unpredictability of all records in a data set and can be quantified using Shannon entropy [27]. Therefore, a MapReduce-based join is naturally regarded as an entropy-based system for storing, processing, and transmitting data sets. With the use of hyperparameter optimization with automated tools, MapReduce-based joins can be easily applied in big data production environments [28]. In the future, automated join optimizer systems will be able to perform optimization of the hyperparameters of join operations and obtain the most efficient join strategy for various big data applications [29].

3. Materials and Methods

We first illustrate the concept of Shannon entropy of data sets to be joined using MapReduce and then present a framework to evaluate the entropy of data sets for existing join algorithms using MapReduce. A novel join algorithm with dynamic partition strategies is also proposed.

3.1. Shannon Entropy of Data Sets. Shannon's theory indicates that information is a measurable commodity. Considering entropy H of a discrete random variable $X = \{x_1, \dots, x_n\}$ and the

probability mass function is defined as $P(X)$, we obtain $H(X) = E[I(X)] = E[-\log(P(X))]$ according to Shannon's theory. Therefore, MapReduce stages and the transmission of data sets over Hadoop clusters can be treated as a message channel between senders (mappers) and receivers (reducers). A data set to be joined is a memoryless message source that contains a set of messages with its own probabilities to send to receivers.

Figure 1 illustrates a toy example of two data sets to be joined in MapReduce-based joins. A message in a data set is assumed as a record (a row) with data fields. For example, given two data sets (i.e., left data set (L) and right data set (R)) in data fields D ($T \in L$) and C ($T \in R$), T is a join key between L and R . A two-way join operation discussed in this study is to join two data sets into a new data set containing data fields D , C , and T , where $T \in L$ and $T \in R$.

DS is a data set with two data fields (D and T) similar to the left data set in Figure 1, where each record stands for an instance of relations between D and T . The amount of information of a record (r) is regarded as self-information (I) of a record in entropy. Thus, $I(r) = -\log(p_r)$, where p_r stands for the probability of r within DS. Therefore, the total information of DS is obtained as follows:

$$H(\text{DS}) = \sum_{i=1}^N I(r_i), \quad (1)$$

where r_i represents the i th of the unique records from DS and N is the number of unique records. $H(\text{DS})$ is called the first order of entropy of DS, which is also regarded as the degree of uncertainty within DS.

Equation (1) is ideally an extreme case without skewed data intensity. In general, supposing that p_i is the probability of a relation i in DS, we have the following equation:

$$H(\text{DS}) = -\sum_{i=1}^N p_i \log_2 p_i, \quad (2)$$

where p_i is the number of occurrences of the records divided by N . The maximum entropy of DS is $H(\text{DS}) = \log N$ if all records have equivalent p_i . The difference between $\log N$ and $H(\text{DS})$ in (2) is called the redundancy of data set. Equations (1) and (2) leverage Shannon entropy to estimate the uncertainty of data sets or the partitions using MapReduce on Hadoop clusters.

Figure 2 illustrates the information changes of data sets in the different stages of MapReduce. Four data accessing points, namely, mapper input (MI), mapper output (MO), reducer input (RI), and reducer output (RO), are evaluated using Shannon entropy. Two data sets are input into separate mappers and output to an intermediate file. MI and MO are the input and output, respectively, of a mapper. After the shuffle stage, the merged and sorted outputs of a mapper are fed into numerous reducers. RI and RO are the input and output, respectively, of a reducer.

Supposing that a data set is divided into m partitions after MI, the entropy of a partition (Partition_i) can be defined as follows:

Left data set		Right data set	
D	T	C	T
D1	T1	C1	T1
D2	T2	C2	T2
D3	T2	C3	T2
D3	T1	C4	T2

FIGURE 1: Toy example of two data sets to be joined.

$$H(\text{Partition}_i) = -\sum_{j=1}^{|\text{Relations}|} p_{i,j} \log_2 p_{i,j}, \quad (3)$$

where $p_{i,j} = m(i, j)/m(i)$ is the probability that a record of a partition i belongs to partition j , $m(i, j)$ is the number of records in partition i with relation j , and m_i is the number of records in partition i . The total entropy of all partitions after MI is defined as follows:

$$\bar{H}(\text{Partition}) = \sum_{i=1}^K \frac{m_i}{m} H(\text{Partition}_i), \quad (4)$$

where m_i is the number of records in partition i and m is the total number of records in a data set. Equations (3) and (4) represent how the total entropy of a group of partitions is calculated after MO.

To measure the impurity of a group of partitions from a data set, information gain (G) is used to estimate the gain of entropy when a join algorithm divides the data set into a group of partitions (PS) following a pipeline of MI, MO, RI, and RO. For example, if the gain of entropy from MI to MO increases, then the aggregated output of mappers contains high uncertainty of data sets. Accordingly, reducers need capability to complete further join operations. If PS is a group of partitions, then we have the following equation:

$$G(\text{DS}, \text{PS}) = H(\text{DS}) - \sum_{v \in \text{values}(\text{PS})} \frac{|\text{DS}_v|}{|\text{DS}|} H(\text{DS}_v), \quad (5)$$

where DS_v is a partition of DS and $|\text{DS}|$ represents the number of records in DS. Therefore, the information change between the input and output of mappers is denoted as $G(\text{mapper}) = H(\text{MO}) - H(\text{MI})$. Similarly, the information change of the reducers is denoted as $G(\text{reducer}) = H(\text{RO}) - H(\text{RI})$, whereas the information gain between MO and RI is denoted as $G(\text{shuffle}) = H(\text{RI}) - H(\text{MO})$. Equations (1) to (5) present a systematic modeling method based on Shannon entropy to estimate the uncertainty of the data sets in join algorithms during the different MapReduce stages on clusters.

