# ClusterJoin: A Similarity Joins Framework using Map-Reduce

Akash Das Sarma*
Stanford University
Palo Alto, CA, 94305
akashds@stanford.edu

Yeye He
Microsoft Research
Redmond, WA, 98052
yeyehe@microsoft.com

Surajit Chaudhuri
Microsoft Research
Redmond, WA, 98052
surajitc@microsoft.com

## ABSTRACT

Similarity join is the problem of finding pairs of records with similarity score greater than some threshold. In this paper we study the problem of scaling up similarity join for different metric distance functions using MapReduce. We propose a ClusterJoin framework that partitions the data space based on the underlying data distribution, and distributes each record to partitions in which they may produce join results based on the distance threshold. We design a set of strong *candidate filters* specific to different distance functions using a novel bisector-based framework, so that each record only needs to be distributed to a small number of partitions while still guaranteeing correctness. To address data skewness, which is common for high dimensional data, we further develop a *dynamic load balancing* scheme using sampling, which provides strong probabilistic guarantees on the size of partitions, and greatly improves scalability. Experimental evaluation using real data sets shows that our approach is considerably more scalable compared to state-of-the-art algorithms, especially for high dimensional data with low distance thresholds.

## 1. INTRODUCTION

Similarity join is the well known problem of finding all pairs of records from a given set that have similarity scores greater than a predefined similarity threshold under a given similarity function (or distance values less than a distance threshold). It is an essential operation in a variety of applications, including data cleaning [11], web page deduplication [16], document clustering [8], plagiarism detection [17], click fraud detection [18], entity resolution [26], data integration [14], etc. As these applications need to handle increasingly vast amounts of data, the problem of scaling up similarity joins is getting ever more important.

Performing similarity joins on massive amounts of data presents two key challenges. First, the data can no longer fit in the memory of one machine, which calls for workload partitioning. Given the pairwise-comparison nature of the problem, partitioning data to ensure *load balancing* while minimizing communication cost and redundancy is difficult. The difficulty of load balancing is further compounded by the need to handle diverse data sets with skewed distributions and high dimensionality. Second, since the number of comparisons needed grows quadratically as data increases, techniques that require comparing all pairs of records do not scale well. So, another challenge is to design *candidate filters* that can prune away a large fraction of candidate pairs without actually computing their similarity. Designing filters to support a large class of useful similarity functions is of great practical importance.

In this work we propose a general framework to compute similarity joins in MapReduce on metric distance functions. While similarity join in MapReduce has been studied [13, 18, 24, 25], most existing approaches focus on set-based or string-based similarity metrics (Jaccard similarity, set-based Cosine similarity, and edit distances). In this work we focus on general metric distance, which represent a much larger class of similarity/distance functions, including Euclidean distance, (vector-based) Cosine similarity, Hamming distance, and a variety of statistical distance functions on data distributions such as Jensen-Shannon distance, Total Variation distance and Earth Mover distance.

Furthermore, we make two key contributions to address the challenges mentioned above. First, we design a general filter that can prune away candidate pairs that are impossible to join given the similarity threshold, without performing the actual comparisons. This general filter works for any metric distance functions. We also develop a set of *bisector-based filters* specific to a number of important distance functions, including Euclidean, Hamming, Total Variation, and other distance functions, that are strictly stronger than the generic filter. Our filters are derived using a bisector-based reasoning, which is a significant departure from the prefix/partition based filters previously developed for set similarity.

Second, we propose a *dynamic load balancing* scheme that is adaptive to data distribution and skewness, with strong probabilistic load balancing guarantees. Our scheme ensures that the size of each partition does not exceed a small factor of the desired threshold with high probability. By ensuring that no worker is overloaded with out-sized data partitions, we avoid "the curse of the last reducer", which greatly improves scalability.

