A Maximum Dimension Partitioning Approach for Efficiently Finding All Similar Pairs

Jia-Ling Koh and Shao-Chun Peng

Department of Information Science and Computer Engineering
National Taiwan Normal University
Taipei, Taiwan 106, R.O.C
Email: jlkoh@ntnu.edu.tw

Abstract. For solving the All Pair Similarity Search (APSS) problem efficiently, this paper provides a maximum dimension partitioning approach to effectively filter non-similar pairs in an early stage. At first, for each data point, the dimension with the maximum value is used to decide the corresponding segment of data partition. An adjusting method is designed to balance the number of elements in each data segment. The similar pairs consist of inter-segment similar pairs and intra-segment similar pairs, where most effort of computing APSS comes from the computation of finding inter-segment similar pairs. For speeding up the computation, a pilot-vector is used to represent each segment for estimating the upper bound of similarity between each segment pair. Only the segment pairs, whose upper bounds of similarity are larger than the given similarity threshold, need to generate the inter-segment data pairs as candidates. Moreover, based on the proposed partitioning method, we designed a MapReduce framework to solve the APSS problem in parallel. The performance evaluation results show the proposed method provides better pruning effectiveness on non-similar data pairs than the related works. Moreover, the proposed partition-based method can properly fit into the MapReduce programming scheme to effectively reduce the response time of solving the APSS problem.

1. Introduction

In real-world applications of data mining, a crucial problem is to perform similarity search, such as collaborative filtering for similarity-based recommendations, near duplicate document detection, and coalitions of click fraudster identification. Given a function $Sim(x, y)$ and a similarity threshold t, a similarity search aims to find all objects in a dataset with a similarity value of at least t compared to a query object. The All Pair Similarity Search (APSS) problem performs a similarity search for each object in a dataset to find all similar pairs in the dataset.

A data object in an application is generally numerically represented by a high dimensional vector, where each dimension is a feature extracted from the object. Suppose that the dataset consists of n objects and each object has a m dimensional feature vector. The time complexity of a brute force algorithm for APSS is $O(mn^2)$, which is infeasible in practice. Accordingly, there have been many works studied how to improve the performance efficiency for solving APSS [1, 2, 3, 5, 9, 12, 14]. In order to reduce the computation cost of solving APSS, it is necessary to effectively reduce the search space of the problem. In other words, it requires some pruning strategies to reduce the number of generated data pairs which need similarity computation. The cost of a pruning strategy is count into the total cost for solving the

problem. Therefore, the pruning strategy should be both effective and efficient. Moreover, to develop a parallelized approach for reducing response time is the recent direction for solving the issue of huge amount of data [10][11][15].

For solving the All Pair Similarity Search (APSS) problem efficiently, this paper provides a maximum dimension partitioning approach to effectively filter non-similar pairs in an early stage. At first, for each data point, the dimension with the maximum value is used to decide the corresponding segment of data partition. An adjusting method is designed to balance the number of elements in each data segment. The similar pairs consist of inter-segment similar pairs and intra-segment similar pairs, where most effort of computing APSS comes from the computation of finding inter-segment similar pairs. For speeding up the computation, a pilot-vector is used to represent each segment for estimating the upper bound of similarity between each segment pair. Only the segment pairs, whose upper bounds of similarity are larger than the given similarity threshold, need to generate the inter-segment data pairs as candidates. Moreover, the prefix filtering strategy is used to improve the efficiency of computing similarity of both segment pairs and intra-segment data pairs. Based on the partitioning method, we designed a MapReduce framework to solve the problem in parallel. The performance evaluation results show the proposed method provides better pruning effectiveness on non-similar data pairs than the related works. Moreover, the proposed partition-based method can properly fit into the MapReduce programming scheme to effectively reduce the response time of solving the APSS problem.

This paper is organized as follows. The problem definition and related work are introduced in Section 2. In Section 3, the details of the proposed partitioning method and the pruning strategy are introduced. The MapReduce extension is proposed in Section 4. The performance evaluation on the proposed methods and related works is reported in Section 5. Finally, in Section 6, we conclude this paper.

2. Preliminaries and Related Work

2.1 Problem Definition

Let $D = \{d_1, d_2, d_3, \ldots, d_n\}$ denote a set of data, where each data d_i is represented by a m dimensional vector $d_i = <d_i[1], d_i[2], d_i[3], \ldots d_i[m]>$. It is assumed that each vector is normalized. Accordingly, the similarity score between two data d_i and d_j is computed by the cosine-similarity function as follows:

$$Sim(d_i, d_j) = \sum_{k=1}^{m} d_i[f_k] \times d_j[f_k].$$

Given a threshold t, two vectors d_i and d_j form a *similar pair* if their similarity score is larger than or equal to the threshold value t, i.e. $Sim(d_i, d_j) \geq t$.

The All Pair Similarity Search (APSS) problem is to find all (d_i, d_j) pairs, where d_i, $d_j \in D$ and $Sim(d_i, d_j) \geq t$.

2.2 Related Work

When a dataset consists of high-dimensional data vectors, it is costly to perform similarity computation for data pairs. Accordingly, many strategies were designed to approximately estimate the similarity of a pair of data with the assistance of an

inverted list index on the dataset [2][4][5]. In order to save the computation time, when the partially computed similarity value of a data pair can decide that the pair is dissimilar, it is not necessary to compute all dimensions of the feature vectors for getting the exact similarity of the pair.