3.2. MapReduce-Based Evaluation for Join Algorithms. Evaluating information changes of data sets during MapReduce join operations are difficult when handling data sets with massive contents. Therefore, a five-stage

FIGURE 2: Information changes of data in MapReduce join algorithms.

MapReduce-based evaluation framework for MapReduce join algorithms is proposed (see Figure 3). The framework with Stages 1 to 5 is designed by using the MapReduce programming model to overcome the difficulty of the data set sizes.

In Stage 1, the entropy of two data sets is calculated from the Left Mapper for the left data set and Right Mapper for the right data set. Thereafter, the merged and sorted results are fed into the reducers. Eventually, H (left data set) and H (right data set) are obtained. Thereafter, H (MI) is defined as in equation (6). In Stage 2, the entropy of the intermediate output of the mappers for the left and right data sets in a join algorithm is obtained in H (MO of left data set) and H (MO of right data set). Thereafter, the total entropy H (MO) is the same as in equation (7). In Stage 3, the intermediate results after the shuffle stage in the join algorithm are evaluated in H (RI). In Stage 4, the entropy of RO in the join algorithm is obtained. Lastly, all entropy values of Stages 1 to 4 are summarized in Stage 5.

$$H(MI) = H(L) + H(R), \quad (6)$$

$$H(MO) = H(L_MO) + H(R_MO). \quad (7)$$

The entropy-based metrics representing the information changes of the data sets in join operation are defined as H (MI), H (MO), H (RI), and H (RO). Comparing the measures illustrates the performance of different MapReduce join algorithms in terms of change of entropy and accordingly reflects their efficiency of join operations.

3.3. MapReduce Join Algorithms. The existing join algorithms have advantages in different application scenarios [30]. The entropy-based framework in Figure 3 can integrate into these algorithms to quantify the uncertainty of the intermediate results produced in the Map and Reduce stages in Hadoop. To simplify the descriptions of these algorithms, Figure 4 illustrates the toy examples of two data sets to be joined in different MapReduce join algorithms. In the figure, T_1 and T_2 are two toy data sets, where the values of the join key are denoted as $V_1, V_2, \dots, V_n$.

A reduce-side join, which is also called sort-merge join, in Figure 4(a) preprocess T_1 and T_2 to organize them in terms of join keys. Thereafter, all tuples are sorted and merged before being fed into the reducers on the basis of the join keys, such that all tuples with the same key go to one reducer. The join works only for equi-joins. A reducer in this join eventually receives a key and its values from both data sets. The number of reducers is the same as that of the keys generated from the mappers. Thereafter, the Hadoop MapReduce framework sorts all keys and passes them to the reducers. Tuples from a data set come before another data set because the sorting is done on composite keys. Lastly, a reducer performs cross-product between records to obtain joining results.

A map-side join in Figure 4(b) occurs when a data set is relatively smaller than the other data set, assuming that a smaller data set can fit into memory easily [31]. The map-side join is also called memory-backed join [32]. In this case, this algorithm initially replicates the small data set to the distributed cache of Hadoop to all the mapper sides where the data from the smaller data set is loaded into the memory of the mapper hosts before implementing the tasks of the mapper. When a record from a large data set is input into a mapper, each mapper looks up join keys from the cached data sets and fetches values from the cache accordingly. Thereafter, the mapper performs join operation using data sets in the mapper for direct output to HDFS without using reducers.

When the small data set cannot fit into memory, the cached table in the framework is replaced by a hash table or the Bloom filter [33] to store join keys from the aforementioned data set. Eventually, join performance is improved by reducing the unnecessary transmission of data through the network, which is called semireduce side join (see Figure 4(c)). This join often occurs when the join keys of the small data set can fit into the memory. Thereafter, a reducer is responsible for the cross-product of records similar to the reduce-side join. The Bloom filter is used to solve the problem in which Hadoop is inefficient to perform the join operation because the reduce-side join constantly processes all records in the data sets even in the

FIGURE 3: Entropy evaluation for join algorithms based on the five MapReduce jobs on Hadoop.

FIGURE 4: Example data flows of typical join algorithms in MapReduce. Input data sets are T1 and T2. (a) Reduce-side join. (b) Map-side join. (c) semi-reduce-side join. (d) Semi-reduce-side join with Bloom filter.

case when only a small fraction of the data sets are relevant. The probability of a false positive (p) after inserting n elements in the Bloom filter can be calculated in [34] as follows:

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k, \quad (8)$$

where n is the number of elements, k is the number of independent hash functions, and m is the number of bits. The size of the Bloom filter is fixed regardless of the number of the elements n . When using the Bloom filter as shown in Figure 4(d), the parameters of the filter should be predefined to ensure that all keys from the small data set have sufficient spaces for indexing the join keys by the Bloom filter.

3.4. Join with Dynamic Partition Strategies. Although the Hadoop framework internally divides the data sets into partitions for different mappers to process, the existing join algorithms, such as reduce-side join, cannot perform dynamic partitions on the basis of the data intensity of the data sets to be joined. In existing algorithms, an output key of a mapper is consistently the join key of reducers, thereby forcing all values belonging to one key to be fed into a single reducer. This scenario causes heavy workloads to a reducer when the reducer receives numerous values, whereas other reducers receive considerably less values.