At a very high level, our ClusterJoin framework works in three phases. In the first, sampling phase, we randomly sample data points that we call "anchor points" from the data set, These form centers around which records can be clustered to form partitions. Note that since the sampled data points represent the underlying data distribution by sampling more points in dense regions and less in sparse regions, the space partitioning induced by the set of anchor points tends to partition data evenly. We also sample a separate set of "query point" from the data set and apply our candidate filters to decide whether each query point needs to be mapped to any given anchor partition. This allows us to estimate on the size of

*Work done while visiting Microsoft Research


*Proceedings of the VLDB Endowment,* Vol. 7, No. 12

each anchor partition when the full data set is mapped. For partitions that are estimated to be larger than a predetermined threshold, we use 2-dimensional hashing to ensure that with high probability the partition size does not exceed the desired threshold.

In the second phase, we use previously computed anchor partition centers and apply a set of novel candidate filter rules to process the full data set in parallel. In this step we decide which anchor partitions each record needs to be mapped to in order to ensure that all similarity pairs are discovered.

In the last phase, each machine will work on a separate anchor partition, to perform the pairwise verification of records in the same partition. The union of the results (matching pairs) from all partitions is the output for the similarity join.

We have conducted extensive experiments for a variety of distance functions using real world data sets. Our approach is shown to be especially effective for high dimensional data with high similarity (low distance) thresholds, where it outperforms state-of-the-art approaches by up to an order of magnitude, in terms of both pruning effectiveness of filters and end-to-end runtime. Our algorithm is promising not only because it is effective for a large class of similarity functions, but also it represents an extensible framework that can be tailored to additional metric distance functions. Overall, we believe ClusterJoin is a competitive approach in the complex landscape of performing similarity join using Map Reduce.

# 2. RELATED WORK

The problem of performing efficient similarity joins has a wide variety of applications. Numerous techniques have been proposed, including prefix-based filters [11], All-pairs [6],PP-Join [27], and many others. This long and fruitful line of work has lead to a significant improvement in the scalability of similarity joins.

More recently, similarity join using MapReduce have attracted significant attention, where the goal is to scale to even larger data sets. Vernica et al. [24] are among the first to use ideas from prefix filters and PP join in a MapReduce setting. Their approach is applicable to set-based similarity metrics like Jaccard similarity.

Metwally and Faloutsos [18] propose a V-SMART-Join approach that aggregates the contribution of similarity scores at a token level to compute pairwise similarity. They show that their approach works well for sparse data sets with a large alphabet. Their approach does not prune away any candidate pairs.

Afrati et al. [4] study techniques such as ball hashing and anchor points analytically. Our Cluster-Join algorithm draws inspiration from their anchor points approach. However, their approach can be viewed as *uniform space-partitioning*, which is likely to lead to imbalanced partition with skewed data distributions.

Okcan and Riedewald [21] design a Theta-Join framework that can handle joins for arbitrary predicates. Their approach is very general and is capable of handling any joins. However this approach cannot prune away candidate pairs.

Very recently, Wang et al. [25] develop MAPSS using a distance-based filter that is applicable to any metric distance functions. In comparison, we develop an extensible framework that uses bisectors to design an array of distance specific filters. Combining the more powerful filters with our dynamic load balancing scheme, our approach is experimentally shown to be up to an order of magnitude more efficient than MAPSS.

*Approximate* similarity join (e.g., [22]) is the related problem of discovering similar pairs with a small false negative probability. In this paper, we focus on the *exact* similarity join problem, where all matching pairs are to be found, with no false negatives.

# 3. PRELIMINARIES

## 3.1 Metric distance

In this paper, we focus on metric distance functions. Many widely used distance functions are metric distances, such as Euclidean distance, Angular distance (Cosine similarity), and distribution based distances like Jensen Shannon distance, Total Variation distance and Earth Mover distance, etc. Metric distances have a number of nice properties that we use to design candidate pruning filters.