An inverted list is a data structure of indexing, for each feature dimension *f*, which constructs a linked list to store the object identifiers with a non-zero value on the dimension. It is the most popular data structure used in document retrieval systems for finding documents containing a query keyword. The studies in [5] and [6] considered that it is not necessarily to build a complete inverted index over the vector input. In [5], for a data object *x*, its cumulative estimated similarity on features with other objects is computed. The suffix feature values of *x* start being indexed only when the cumulative estimated similarity with other objects has reached the similarity threshold *t*. Besides, the prefix filtering principle was proposed to reduce the generated candidate pool size. Bayardo et. al [5] showed that, by using this method, the indexed feature values of the object *x* are enough to identify any object *y* that is potentially similar to *x* during the similarity search. [2] applied the Cauchy-Schwarz inequality to provide l^2-norm filtering. The l^2-norm was used to improve the estimation bound of similarity between feature vectors for getting a smaller inverted index size and pruning more candidates of APSS. However, to compute l^2-norm is costly for a high-dimensional dataset. [4] extended the prefix-filtering strategy in [5] for solving the Incremental All Pair Similarity Search (IAPSS) problem, which performs APSS multiple times over the same dataset by varying the similarity threshold. The previously mentioned works focused on reducing candidates of APSS, but obtaining better pruning effect in candidate generation may cause much processing effort. Accordingly, [12] aimed to decrease the computational cost of candidate generation by reducing the number of indexed objects. [16] also stated that the length of inverted index has significant effect on processing efficiency, i.e. prefix filtering does not always achieve high performance. Therefore, a cost model was proposed to decide the length of its prefix index for each object.

The key idea of Locality-sensitive hashing (LSH) methods is to hash the points using several hash functions so as to ensure that, for the vectors hashed into one bucket, the probabilities of similar pairs are much higher than non-similar pairs [9][13]. Since the Locality-sensitive hashing (LSH) approach groups similar vectors into one bucket with approximation, it has a trade-off between precision and recall. Besides, redundant computation occurs when multiple hash functions are used. The study [4] provided an exact algorithm to perform set-similarity join. The algorithm first generates signatures for input sets, where all pairs of sets whose signatures have overlap are candidates to find the similarity set pairs. Its experiments showed that, when inverted indexing and computation filtering methods are applied, the exact algorithm performs competitive to LSH.

In recent years, many studies solved the APSS by parallel processing for reducing the response time, mostly by providing a MapReduce framework [8, 10, 11, 15]. [8] implemented a basic algorithm for APSS, which consists of two separate MapReduce jobs: 1) construct the inverted list, 2) compute the similarities of data pairs according to the inverted list. Furthermore, three variations are provided to prune the non-similar pairs early from the computation for reducing its cost. [11] proposed the V-SMART-Join framework, which consists of two phases of MapReduce processing:

the first phase generates the candidate pairs and the second phase computes the similarity between all candidate pairs according to the constructed inverted index. The MapReduce framework proposed in [15] introduces a partitioning method where a data pre-processing phase first selects random points as centroids and the data points are assigned to the nearest centroid for computing centroid statistics. In the Similarity computation phase, the original dataset as well as the centroid statistics are read to construct the independent work sets, i.e. the set pairs for generating candidate pairs. Besides, an optional repartitioning step is used to enhance the load-balancing of the partitioning before computing similar pairs from the work sets.

[1] considered that most parallel methods of APSS using an inverted index to perform computation filtering suffer from excessive I/O and communication overhead of the intermediate partial results. Accordingly, a partition-based approach was proposed to solve APSS in parallel. This approach statically groups data vectors into partitions such that the dissimilar partitions can be revealed in an early stage to avoid unnecessary data loading and comparison. There are two steps to perform partitioning in [1]. The first step is to sort the data vectors according to their l^1-norm values in a non-decreasing order. The ordered list of data vectors is divided evenly into consecutive groups. In the second step, for the i-th group G_i, its vectors are further divided into i disjoint subgroups $G_{i,1};G_{i;2}$; ... ;and $G_{i;i}$ such that the vectors in $G_{i,j}$, where $j \leq i$, must be dissimilar with G_k for each $k \leq j$. For each data vector d in G_i, $maxw(d)$ denote the maximum value in d. Besides, $Leader(G_j)$ denote the vector in G_j with the maximum l^1-norm length. It is induced that $Sim(G_j, d) \leq maxw(d) \times Leader(G_j)$. Therefore, $maxw(d) \times Leader(G_j)$ is an estimation for the upper bound of similarity between d and the vectors in G_j. If $maxw(d) \times Leader(G_j)$ is less than the similarity threshold t, it implies that d is dissimilar with all the vectors in G_j such that d is assigned to $G_{ij.}$ From the generated groups, except for the group pairs whose vectors are sure dissimilar to each other, the other group pairs need to generate inter-group data pairs as candidates of APSS. The intra-group data pairs generated in each group are also candidate of APSS. Furthermore, $Sim(d_i, d_j) \leq \min(maxw(d_i) \times \|d_j\|_1, maxw(d_j) \times \|d_i\|_1)$ is used estimate the upper bound of similarity for pruning the dissimilar data pairs in the candidates. Finally, the remained data pairs need exactly compute their similarities to find similar pairs. [14] applies $\|d_i\|_r \times \|d_j\|_s$ to estimate the upper bound of similarity between d_i and d_j to identify more dissimilar vectors. Besides, the work aimed to solve the problem of load balance for [1] and tried to reduce the size gap among partitions. The partition-based approach is a good strategy for pruning non-similar pairs in an early stage. However, the weakness of the PSS (Partition-based Similarity Search) method in [1] and [14] occurs when applying to high dimensional vectors. As the dimension of vectors grows, the maximum l^1-norm length of the normalized vectors will grow as well. Accordingly, in most cases, the estimated upper bound of similarity in [1] is much higher than the exact similarity value. It causes the effectiveness of pruning dissimilar group pairs not well.