To address the aforementioned problem, a novel join algorithm with dynamic partition strategies, which is called dynamic partition join (DPJ), over the number of records of data sets is proposed. DPJ aims to dynamically revise the number of Reduce tasks after a mapper's output by introducing a parameter that determines the number of partitions that the data set should be divided into. The join enables users to specify the number of partitions over the data sets to change the number of reducers. A two-phrase MapReduce job on Hadoop is required to perform this join. A toy example based on the data sets in Figure 1 is demonstrated as follows.

In the first stage, a counter job to calculate the number of records within the two data sets is performed. After the job is completed, the total numbers of records within the data sets are determined and fed into the second job as a parameter to perform join operations. The counter job uses Hadoop counters to obtain numbers of records. Thus, its time cost can be ignored.

In the second stage, a join job is performed to generate the joined results. This job requires three parameters, namely, number of partitions (N), total record number of the left data set (RN_L), and total record number of the right data set (RN_R). The steps are as follows.

Step 1. Calculate the number of partitions of the left (P_L) and right (P_R) data sets.

After N is determined, the P_L and P_R parameters by using equations (9) and (10) are determined to mappers and reducers of the job during the job configuration.

$$P_L = \begin{cases} \frac{RN_L}{N+1}, & RN_L \% N > 0, \\ \frac{RN_L}{N}, & \text{otherwise,} \end{cases} \quad (9)$$

$$P_R = \begin{cases} \frac{RN_R}{N+1}, & RN_R \% N > 0, \\ \frac{RN_R}{N}, & \text{otherwise.} \end{cases} \quad (10)$$

Step 2. Set up mappers for the left and right data sets.

The left data set is input into the Left Mapper defined in Algorithm 1 and the right data set is input into the Right Mapper defined in Algorithm 2. In the algorithms, [key, value] stands for a record consisting of a specified join key and its values in a data set, $L(i)$ stands for the i th partitions

of a key from the left data set, $R(i)$ stands for the i th partitions of a key from the right data set, LN is a Hadoop counter to count the number of records in the left data set, and RN is another counter to count the number of records in the right data set.

Step 3. Set up reducers for the output of the combiners.

The reducer defined in Algorithm 3 joins the partitioned records with the same key. Meanwhile, the job reduces the workload of each reducer because the number of Reduce task increases. In the algorithm, L stands for the left data set, R stands for the right data set, and k is the original join key located in the third part within the partition key.

The proposed DPJ on Hadoop is defined in Steps 1 to 3. Table 1 illustrates a split data analysis based on the toy data sets. Table 2 shows its inputs of reducers.

Various variant methods based on DPJ include DPJ with combiner, DPJ with Bloom filter, and DPJ with combiner and Bloom filter. The combiner is used to reduce the time cost of shuffle stage. The Bloom filter is used to utilize the join keys of the small data set in reducing the number of unnecessary join operations between the two data sets. The experimental results are examined in the different settings of the data set sizes for existing join algorithms and variant methods.

4. Experimentation and Results

To empirically evaluate the aforementioned join algorithms, several experiments are conducted in a Hadoop cluster to join two data sets with different settings of sizes. This section firstly introduces the detailed environment settings and the data set configurations in our experiments. Then, eight types of join algorithms on Hadoop have been defined and used in performance evaluation. Finally, a comparison of running time cost, information changes of data sets, and performance change over hyperparameters, respectively, is conducted.

4.1. Environment Settings. Our experiments are performed in the distributed Hadoop architecture. The cluster consists of one master node and 14 data nodes to support parallel computing. Its total configured capacity is 24.96 TB, where each data node has 1.78 TB capacity of distributed storage and 8.00 GB random-access memory. The performance metrics from MapReduce job history files are collected for analysis.

The process of evaluating the proposed method is as follows. First, several data sets needed in performing MapReduce join operations are uploaded to Hadoop Distributed File Systems (HDFS). The data sets have different data sizes according to simulating different workloads of join operations in big data analysis. Then, the proposed join algorithms have been revised following the MapReduce scheme in order to run on the Hadoop Reduce environment. Hyperparameters such as the input and output directories of data sets and partition number are supported flexibly to configure within the join algorithms, so we do not need to revise the same join algorithm in different cases of join

```

(1) class Mapper
(2)   method Map ([key, value])
(3)     set LN = LN + 1
(4)     set LPN = (LN - 1) /  $P_L$ 
(5)     for each  $i$  in [1, 2, ...,  $N$ ]
(6)       write (( $L$  (LPN),  $R$  ( $i$ ), key), ( $L$ , value))

```

ALGORITHM 1: Left Mapper for the left data set.

```

(1) class Mapper
(2)   method Map ([key, value])
(3)     set RN = RN + 1
(4)     set RPN = (RN - 1) /  $P_R$ 
(5)     for each  $i$  in [1, 2, ...,  $N$ ]
(6)       write (( $L$  ( $i$ ),  $R$  (RPN), key), ( $R$ , value))

```

ALGORITHM 2: Right Mapper for the right data set.

```

(1) class Reducer
(2)   method Reduce (key, [value])
(3)      $k = \text{key.get}()$ 
(4)     List ( $L$ ) = [value] from left data set
(5)     List ( $R$ ) = [value] from right data set
(6)     for each  $l$  in List ( $L$ ):
(7)       for each  $r$  in List ( $R$ )
(8)         write ( $k$ , ( $l$ ,  $r$ ))

```

ALGORITHM 3: Reducing for both data sets.

TABLE 1: Mapper outputs of the dynamic partition join using toy data sets.