DEFINITION 3.1 (METRIC DISTANCE [9]). *Let $\mathbf{D}$ be the domain of all records. A metric distance on $\mathbf{D}$ is any function $d : \mathbf{D} \times \mathbf{D} \rightarrow \mathbb{R}$ satisfying the following properties $\forall x, y, z \in \mathbf{D}$:*
- *Non-negativity: $d(x, y) \geq 0$*
- *Coincidence Axiom: $d(x, y) = 0$ iff $x = y$*
- *Symmetry: $d(x, y) = d(y, x)$*
- *Triangle Inequality: $d(x, z) \leq d(x, y) + d(y, z)$*

The framework we propose in this work is generic and can handle any metric distance function, including those mentioned above.

## 3.2 MapReduce

MapReduce [12] is a popular framework for parallel computation. In the MapReduce programming model, data is expressed through (key, value) pairs, and computation is represented by a Map function and a Reduce function.

Map: $(k1, v1) \rightarrow list(k2, v2)$;

Reduce: $(k2, list(v2)) \rightarrow list(k3, v3)$;

More details on this computation framework can be found in [12].

# 4. PROBLEM DEFINITION

The problem of similarity join can be formally stated as follows.

DEFINITION 4.1 (SIMILARITY JOIN). *Let $\mathbf{D}$ be the domain of all data records, we are given a set of records $\mathbf{R} \in 2^{\mathbf{D}}$, a metric distance function $d : \mathbf{D} \times \mathbf{D} \rightarrow \mathbb{R}$, and a distance threshold $d_{thresh}$. Our problem is to find all pairs of records $(R_1, R_2) \in \mathbf{R} \times \mathbf{R}$ that satisfy $d(R_1, R_2) \leq d_{thresh}$. We call this the Similarity-Join$(\mathbf{R}, d(\cdot), d_{thresh})$ problem.*

Note that although we use distance functions and distance thresholds to leverage results established for metric distances, we still refer to this problem as *similarity join* to be consistent with existing literature. Many distance functions actually have direct counterparts in similarity functions (e.g., Angular distance and Cosine similarity), and in such cases the problem above can be alternatively stated as finding all pairs of records with similarity scores above a given threshold. We use similarity functions and distance functions interchangeably in this paper when the meaning is clear from the context.

Similarity join has been used in a variety of applications with diverse data sets and different distance functions. We use a simple running example with Euclidean distance in two dimensional space to illustrate the similarity join problem and our approach – in practice, data records are often of much higher dimensionality and are compared using more complex distance functions.

EXAMPLE 4.1 (SIMILARITY JOIN). *Figure 1 shows a 10-record data set, $\mathbf{R} = \{R_1, \ldots, R_{10}\}$ with $R_1 = (0, 0), R_2 = (1, 5), R_3 = (1, 6), R_4 = (1, 11), R_5 = (4.5, 5), R_6 = (5, 5.5), R_7 = (7, 11), R_8 = (10, 2), R_9 = (10, 10), R_{10} = (12, 0)$. Suppose Euclidean distance function (written below as $d_E(\cdot)$) is used, and the distance threshold is $d_{thresh} = 1$.*

*The output of the problem SimilarityJoin$(\mathbf{R}, d_E, 1)$ is the two pairs $\{(R_2, R_3), (R_5, R_6)\}$ where the distance between each pair of records is less than 1.*

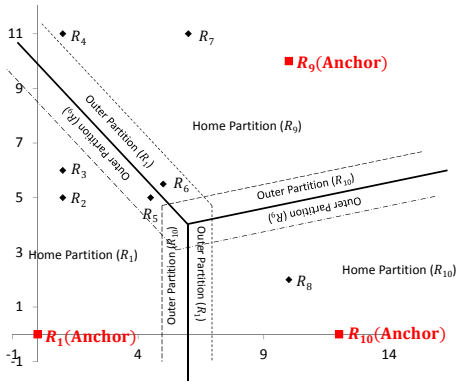



Figure 1: Running example using Euclidean

# 5. DATA PARTITIONING SCHEME

Recall that in order to allow parallel computation, our approach needs to partition the domain of records into regions, in which similar records can be clustered and verified by different machines. Intuitively, we want to produce partitions such that our data is evenly distributed for load balancing purposes.