To summarize the above related works, this paper would combine the benefits of 2-level pruning strategies for speeding up the processing of APSS. We proposed a new static data partitioning in data pre-processing to reduce similarity computation of inter-segment data pairs, which improve the weakness of the PSS method [1]. Our proposed approach also applied the inverted index to perform dynamic filtering but requires less communication cost in a MapReduce framework.

3. A Partition-based approach for Solving APSS

In the maximum partitioning approach, the processing is divided into 4 main tasks for computation: 1) data partitioning, 2) find intra-segment similar data pairs, 3) generate candidate segment pairs, 4) prune inter-segment dissimilar data pairs and find inter-segment similar pairs. The strategies provided for processing each task are described in the following subsections.

3.1 Partitioning Method

According to the definition of cosine similarity, it is more likely that two vectors are similar if the maximum values in their vectors are located on the same dimension. Therefore, in our partitioning strategy, the vectors with the same maximum dimension are initially assigned to the same segment. There are three steps in the proposed partitioning method: 1) dimension reordering, 2) maximum-dimension partitioning, and 3) pilot vector computation.

Step 1: Dimension reordering

For each data dimension f_i, the number of data objects in D whose f_i has a non-zero value is counted, which is denoted as non-zero-count(f_i). The dimensions of the data objects in D are re-ordered by their non-zero-count values in descending order.

For example, in the sample dataset shown in Table 1, the non-zero-count values of dimensions f_1, f_2, f_3, and f_4 are 4, 3, 4, and 2, respectively. Therefore, the dimensions are reordered as f_1, f_3, f_2, and f_4.

Step 2: Maximum-dimension partitioning

For each data object d_i, find its dimension with the maximum value, denoted as $maxf(d_i)$. The data objects with the same $maxf$ value are assigned to the same segment. The partitioning result of the sample dataset D is shown in Table 2.

Step 3: Pilot vector computation

For each segment S_i, its pilot vector $S_i.pilot$ is defined as follows:

$$S_i.pilot[f_j] = \max(d_k[f_j] \mid \forall d_k \in S_i) \text{ for } j = 1, \ldots, m.$$

According to the partitioning result shown in Table 2, the pilot vectors of the three segments are shown as Table 3.

The number of the initial generated segments by the maximum-dimension partitioning method depends on the distribution of data. Accordingly, we provide the following adjusting methods to control the number of the generated segments.

1) Increase the number of segments:

The segment with the maximum number of data objects, say S_i, is selected to be divided into two segments. The data objects in S_i are sorted in decreasing order according to the l^1-norm length of their vectors. Then S_i is divided into two segments with equal size at the middle of the sorted objects. If there is more than one segment with the maximum number of data objects, let $max\_1\text{-}norm(S)$ and $min\_1\text{-}norm(S)$ denote the maximum l^1-norm length and the minimum l^1-norm length among the vectors in such a segment S, respectively. The segment S_j with a larger difference between $max\_1\text{-}norm(S_j)$ and $min\_1\text{-}norm(S_j)$ has a sparser data distribution. Therefore, in this case, the segment containing vectors with the largest range on their l^1-norm length is selected to be split.

Table 1: A sample dataset D.

	f_1	f_2	f_3	f_4
d_1	0	0	1	0
d_2	0	1	0	0
d_3	0.45	0.9	0	0
d_4	0.65	0	0.76	0
d_5	0.6	0	0.6	0.52
d_6	0.5	0.5	0.5	0.5

Table 2: The partitioning result on the sample dataset D.

	f_1	f_3	f_2	f_4	*maxf*	l^1-norm	assigned segment
d_6	0.5	0.5	0.5	0.5	f_1	2	S_1
d_5	0.6	0.6	0	0.52	f_1	1.72	S_1
d_3	*0.45*	*0*	*0.9*	*0*	f_2	*1.35*	S_2
d_2	*0*	*0*	*1*	*0*	f_2	*1*	S_2
d_4	0.65	0.76	0	0	f_3	1.41	S_3
d_1	0	1	0	0	f_3	1	S_3

Table 3: The pilot vectors of the segmentations for the sample dataset D.

	f_1	f_3	f_2	f_4
S_1.*pilot*	0.6	0.6	0.5	0.52
S_2.*pilot*	0.45	0	1	0
S_3.*pilot*	0.65	1	0	0

Table 4: The postfix-based inverted index of the segment S_1.