Partitions of left data set	Partitions of right data set
L1:	R1:
(L1, R1, T1), D1	(L1, R1, T1), C1
(L1, R2, T1), D1	(L2, R1, T1), C1
(L1, R1, T2), D2	(L1, R1, T2), C2
(L1, R2, T2), D2	(L2, R1, T2), C2
L2:	R2:
(L2, R1, T2), D3	(L1, R2, T2), C3
(L2, R2, T2), D3	(L2, R2, T2), C3
(L2, R1, T1), D4	(L2, R2, T2), C4
(L2, R2, T1), D4	(L2, R2, T2), C4

TABLE 2: Reducer inputs of dynamic partition join using toy data sets.

Reducer	Input key	Input values
Reducer 1	(L1, R1, T1)	(D1, C1)
Reducer 2	(L1, R1, T2)	(D2, C2)
Reducer 3	(L1, R2, T2)	(D2, C3, C4)
Reducer 4	(L2, R1, T2)	(D3, C2)
Reducer 5	(L2, R2, T2),	(D3, C3, C4)

operations. After the join algorithm finishes running, the running time cost from its job history file is collected and analysed. Other measures related to CPU, virtual memory, physical memory, and heap use can be collected from the log file of running MapReduce jobs. Finally, a performance comparison from the perspectives of running time cost, information changes, and tuning performance is conducted.

4.2. Data Sets. The data sets used in the experiments are synthetic from real-world data sets. Different settings of using the data sets include three types of combinations of data set sizes, namely, Setting 1 (S1) in which both are relatively small data sets, Setting 2 (S2) in which one is a relatively large data set and another is a relatively small data set, and Setting 3 (S3) in which both are relatively large data sets. The small and large data sets referred to here are relative concepts. For example, a small data set may contain several hundred records, while the larger one may contain over one million records. The data sets have the same width in number of fields for proper comparison.

In S1, one data set with 15,490 records and another data set with 205,083 records are used. In S2, one data set with 205,083 records and another data set with 2,570,810 records are used. In S3, one data set with 1,299,729 records and another data set with 2,570,810 records are used. The same input of data sets and joined results in the experiments ensure that the measures are comparable.

4.3. Use of Join Algorithms in the Experiments. To simplify the references of the algorithms in the following experiments, the abbreviations of join are used in Table 3. The join algorithms include map-side join (MSJ), reduce-side join (RSJ), semi-reduce-side join (S-RSJ), semi-reduce-side join with Bloom filter (BS-RSJ), dynamic partition join (DSJ), dynamic partition join with combiner (C-DPJ), dynamic partition join with Bloom filter (B-DPJ), and dynamic partition join with combiner and Bloomer filter (BC-DPJ). Here the C-DPJ, B-DPJ, and BC-DPJ are variant methods of DPJ.

Each join algorithm is able to configure its hyperparameters in a flexible manner so we can evaluate the same join algorithm in different settings of hyperparameters. Our objective is to use multiple existing join algorithms and the proposed variant methods to examine the performance change of join operations in different cases.

4.4. Comparison of Time Cost between Join Algorithms. This section presents the result of comparison of time cost between the used algorithms in different cases in Table 3. Figure 5 illustrates the comparison of the time cost of joins in S1, S2, and S3. With increasing data set sizes, all time costs of joins increase. In S1, the difference of time cost is relatively small, but the existing joins perform slightly better than the variant methods of DPJ. With increasing data set sizes, time cost varies. In S2, MSJ, BS-RSJ, RSJ, and S-RSJ perform better than all DPJ-related variant methods, but B-DPJ is the best among the variant methods. In S3, the performances of

BS-RSJ and RSJ are better than those of the other joins. However, the performance of C-DPJ is approximately similar to that of S-RSJ. MSJ performs well in S1 and S2 but fails to run in S3. BS-RSJ performs best in S2 and S3.

Furthermore, we examine the time cost in different MapReduce stages over different settings of data sets. Table 4 summarizes the time costs in the different MapReduce stages in S1, S2, and S3. B-RSJ in S1 and S2 requires the largest setup time cost before its launch, although B-DPJ and BC-DPJ, which also use the Bloom filter, do not take substantial time for setup. However, the DPJ-derived methods need additional time during the cleanup. In S1, all time costs of the Map tasks are larger than those of the Reduce tasks. By contrast, the difference between them varies in S2 and S3. The time costs of the Map stage of B-DPJ and RSJ in S1 are larger than those of the Reduce stage in S2 and S3. The time cost of the Shuffle and Merge stages in S3 is less than those in S1 and S2. The time cost of MSJ in S3 is substantially larger than that in any other method owing to its failure to run in the cluster.

4.5. Comparison of Information Changes over Data Set Sizes. To investigate the amount of information changes in different data set sizes during performing join operations, we used the proposed MapReduce-based entropy evaluation framework to estimate information changes and make a comparison of their entropy in different MapReduce phrases. The information changes of each stage in MapReduce join algorithms with entropy-based measures are estimated in S1, S2, and S3 (see Figure 6). The inputs and outputs of the different joins are comparable because all MIs and ROs in S1, S2, and S3 are equivalent. Figure 6 shows that the entropy in MO and RI varies. Therefore, the joins change the amount of information during the Map and Reduce stages owing to their different techniques. The majority of the DPJ variant methods have less value of entropy than those of the other existing joins. Hence, the uncertainty degree of the data sets and partitions has changed, thereby reducing the workload of the Reduce tasks during MO and RI.