Given that different data sets have different distributions and are often very skewed in high dimensional space, uniform space partitioning is unlikely to balance data well. For example, uniform rectilinear partition based on value domains is likely to lead to skewed sub-partitions and overloaded workers. Instead, we use a non-uniform space partitioning based approach by sampling the data and allowing the sampled set to determine our space partitioning. By doing so, our partitions capture the underlying data distribution, and lead to a more balanced partitioning scheme.

We randomly sample a set of records $\mathbf{A}$ as what we call "anchor points". These are essentially used as partition centers. We use the set of anchors $\mathbf{A}$ to induce a space partitioning of the domain of data records, by assigning all records to the partition with the closest anchor point.

DEFINITION 5.1 (HOME PARTITION). *Let $C \in \mathbf{A}$ be an anchor record, $\mathbf{D}$ be the domain of records, and $d$ be the distance function. The home partition of $C$, denoted by $H_{\mathbf{A}}(C)$, is defined as $\{R|R \in \mathbf{D}, d(R,C) \leq d(R,A')\forall A' \in \mathbf{A}\}$.*

Note that the choice of anchors $\mathbf{A}$ determines the home partition space. For ease of notation, we drop the subscript from $H_{\mathbf{A}}(C)$ and write $H(C)$ to denote the home partition of $C$ when the underlying set of anchor records is clear from the context.

Intuitively, all records are clustered to the closest anchor point to form partitions. This distance based partitioning scheme can be thought of as similar to Voronoi diagrams, but generalized to arbitrary distance metrics. Also note that each record $R$ can be assigned to a unique home partition by choosing some hash function $h(\cdot)$ and picking the anchor $A$ with the smallest $h(A+R)$ hash value in the case of ties in distance.

EXAMPLE 5.1 (HOME PARTITION). *In our example in Figure 1, suppose $\mathbf{A} = \{R_1, R_9, R_{10}\}$, it is easy to verify that $\{R_1, R_2, R_3, R_5\} \subseteq H(R_1)$, $\{R_4, R_6, R_7, R_9\} \subseteq H(R_9)$ and $\{R_8, R_{10} \subseteq H(R_{10})\}$.*

Note that although *home partitions* group similar records together, it is not sufficient to just join records within the same partition. Records that are outside the partition, but close to the boundary can still join with a record within the partition. We use the notion of *outer partitions* to capture this idea.

DEFINITION 5.2 (OUTER PARTITION). *Let $C \in \mathbf{A}$ be an anchor record, $d$ some distance function, and $d_{thresh}$ the given threshold. The outer partition of $C$, denoted by $O_{\mathbf{A}}(C)$, is defined as $\{R|R \in \mathbf{D}, \exists R' \in H_{\mathbf{A}}(C), d(R,R') \leq d_{thresh}\}$.*

Note that we drop the subscript from $O_{\mathbf{A}}(C)$ and just write $O(C)$ to denote the home partition of $C$ when the underlying set of anchor records is clear from the context.

Intuitively, an outer partition represents the corresponding home partition plus the set of points close to the home partition boundary that can potentially be similar to the points within the home partition. That is, given a threshold $d_{thresh}$, the outer partition of an anchor point is the set of all points within distance $d_{thresh}$ of any point in the corresponding home partition.

Note that while home partitions are disjoint, outer partitions are overlapping, such that each record can belong to several outer partitions. Furthermore, for metric distances, an outer partition is a superset of the corresponding home partition, because $d(R,R) = 0 < d_{threshold}$ by the *coincidence axiom*.

EXAMPLE 5.2 (OUTER PARTITION). *In our example, we have $R_5 \in H(R_1), O(R_1)$ $R_6 \in H(R_9), O(R_9)$. Furthermore $R_5 \in O(R_9)$. By considering the outer partition for $R_9$, the pair ($R_5$, $R_6$) that has a distance $d_E(R_5,R_6) \leq d_{thresh}$ will be compared and produced as output. However, if we only consider home partitions separately, because $R_5 \notin H(R_9), R_6 \notin H(R_1)$, the pair ($R_5$, $R_6$) will be missed.*

In general, it is sufficient to consider each outer partition separately from other outer partitions when comparing pairs of records. This guarantees that no pairs of records satisfying the distance threshold will be missed.