	Postfix-based inverted list
f_1	Null
f_3	Null
f_2	(d_6, 0.5)
f_4	(d_5, 0.52) -> (d_6, 0.5)

Performing the above processing one time will increase 1 of the number of segments. The pilot vectors of the resultant two segments have to be recomputed. However, the resultant two segments remain their *maxf* values unchanged, which are the same with the *maxf* value of the segment before splitting.

2) Decrease the number of segments

Two segments are selected to be merged into one segment. At first, the smallest data segment S_i is selected. Then another segment, denoted as S_c, should be selected to be merged with S_i. Suppose that S_i and S_c are selected to be merged to get the resultant segment denoted as S_{merge}. S_{merge}.*pilot* will be no less than both S_i.*pilot* and S_c.*pilot* on each dimension. A larger value difference on S_{merge}.*pilot* with respect to S_i.*pilot* and S_c.*pilot* implies that the vectors in S_{merge} will have wider range of feature values. It will cause the similarity estimation between segments less effective. Accordingly, the segment whose pilot vector has the smallest distance with S_i.*pilot* is a better candidate to merge with S_i.

In order to reduce the computation cost when selecting the segment S_c, only the

data segment, whose size is less than the average size of all segments, are used as candidates. Besides, the differences between $S_i.pilot$ and $S_c.pilot$ on the features $maxf(S_i)$ and $maxf(S_c)$ are used to estimate the dissimilarity of two segments S_i and S_c as follows:

$dis\_similar(S_i, S_c) =$
$\max(|S_i.pilot[maxf(S_i)]-S_c.pilot[maxf(S_i)]|, |S_i.pilot[maxf(S_c)]-S_c.pilot[maxf(S_c)]|)$.

Accordingly, the candidate segment has the least dissimilarity with S_i is selected to be merged with S_i to get a new segment S_{merge}.

The pilot vector of the resultant segment S_{merge} is updated as follows:

$$S_{merge}.pilot[f_k] = max(S_i.pilot[f_k], S_c.pilot[f_k]) \text{ for } k = 1,\ldots, m.$$

Performing the above processing one time will decrease 1 of the number of segments.

We call the above partitioning method the max-d partitioning (MD) method. In order to prevent generating unbalanced sizes of segments, we proposed an alternative method to set an upper bound constraint on the segments for the MD method, which is called the max-d partitioning with balance constraint (BMD). We will compare the performance of the partitioning method in the experiments.

3.2 Find Intra-segment Similar Pairs

For each data segment S_i, we have to find all the intra-segment similar pairs in S_i. In order to dynamically reduce the cost of similarity computation, an inverted list index is constructed for the data objects in each segment and the prefix filtering strategy [5] is applied.

In our implementation, the boundary of constructing the inverted list for the postfix vector of a data d_j is determined according to the maximum similarity estimated between d_j and $S_i.pilot$. For example, the segment S_1 in the sample dataset has $S_1.pilot$ = <0.6, 0.6, 0.5, 0.3>. The data d_5 in S_1 has feature vector = <0.6, 0.6, 0, 0.52>. Let the similarity threshold t set to be 0.8. The partial inner product results between d_5 and $S_1.pilot$ on the first 3 dimensions, i.e. 0.72, is less than t and on the 4 dimensions is larger than t. Therefore, the prefix vector of d_5 consists of its first 3 dimensions: <0.6, 0.6, 0> and the postfix vector of d_5 consists of the last dimension: <0.3>. Accordingly, the constructed postfix-based inverted index of the segment S_1 is shown in Table 4.

Due to space limit, please refer [5] about the detailed constructing process of inverted list and the prefix filtering strategy. From the sample dataset, the discovered intra-segment similar pairs are (d_5, d_6) and (d_2, d_3).

3.3 Dissimilar segment pairs pruning strategy

A *dissimilar segment pair* (S_i, S_j) means that for each data d in segment S_i and each data d' in segment S_j, $Sim(d, d') < t$. To discover dissimilar segment pairs in early stage can eliminate the similarity computations on the data pairs across the dissimilar segment pairs. Therefore, we estimate the upper bound of similarity between two segments to prune some dissimilar segment pairs.

For each data segments S and S', the similarity between $S.pilot$ and $S'.pilot$ is computed. According to the definition of the pilot vector of a segment, for each d in S and each d' in S', $Sim(d, d') \le Sim(S.pilot, d') \le Sim(S.pilot, S'.pilot)$. Therefore, if $Sim(S.pilot, S'.pilot) < t$, it implies $Sim(d, d') < t$ for each d in S and each d' in S'.

Accordingly, (S, S') is a dissimilar segment pair and is removed from the candidate segment pairs.

For example, according to the partitioning result shown in Table 2 and the pilot vectors of the 3 segments shown in Table 3, the upper bounds of the similarity of the segment pairs are computed as follows:

$Sim(S_1.pilot, S_2.pilot) = 0.6 * 0.45 + 0.5 * 1 = 0.77$
$Sim(S_1.pilot, S_3.pilot) = 0.6 * 0.65 + 0.6 * 1 = 0.99$
$Sim(S_2.pilot, S_3.pilot) = 0.45 * 0.65 = 0.2925$

Suppose that the similarity threshold $t = 0.8$, (S_1, S_2) and (S_2, S_3) are discovered to be dissimilar segment pairs and pruned. Only the segment pair (S_1, S_3) is remained as a candidate segment pair to generate inter-segment data pairs.