4.6. Comparison of Performance in Different DPJ Variants. To compare the performances of different DPJ variant methods, we conduct several experiments to examine DPJ, C-DPJ, B-DPJ, and BC-DPJ in terms of different performance measures such as cumulative CPU, virtual memory, physical memory, and heap over data set settings. The usage of cumulative CPU, virtual memory, physical memory, and heap is summarized in Figure 7. To evaluate the performance of the DPJ variant methods from the perspective of resource usage in the cluster, Figure 7 shows that B-DPJ using the Bloom filter in Figure 7(b) uses less CPU resources than PDJ in Figure 7(a). However, the virtual memory of B-DPJ is larger than that of PDJ in Reduce tasks. C-DPJ using combiners in Figure 7(c) uses larger CPU resources than DPJ, but other metrics remain similar. BC-DPJ using the Bloom filter and combiner in Figure 7(d) uses larger physical memory than PDJ, B-DPJ, and C-DPJ. However, the usage of

TABLE 3: Abbreviation of names of the join algorithms used in this study.

Name of join algorithm	Abbreviation
Map-side join	MSJ
Reduce-side join	RSJ
Semi-reduce-side join	S-RSJ
Semi-reduce-side join with Bloom filter	BS-RSJ
Dynamic partition join	DPJ
Dynamic partition join with combiner	C-DPJ
Dynamic partition join with Bloom filter	B-DPJ
Dynamic partition join with combiner and Bloom filter	BC-DPJ

FIGURE 5: Comparison of the time costs of the MapReduce join algorithms in three settings of data sets' sizes.

TABLE 4: Summary of the time cost of joins in different settings (unit: seconds).

Setting	Join	Setup	Map	Reduce	Shuffle	Merge
S1	B-RSJ	0.30	3.55	1.79	11.94	11.97
	C-DPJ	0.03	8.71	3.97	0	0
	MSJ	0.04	1.84	1.78	51.31	51.35
	B-DPJ	0.03	4.89	3.91	0	0
	BC-DPJ	0.04	6.92	2.73	0	0
	DPJ	0.04	7.46	6.66	0	0
	RSJ	0.05	3.33	2.42	0	0
	S-RSJ	0.05	3.31	1.96	4.35	4.37
S2	B-RSJ	0.03	10.87	6.76	0	0
	C-DPJ	0.04	97.06	48.05	0	0
	MSJ	0.05	16.61	3.28	20.39	20.39
	B-DPJ	0.03	28.3	40.79	18.04	18.04
	BC-DPJ	0.04	71.73	44.59	0	0
	DPJ	0.03	67.34	56.38	0.93	0.93
	RSJ	0.03	10.00	14.64	0	0
	S-RSJ	0.04	11.46	8.56	0	0
S3	B-RSJ	0.15	26.86	14.18	0	0
	C-DPJ	0.04	60.70	37.33	0	0
	MSJ	0.03	462.02	0	0	0
	B-DPJ	0.03	55.62	120.22	0	0
	BC-DPJ	0.04	102.71	74.60	9.24	9.24
	DPJ	0.07	58.09	63.07	0	0
	RSJ	0.03	11.66	21.87	0	0
	S-RSJ	0.04	89.56	17.67	0	0

FIGURE 6: Comparison of the entropy changes of the join algorithms in the different settings of data set sizes. (a) Setting 1 (both small data sets). (b) Setting 2 (one small data set and one large data set). (c) Setting 3 (both large data sets).

virtual memory in B-DPJ and BC-DPJ is larger than that of PDJ and C-DPJ.

Regarding the comparison of running time cost, Figure 8 summarizes the changes of time cost over the number of partitions and number of reducers. The time cost in Figure 8(a) increases over the number of partitions set in DPJ. Meanwhile, the time cost in Figure 8(b) decreases over the number of reducers and tends to be stable when approximately 20 reducers are set during the job configuration. Figure 8 illustrates the changes of time cost of job in four DPJ variant methods over two critical Hadoop parameters, namely, *sort.spill.percent* and *reduce.tasks*. The time costs in Figures 9(c) and 9(d) tend to remain relatively stable (overall less than 11 seconds), while the time costs in Figures 9(a) and 9(c) have larger fluctuations over the changes of two parameters (up to 14 seconds). The results showed that C-DPJ and BC-DPJ can provide stable and relatively great performance of time cost over changes of values of these parameters, while DPJ and B-DPJ have large different performance of time cost over the changes of values of two parameters.

5. Discussion

This study introduces Shannon entropy of information theory to evaluate the information changes of the data intensity of data sets during MapReduce join operations. A five-stage evaluation framework for evaluating entropy changes is successfully employed in evaluating the

information changes of joins. The current research likewise introduces DPJ that dynamically changes the number of Reduce tasks and reduces its entropy for joins accordingly. The experimental results indicate that DPJ variant methods achieve the same joined results with relatively smaller time costs and a lower entropy compared with existing MapReduce join algorithms.

DPJ illustrates the use of Shannon entropy in quantifying the skewed data intensity of data sets in join operations from the perspective of information theory. The existing common join algorithms, such as MSJ and BS-RSJ, perform better than other joins in terms of time cost (see Figure 5). Gufler et al. [35] proposed two load balancing approaches, namely, fine partitioning and dynamic fragmentation, to handle skewed data and complex Reduce tasks. However, the majority of the existing join algorithms have failed to address the problem of skewed data sets which may affect the performance of joining data sets using MapReduce [36]. Several directions for future research related to join algorithms include exploring indexing methods to accelerate join queries and design optimization methods for selecting appropriate join algorithms [37]. Afrati and Ullman [38] studied the problem of optimizing shares given a fixed number of Reduce processes to determine map-key and the shares that yield the least replication. A key challenge of ensuring balanced workload on Hadoop is to reduce partition skew among reducers without detailed distribution information on mapped data.