To speed up the computation, we also construct an inverted list index for the pilot vectors of segments. The strategy described in 3.2 is used to find similar pilot vector pairs such that the corresponding segment pairs are candidate segment pairs.

3.4 Inter-segment data pairs pruning strategy

Let (S_k, S_l) denote a segment pair where $Sim(S_k.pilot, S_l.pilot) \geq t$, such that the segment pair is remained after performing the pruning strategy introduced in the previous subsection. It is possible to find a data object $d \in S_k$ and a data object $d' \in S_l$ such that (d, d') is a similar pair. To prevent from performing similarity computations for all inter-segment data pairs between S_k and S_l, an efficient pruning strategy is proposed to prune part of the dissimilar objects pairs as follows.

Let $max\_v(S_k)$ denote the maximum feature value contained in the data vectors in segment S_k.$max\_v(S_k)$ is obtained by computing $\{v \mid \exists\ i\ (m \geq i \geq 1),\ v = S_k.pilot[f_i] \wedge \forall\ m \geq j \geq 1 \wedge j \neq i,\ v \geq S_k.pilot[f_j]\}$. For a data object d_x in S_l, if $\|d_x\|_1 < t/max\_v(S_k)$, it implies that $Sim(d_x, d_y) < t$ for all d_y in S_k. Therefore, when generating inter-segment data pairs between S_k and S_l, it is not necessary to generate data pair consisting d_x.

For example, after performing the pruning strategy described in section 3.3 on the sample dataset, the inter-segment data pairs are generated between segments S_1 and S_3. From Table 3, we obtain $max\_v(S_1) = 0.6$ and $max\_v(S_3) = 1$. Accordingly, any data object in S_3 with l^1-norm length less than $0.8/0.6 = 1.3$ is dissimilar with all data objects in S_1. It implies that the data object d_1 can be pruned. Only the data pairs (d_4, d_6) and (d_4, d_5) are generated for further computing the similarities. Finally, the discoverer inter-segment similar pair is (d_4, d_5).

4. A MapReduce Framework for Solving APSS

In this section, we will introduce the proposed MapReduce framework for performing the 4 tasks of the maximum-dimension partitioning approach. At first, the partitioning is performed centralized. After that, the other three tasks can be performed in parallel as described in the following three sub-sections.

4.1 Parallel Processing for Finding Intra-segment Similar Pairs

For each segment, the task of finding the intra-segment similar pairs is performed independently by 2 stages of MapReduce tasks. The first stage is to construct the inverted list index for the data objects in the segment, and the second stage is to perform similarity computation for the intra-segment data pairs.

1) Index construction

<Mapper>:

Input: the set of data objects in the database, where each object has its object id d_i, the assigned segment id S_k, and the list of non-zero feature and value on its vector.

The mappers output the <key: value> pair for each non-zero feature f_j on a vector of object d_i. The feature id f_j is the key. Besides, the feature value, object id d_i, and the segment id S_k are combined to be the value.

For example, the object d_3 is assigned to segment S_2, which has non-zero features f_1=0.45 and f_2=0.9. Two <key: value> pairs are generated: <f_1: (0.45, d_3, S_2)> and <(f_2: (0.9, d_3, S_2)>.

<Reducer>:

The reducers combine the <key: value> pairs with the same key value f_i to generate the inverted list of the feature f_i.

By collecting the results of the reducers, the inverted list of the dataset in a segment is constructed.

2) Find intra-segment similar pairs

<Mapper>:

Input: the inverted list of each feature.

Let the inverted list of a feature f_i be denoted as <(d_{i1}, $d_{i1}.f_i$, $d_{i1}.seg\_id$), (d_{i2}, $d_{i2}.f_i$, $d_{i2}.seg\_id$), …, (d_{im}, $d_{im}.f_i$, $d_{im}.seg\_id$)>. For each d_{ik} and d_{il} in the inverted list, if $d_{ik}.seg\_id$ is equal to $d_{il}.seg\_id$, a key value pair: <(d_{ik},d_{il}): $d_{ik}.f_i \times d_{il}.f_i$> is generated.

<Reducer>:

The reducers combine the <key: value> pairs with the same key value (d_{ik},d_{il}) to compute the sum of the $d_{ik}.f_i \times d_{il}.f_i$ on various feature to get $Sim(d_{ik}, d_{il})$. If $Sim(d_{ik}, d_{il}) \geq t$, ($d_{ik}$, d_{il}) is output as a similar pair with its similarity value.

The similar pairs with similarity value no less than t are intra-segment similar pairs.

4.2 Parallel Processing for Pruning Dissimilar Segment Pairs

This task is to generate the candidate segment pairs (S_i, S_j) which are possible to find inter-segment similar pairs. The MapReduce processing described in 4.1 is used to find the candidate segment pairs, where the pilot vector of each segment is considered a data object. Besides, all their corresponding segment ids are set to be the same.

4.3 Parallel processing for Finding Inter-segment Similar Pairs

After finding the candidate segment pairs (S_i, S_j) which are possible to find inter-segment similar pairs, the strategy introduced in 3.4 is used to prune some inter-segment dissimilar object pairs centralized. Then the following MapReduce approach is used to find the inter-segment similar pairs.

<Mapper>:

Input: the candidate object pairs (d_i, d_j) and the individual non-zero feature values of the two objects. <$f_{i1}=d_i.f_{i1}, f_{i2}=d_i.f_{i2}, \ldots, f_{in}=d_i.f_{in}, f_{j1}=d_j.f_{j1}, f_{j2}=d_j.f_{j2}, \ldots, f_{jn}=d_j.f_{jm}$>.

Output: Generate the key value pairs <(d_i, d_j, f_{ik}): $d_i.f_{ik}$> for k=1…n, and <(d_i, d_j, f_{jk}): $d_j.f_{jk}$> for k=1…m.

< Combiner >:

The key value pairs <(d_i, d_j, f): $d_i.f$> with the same key value are collected, which gets the feature value for both objects in the data pair (d_i, d_j). Accordingly, the values of the same key value are multiplied if both d_i and d_j have non-zero values on f. In addition, the key value pair <(d_i, d_j): $d_i.f \times d_j.f$ > is output.

<Reducer>:

The key value pairs <(d_i, d_j): $d_i.f \times d_j.f$ > with the same (d_i, d_j) pair are collected. Therefore, the summation of the multiplication result on various feature values is computed to get $Sim(d_i, d_j)$. If $Sim(d_{ik}, d_{il}) \geq t$, ($d_{ik}$, d_{il}) is output as a similar pair with its similarity value.

The similar pairs with similarity value no less than t are inter-segment similar pairs.

5. Performance Study

5.1 Experimental environment

The dataset used in the experiments is a real dataset, which consists of the collected documents posted in Yahoo! Answer. The vector of each document is represented by the TF-IDF of the bag of words in the document.

We performed the experiments on the following two types of hardware:

1) A client running Windows7 on 2 cores (2.4GHz), 4 GB RAM.
2) The Hadoop parallelized environment consists of 1 master and 2 slave nodes, where each node running Ubuntu Linux on 2 core (3.4GHz) and 16G memory.

In addition to our approach, the following related works are also implemented for comparison.

(1) Prefix-filtering approach (PF): an inverted-list index structure is constructed for prefix-filtering when computing similarity as the strategy used in [2].
(2) The PSS algorithm (PSS): a partition-based method proposed in [1], where the initial partitioning is based on the descending order of the l^1-norm length of data.

The proposed approach and related works are implemented by Java language.

5.2 Performance Evaluation on the Partitioning Methods

In the first part of experiments, we would like to compare the pruning effectiveness of the partitioning methods. The following three partitioning methods are compared:

1) the maximum dimension partitioning method (MD),
2) the maximum dimension partitioning method with balance constraint (BMD), and
3) the l^1-norm length partitioning (1N) method, where even size partitioning is performed based on the descending order of the l^1-norm length of data. This method is the initial partitioning result of the PSS algorithm [1].

In this part of experiments, a set consisting of 7800 data vectors, where each vector has 11800 dimensional features, was used. In the dataset, there are 299 similarity pairs when the similarity threshold t is set to be 0.8.

[Exp. 1-1] Compare the percentage of similar data pairs discovered from intra-segment data pairs.

The goal of this experiment is to observe the effectiveness that the partitioning methods can group similar pairs in the same segment. Therefore, the percentages of

similar pairs discovered from intra-segment data pairs are computed for the MD, BMD, and 1N methods, respectively. According to the results shown in Fig. 1(a), by using the MD method, about 2/3 similar pairs can be discovered from the intra-segment data pairs when the number of segments is set from 2000 to 3000. It indicates that the MD method can effectively group most similar pairs within segments. Besides, the BMD method keeps more similar pairs within the same segments than the 1N method.

[Exp. 1-2] Compare the percentage of pruned dissimilar segment pairs among all segment pairs.

The goal of this experiment is to observe the effectiveness that the partitioning methods combined with the pruning strategy by computing similarity of pilot vectors to prune dissimilar segment pairs. According to the number of the remained candidate segment pairs, we can compute the percentage of pruned segment pairs among all possible segment pairs as shown in Fig. 1(b). It shows that the BMD method can prune 99.7% segment pairs, which is better than both the MD and 1N methods when the number of segments is less than 3200. It indicates that fewer inter-segment data pairs are generated by using the BMD method.

The pruning effect of the MD method is not as good as the other twos is because that the MD method may generate some large segments whose pilot vectors tend to similar with the pilot vectors of other segments.

[Exp. 1-3] Compare the percentage of pruned data pairs by the partitioning methods combined with the pruning strategies.

In this experiment, in addition to the MD, BMD, and 1N method, we also observe the pruning effectiveness of the PSS method [1]. As the result shown in Fig 1(c), the pruning effectiveness of the MD, BMD, and 1N are consistent with the results shown in Fig. 1(b) because most data pairs are generated from the inter-segment data pairs. It shows the pruning effectiveness of using pilot vector is much better than the strategy proposed in PSS for high dimensional data.