FIGURE 7: Comparison of the maximum resource usage among PDJ, B-DPJ, C-DPJ, and BC-DPJ with different settings of sizes. S1: small-size data sets, S2: middle-size data sets, S3: large-size data sets. (a) PDJ. (b) B-DPJ. (c) C-DPJ. (d) BC-DPJ.

Shannon entropy has been used in handling the heterogeneity of nodes in virtualized clusters and new data sets [39]. MapReduce stages (see Figure 2) are considered a channel between senders and receivers. Therefore, information change among the different stages reflects the efficiency of information processing tasks in mappers and reducers. A large entropy of input produces a small

entropy of output that often stands for a low uncertainty degree of generated output data, which is confirmed in the comparison in Figure 6.

DPJ utilizes a parameter of the number of partitions to adjust the number of reduce tasks; this process is not based on join keys of a data set but a dynamic parameter. Wang et al. [40] used similar techniques to divide mapped data into

FIGURE 8: Trends of time cost over the partition numbers and reducer numbers in DPJ. (a) Changes of time cost over partition numbers of DPJ. (b) Changes of time cost over reducer numbers of DPJ.

FIGURE 9: Comparison of the time cost of job in four DPJ methods over two Hadoop parameters, namely, *sort.spill.percent* and *reduce.tasks*. The unit of time cost is millisecond. (a) DPJ. (b) C-DPJ. (c) B-DPJ. (d) BC-DPJ.

many micropartitions and gradually gather statistics on their sizes during mapping. The parameter is not for physical partitions but logical partitions. In particular, this parameter promotes the proper processing for skewed data sets to be joined. DPJ does not change the number of reducers set during the job configuration because a Reduce task is just an

instance of reducers. Figure 8 indicates that the DPJ variant methods should be optimized by increasing the number of reducers to reduce their time cost and achieve similar time cost using S-RSJ. Therefore, the workload of each Reduce task using DPJ can be divided into several subtasks for one join key (see Tables 1 and 2). Such a division is particularly

beneficial when different operations are required to handle different customized join operations for small partitions with the same key.

Our methods do not aim to replace the existing MapReduce join algorithms for various application scenarios on Hadoop. Common join algorithms, namely, MSJ, RSJ, S-RSJ, and BS-RSJ (see Figure 4), work in different types of join operations for data sets without prior knowledge. Evidently, so does our method. However, if the scheme and knowledge of data set are known, then the Trojan Join running on Hadoop++ has advantages for joining data sets in this specific scenario [41]. This join aims to copartition the data sets at load time and group them on any attribute other than join attributes. Otherwise, if we are working on two-relation joins, then the most cost-effective method is to perform broadcast join, such as S-RSJ and BS-RSJ (see results in Figure 5). The hash table in S-RSJ and Bloom filter in BS-RSJ are used to optimize the broadcast join, but this optimization requires additional setup time cost in accordance with Table 4. The three variant methods of DPJ are proposed by using the combiner, Bloom filter, and both of them, separately. Figure 5 shows that DPJ performs worse than C-DPJ and BC-DPJ. Therefore, the combiner and Bloom filter play a substantial role in MapReduce joins. Overall, the criteria for selecting the proper join algorithms rely on the prior knowledge of schema and join conditions, transmission over network, and number of tuples. We introduce Shannon entropy in evaluating information changes between the input and output of mappers and reducers during join operations. The entropy-based measures naturally fit into the existing join operations because the definitions of senders and receivers in Shannon's theory are similar. In this manner, the mapper output is the input of the Reduce tasks through the Shuffle stage across the network.

There are threats that can result in different outcomes in our study. We proposed the counter-attack methods to address the issues. First, since the distributed computing environments like Hadoop are dynamic and complex, the running time of the same join algorithm every time is slightly different. Hadoop clusters with different data node settings can vary in running job performance. Therefore, we run the same experiment several times to obtain its mean running time for robust analysis in our experimental cluster. Second, the data set sizes used for evaluating MapReduce join performance include three groups of setting, namely, small size, middle size, and large size settings. We cannot examine the performance change of every combination of data sizes. Therefore, the three groups of setting are used to simulate different join scenarios. Third, the different data sets we used have various data skewness that may affect the performance of join operations on Hadoop. But we use entropy to evaluate the data skewness for convenience of comparison in the experiments. Finally, several measures to evaluate the performance on MapReduce jobs are used to ensure the comprehensive understanding of the job performance. With all aforementioned methods, the threats that exist in our experiments are addressed.

The limitation of this study is as follows. First, different data set sizes used to evaluate two-way join operations are only for case study. Evidently, these data sets are not sufficiently large to perform the optimization of joining large data sets in real life. Big data applications in various scenarios based on our methods should be further studied. Second, multiway joins involving over two data sets, such as Reduce-Side One-Shot join, Reduce-Side Cascade Join, and Theta-Join, are not considered [42]. Such multiway joins are increasingly important in big data analysis in the future. Third, although the entropy-based measures are appropriate in evaluating the information change of each MapReduce stage, the time cost of transmission, which has considerable influence on the join performance over the network, should be considered in the future. Thus, the measures only reflect the information change of the data intensity during the stages of MapReduce for two-way joins. Multiway joins with different entropy theories should be examined in the future. Besides, multiway join algorithms that considered data skewness in different distributed computing architectures such as Apache Spark [43] can be further studied on the basis of our research. Nonetheless, this study provides a novel method using MapReduce to achieve logically flexible partitions for join algorithms on Hadoop.