In the second part of experiments, we compared the response time for the proposed methods and the related works.

[Exp. 2-1] Compare the response time of algorithms in the centralized environment by varying the number of segments.

At first, we compare the response time of the three different partitioning methods: MD, SMD, and 1N. As shown in Fig. 1(d), the response time of the MD method is longer than SMD and 1N methods when the number of segments is 2000~3200. The reason is that, as the result shown in Exp. 1-2, the pruning effect of MD is not as good as SMD or 1N when the sizes of segments are not balanced. The response time of SMD method is slightly shorter than 1N because SMD has better pruning effectiveness on data pairs. Besides, the 1N method has additional cost to perform sorting on l^1-norm lengths of data before partitioning. On the other hand, the response time of PSS is much high than SMD because the pruning strategy of PSS didn't work well for high dimensional dataset.

In this experiment, IL has a slightly faster.response time than SMD. It indicates that computing similarity pairs by using the dynamic pruning strategy of inverted list is efficient in a centralized environment. However, in the following experiments, we

will show the benefit of the proposed SMD on larger datasets in a parallel environment.

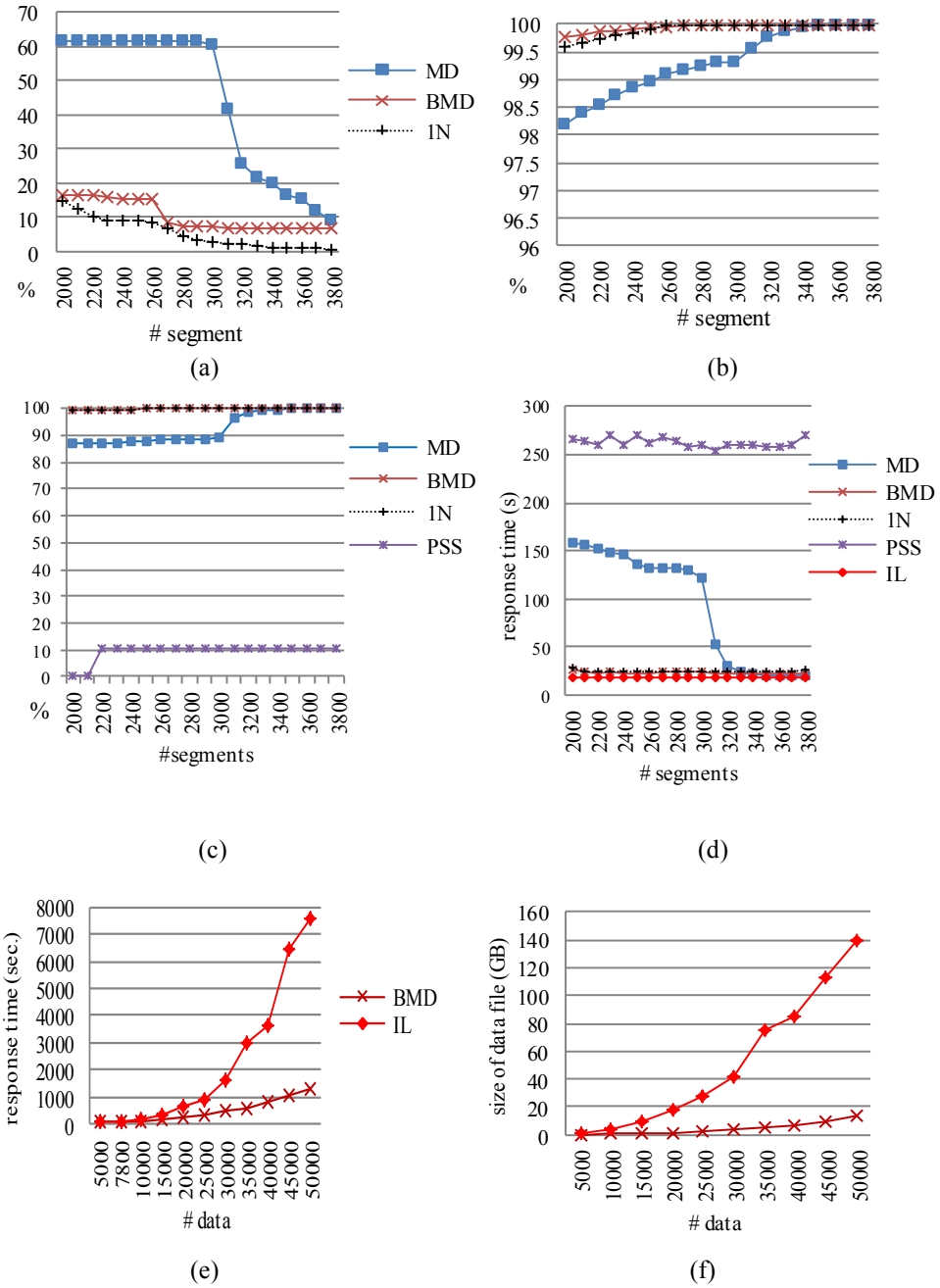

Fig. 1: The results of experiments

[Exp. 2-2] Compare the response time of SMD and IL in the parallelized environment by varying the number of data.

We implemented the MapReduce version for SMD and IL. Another dataset consisting of 49885 data with 30000 dimensions is used. The threshold of similarity is set to be 0.8, where 5047 similar data pairs exist in the dataset.
As the result shown in Fig. 1(e), the response time of BMD grows linearly. However, the growing trend of the response time of IL increases as the size of dataset increases. Therefore, the response time of SMD is about half of the one of IL. The SMD can save much more response time than IL when the size of dataset increases.

Fig. 1(f) shows the size of file storing the <key, value> pairs generated by the Mappers for SMD and IL, respectively. It indicates that the response time of the MapReduce algorithms has strong dependence with the number of the <key, value> pairs generated by the Mappers. As the results shown in the first part of experiments, the SMD method can effectively prune many dissimilar segment pairs to prevent generating the inter-segment data pairs between these segment pairs. Accordingly, in the MapReduce version of SMD, the number of <key, value> pairs generated by the Mapper is much less than the one of IL. It causes less I/O and communication cost among the processors. That is an important effect influencing the response time in the parallel environment.

6. Conclusion and Future Works

In this paper, for solving APSS in parallel, we proposed a new partitioning approach based on the maximum dimension of data vectors. Moreover, a pilot-vector is designed to represent each segment for estimating the upper bound of similarity between each segment pair. The proposed pruning strategy on segment pairs effectively reduces the number of candidate data pairs in an early stage. Besides, the prefix filtering strategy is used to improve the efficiency of computing similarity of both segment pairs and intra-segment data pairs. The results of experiment show the proposed BMD approach improves the weakness of the PSS method performed on high dimensional data sets. Moreover, we implemented a MapReduce framework for the proposed partitioning method. It reveals the benefit of candidate pruning achieved by BMD to effectively reduce the response time. Furthermore, Instead of using a global inverted index to perform computation filtering, this approach prevents the problem of excessive I/O and communication overhead of the intermediate partial results.in a MapReduce framework.

The proposed partitioning method can combine with a load assignment policy to achieve further load balance in a parallel environment. Another future work is to extend the proposed partitioning method to support incremental all pairs similarity search efficiently.

References

[1] M. Alabduljalil, X. Tang and T.Yang, "Optimizing Parallel Algorithms for All Pairs Similarity Search," in Proc. the 6th ACM international conference on Web search and data mining (WSDM), 2013.

[2] D. C. Anastasiu and G. Karypis., "L2AP: Fast Cosine Similarity Search with Prefix L-2 Norm Bounds," in Proc. the 30th IEEE International Conference on Data Engineering (ICDE), 2014.

[3] A. Arasu, V. Ganti and R. Kaushik, "Efficient Exact Set-Similarity Joins," in Proc. the 32nd international conference on Very large data bases (VLDB), 2006.

[4] A. Awekar, N. F. Samatova1 and P. Breimyer, "Incremental All Pairs Similarity Search for Varying Similarity Thresholds with Reduced I/O Overhead," in Proc. the 3rd SNA-KDD workshop, 2009.

[5] R. J. Bayardo, Y. Ma and R. Srikant, "Scaling up All Pairs Similarity Search," in Proc. the 16th international conference on World Wide Web (WWW), 2007.

[6] S. Chaudhuri, V. Ganti and R. Kaushik, "A Primitive Operator for Similarity Joins in Data Cleaning," in Proc. the 24th IEEE International Conference on Data Engineering (ICDE), 2006.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in Proc. OSDI, 2004.

[8] G. D. Francisci, C. Lucchese and R. Baraglia, "Scaling out All Pairs Similarity Search with MapReduce," in Proc. 8th Workshop on Large-Scale Distributed Systems for Information Retrieval (LSDS-IR), 2010.

[9] A. Gionis and P. Indyky, "Similarity Search in High Dimensions via Hashing," in Proc. 25th international conference on Very large data bases (VLDB), 1999.

[10] J. Lin, "Brute Force and Indexed Approaches to Pairwise Document Similarity Comparisons with MapReduce," in Proc. the 32nd international ACM SIGIR conference on Research and development in information retrieval (SIGIR), 2009.

[11] A. Metwally and C. Faloutsos, "V-SMART-Join: A Scalable MapReduce Framework for All-Pair Similarity Joins of Multisets and Vectors," in Proc. VLDB Endowment, Vol. 5, Issue 8, April, 2012.

[12] L. A. Ribeiro and T. Härder, "Efficient Set Similarity Joins Using Min-prefixes," in Proc. the 13th East European Conference on Advances in Databases and Information Systems (ADBIS), 2009.

[13] V. Satuluri, "Bayesian Locality Sensitive Hashing for Fast Similarity Search," in Proc. the VLDB Endowment, Vol. 5, Issue 5, January, 2012.

[14] X. Tang, M. Alabduljalil, X. Jin and Tao Yang, "Load Balancing for Partition-based Similarity Search," in Proc. the 37th international ACM SIGIR conference on Research and development in information retrieval (SIGIR), 2014.

[15] Y. Wang, A. Metwally and S. Parthasarathy, "Scalable All-Pairs Similarity Search in Metric Spaces," in Proc. the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD), 2013.

[16] J. Wang, G. Li and J. Feng, "Can We Beat the Prefix Filtering: An Adaptive Framework for Similarity Join and Search," in Proc. the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD), 2012.