6. Conclusions

This study aims to improve the efficiency and enhance the flexibility of MapReduce join algorithms on Hadoop by leveraging Shannon entropy to measure the information changes of data sets being joined in different MapReduce stages. To reduce the uncertainty of data sets in joins through the network, a novel MapReduce join algorithm with dynamic partition strategies called DPJ is proposed. The experimental results indicate that the DPJ variant methods on Hadoop achieve the same joined results with a lower entropy compared with the existing MapReduce join algorithms. Our study highlights that leveraging the changes of entropy in the partitions of data sets during the Map and Reduce stages revises the logical partitions by changing the original input of the Reduce tasks in Hadoop MapReduce jobs. Such an action is particularly beneficial when different operations are required to handle different customized join operations for data with the same key on Hadoop. In the future, we are going to examine the use of the MapReduce-based multiway join algorithms with different entropy theories and their hyperparameter optimization methods with skewed data sets in distributed computing environments.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by a project of the National Natural Science Foundation of China (62102087 and 62173025) and the Fundamental Research Funds for the Central Universities in UIBE (20QD22 and CXTD12-04).

References

- [1] M. M. Barhoush, A. M. AlSobeh, and A. A. Rawashdeh, "A survey on parallel join algorithms using MapReduce on Hadoop," in *Proceedings of the IEEE Jordan Int. Joint Conf. Electrical Engineering and Information Technology (JEEIT)*, pp. 381–388, Amman, Jordan, April 2019.
- [2] K. Shim, "MapReduce algorithms for big data analysis," in *Proceedings of the International Workshop on Databases in Networked Information Systems (DNIS 2013)*, pp. 44–48, Aizu-Wakamatsu, Japan, March 2013.
- [3] A. O. Hassan and A. A. Hassan, "Simplified data processing for large cluster: a MapReduce and Hadoop based study," *Advances in Applied Sciences*, vol. 6, no. 3, pp. 43–48, 2021.
- [4] K. Meena, D. K. Tayal, O. Castillo, and A. Jain, "Handling data-skewness in character based string similarity join using Hadoop," *Applied Computing and Informatics*, 2020.
- [5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proceedings of the Sixth Conf. Symp. Operating Systems Design & Implementation (OSDI'04)*, vol. 6, pp. 137–150, San Francisco, CA, USA, December 2004.
- [6] D. Dai, X. Li, C. Wang, J. Zhang, and X. Zhou, "Detecting associations in large dataset on MapReduce," in *Proceedings of the IEEE International Conference Trust, Security and Privacy in Computing and Communication (TrustCom)*, pp. 1788–1794, Melbourne, Australia, July 2013.
- [7] X. Ding, W. Yang, K. K. R. Choo, X. Wang, and H. Jin, "Privacy preserving similarity joins using MapReduce," *Information Sciences*, vol. 493, pp. 20–33, 2019.
- [8] R. Vernica, M. J. Carey, and C. Li, "Efficient parallel set-similarity joins using MapReduce," in *Proceedings of the 2010 ACM Int. Conf. Management of Data (SIGMOD'10)*, pp. 495–506, Indianapolis Indiana, USA, June 2010.
- [9] C. Bellas and A. Gounaris, "A hybrid framework for exact set similarity join using a GPU," *Parallel Computing*, vol. 104–105, Article ID 102790, 2021.
- [10] F. N. Afrati, A. D. Sarma, D. Menestrina, A. Parameswaran, and J. D. Ullman, "Fuzzy joins using MapReduce," in *Proceedings of the IEEE 28th International Conference Data Engineering*, pp. 498–509, Arlington, VA, USA, April 2012.
- [11] X. Zhao, J. Zhang, and X. Qin, "kN. N.-D. P.: Handling data skewness in kNN joins using MapReduce," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 600–613, 2018.
- [12] H. C. Yang, A. Dasdan, R. L. Hsiao, and D. S. Parker, "Mapreduce-merge: simplified relational data processing on large clusters," in *Proceedings of the ACM International Conference Management of Data, (SIGMOD)*, pp. 1029–1040, Beijing, China, June 2007.
- [13] A. Okcan and M. Riedewald, "Processing theta-joins using MapReduce," in *Proceedings of the 2011 ACM International Conference Management of Data (SIGMOD'11)*, pp. 949–960, Athens Greece, June 2011.
- [14] R. Baraglia, G. D. F. Morales, and C. Lucchese, "Document similarity self-join with mapreduce," in *Proceedings of the IEEE International Conference Data Mining (ICDM)*, pp. 731–736, Sydney, NSW, Australia, December 2010.
- [15] Y. Kim and K. Shim, "Parallel top-k similarity join algorithms using MapReduce," in *Proceedings of the IEEE International Conference Data Engineering (ICDE)*, pp. 510–521, Arlington, VA, USA, April 2012.
- [16] S. M. Mahajan and M. V. P. Jadhav, "Bloom join fine-tunes distributed query in Hadoop environment," *International Journal of Advanced Engineering Technology*, vol. 4, no. 1, pp. 67–71, 2013.
- [17] S. Rababa and A. Al-Badarneh, "Optimizations for filter-based join algorithms in MapReduce," *Journal of Intelligent and Fuzzy Systems*, vol. 40, no. 5, pp. 8963–8980, 2021.
- [18] Y. Ma, R. Zhang, Z. Cui, and C. Lin, "Projection based large scale high-dimensional data similarity join using MapReduce framework," *IEEE Access*, vol. 8, Article ID 121677, July 2021.
- [19] A. Belussi, S. Migliorini, and A. Eldawy, "Cost estimation of spatial join in spatialhadoop," *GeoInformatica*, vol. 24, no. 4, pp. 1021–1059, 2020.
- [20] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian, "A comparison of join algorithms for log processing in MapReduce," in *Proceedings of the 2010 ACM Int. Conf. Management of Data (SIGMOD'10)*, pp. 975–986, Indianapolis Indiana, USA, June 2010.
- [21] A. C. Phan, T. C. Phan, and T. N. Trieu, "A theoretical and experimental comparison of large-scale join algorithms in spark," *SN Computer Science*, vol. 2, no. 5, 2021.
- [22] G. Chen, K. Yang, L. Chen, Y. Gao, B. Zheng, and C. Chen, "Metric similarity joins using MapReduce," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 3, pp. 656–669, 2016.
- [23] H. Lee, J. W. Chang, and C. Chae, "kNN-join query processing algorithm on mapreduce for large amounts of data," in *Proceedings of the 2021 International Symposium on Electrical, Electronics and Information Engineering*, pp. 538–544, Seoul, South Korea, February 2021.
- [24] S. Migliorini and A. Belussi, "A balanced solution for the partition-based spatial merge join in MapReduce," in *Proceedings of the EDBT/ICDT Workshops*, Copenhagen, Denmark, April 2020.
- [25] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [26] W. Cui and L. Huang, "Knowledge reduction method based on information entropy for port big data using MapReduce," in *Proceedings of the 2015 International Conference on Logistics, Informatics and Service Sciences (LISS)*, Barcelona, Spain, July 2015.
- [27] J. Chen, H. Chen, X. Chen, G. Zheng, and Z. Wu, "Data chaos: an entropy-based MapReduce framework for scalable learning," in *Proceedings of the IEEE International Conference Big Data*, pp. 71–78, Silicon Valley, CA, USA, October 2013.
- [28] D. Chen, R. Zhang, and R. G. Qiu, "Noninvasive MapReduce performance tuning using multiple tuning methods on Hadoop," *IEEE Systems Journal*, vol. 15, no. 2, pp. 2906–2917, 2020.
- [29] A. F. Al-Badarneh and S. A. Rababa, "An analysis of two-way equi-join algorithms under MapReduce," *Journal of King Saud University - Computer and Information Sciences*, vol. 33, no. 9, 2020.
- [30] J. Chandar, *Join Algorithms Using Map/Reduce*, Master of Science, School of Informatics, Univ. Edinburgh, Edinburgh, UK, 2010.
- [31] K. Lee, Y. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with MapReduce," *ACM SIGMOD Record*, vol. 40, no. 4, pp. 11–20.

- [32] J. Lin and C. Dyer, "Data-intensive text processing with MapReduce," *Synthesis Lectures on Human Language Technologies*, vol. 3, no. 1, pp. 1–177, 2010.
- [33] T. Lee, K. Kim, and H. J. Kim, "Join processing using Bloom filter in MapReduce," in *Proceedings of the . 2012 ACM Res. Applied Computation Symp*, pp. 100–105, San Antonio, TX, USA, October 2012.
- [34] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
- [35] B. Gufler, N. Augsten, A. Reiser, and A. Kemper, "Handling data skew in MapReduce," in *Proceedings of the 1st International Conference on Cloud Computing and Services Science*, pp. 574–583, Noordwijkerhout, Netherlands, 2011.
- [36] E. Gavagsaz, A. Rezaee, and H. Haj Seyyed Javadi, "Load balancing in join algorithms for skewed data in MapReduce systems," *The Journal of Supercomputing*, vol. 75, no. 1, pp. 228–254, 2019.
- [37] M. H. Khafagy, "Indexed map-reduce join algorithm," *International Journal of Scientific Engineering and Research*, vol. 6, no. 5, pp. 705–711, 2015.
- [38] F. N. Afrati and J. D. Ullman, "Optimizing multiway joins in a map-reduce environment," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 9, pp. 1282–1298, 2011.
- [39] K. H. K. Reddy, V. Pandey, and D. S. Roy, "A novel entropy-based dynamic data placement strategy for data intensive applications in Hadoop clusters," *International Journal of Big Data Intelligence*, vol. 6, no. 1, pp. 20–37, 2019.
- [40] Z. Wang, Q. Chen, B. Suo, W. Pan, and Z. Li, "Reducing partition skew on MapReduce: an incremental allocation approach," *Frontiers of Computer Science*, vol. 13, no. 5, pp. 960–975, 2019.
- [41] J. Dittrich, J. A. R. Quiané, A. Jindal, Y. Kargin, V. Setty, and J. Schad, "Hadoop++: making a yellow elephant run like a cheetah (without it even noticing)," in *Proceedings of the 36th Int. Conf. Very Large Data Bases (VLDB)*, vol. 3, no. 1, pp. 518–529, Singapore, September 2010.
- [42] X. Zhang, L. Chen, and M. Wang, "Efficient multi-way theta-join processing using MapReduce," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1184–1195, 2012.
- [43] A. Phan, T. Phan, and T. Trieu, "A comparative study of join algorithms in spark," in *Proceedings of the International Conference on Future Data and Security Engineering*, pp. 185–198, Springer, Quy Nhon, Vietnam, November 2020.