LEMMA 5.1. *Let $\mathbf{A}$ be the set of anchors and $\mathbf{R}$ the set of records. Comparing all pairs of records in the set $\mathbf{R} \cap O(A)$ separately for each $A \in \mathbf{A}$ guarantees that no similar record pairs will be missed.*

PROOF. Consider any pair of records $R_1, R_2$ with $d(R_1,R_2) \leq d_{thresh}$. Let $A$ be the home anchor for $R_2$, that is, $R_2 \in H(A)$. Then by definition $R_2 \in O(A)$. Also, $\exists R_2 \in H(A)$s.t. $d(R_1,R_2) \leq d_{thresh} \Rightarrow R_1 \in O(A)$. Since $R_1, R_2 \in O(A)$, they will be compared and produced in our output. This eliminates the possiblity of false negatives. □

# 6. CANDIDATE FILTERS

While the idea of using home partition and outer partition membership to partition a large data set for similarity join is conceptually clear, deciding the exact outer partition membership remains to be a key technical challenge. Specifically, given a set of anchor points $\mathbf{A}$, and a query point $Q$, our problem is to find the subset of anchors for which $Q$ is an outer partition member. We write this as $M(Q,\mathbf{A}) = \{A|A \in \mathbf{A}, Q \in O(A)\}$.

This is generally very difficult. For example, for Euclidean distance, this is related to computing a generalized Voronoi diagram for $\mathbf{A}$ in high dimensions. Yet even for the well-behaved Euclidean distance function, computing Voronoi diagram is nontrivial and a discipline by itself [20]. Generalizing this to other distance functions that are considerably more complex, such as distributional distances like Jensen Shannon [15] is a daunting task.

Instead of solving the hard problem of finding the *exact* outer partition membership $M(Q,\mathbf{A})$ above, we choose to solve it *approximately*. Specifically, we find a superset $S(Q,\mathbf{A}) \supseteq M(Q,\mathbf{A})$ that is as close to $M(Q,\mathbf{A})$ as possible. By mapping $Q$ to every anchor in $S(Q,\mathbf{A})$, we will still map $Q$ to all the outer partition it needs to be mapped to in the exact solution $M(Q,\mathbf{A})$. Thus we guarantee correctness by not missing any joining pair.

Using this approximate solution results in higher computational and communication costs. The more outer partitions we send our query record to, the higher our cost. For instance, one extreme and trivial approximate solution is to send $Q$ to all anchors such that $S(Q, \mathbf{A}) = \mathbf{A}$. This still guarantees correctness but is apparently inefficient. Our goal is to design filters that can eliminate a large fraction of anchors $A \notin M(Q, \mathbf{A})$ so that the set of remaining anchors $S(Q, \mathbf{A})$ is not much larger than $M(Q, \mathbf{A})$.

## 6.1 Intuition: two anchors at a time

All of our filters discussed in this section are derived using the following intuitive idea. We simplify the problem by only looking at two anchor records at a time. Given the query record $Q$, we first find the anchor record closest to it, denoted by $X$. Given any test anchor record $C$, we want to decide whether $Q$ may be in $C$'s outer partition. Note that to make the problem tractable, we ignore all other anchor records as if $X$ and $C$ are the only anchors present. This greatly simplifies the problem, because the partition boundary when there are only two anchors is simply the bisector plane of the two anchors.

Further, we guarantee correctness by observing that if $Q$ does not need to be mapped to $C$ if there are only two anchors $C$ and $X$, then $Q$ does not need to be mapped to $C$ if there are more anchors. The intuition for this is that every additional anchor added to the system can only remove records from the set forming $C$'s outer partition. Therefore, if $Q$ did not lie in $C$'s outer partition when $X$ was the only other anchor record, then $Q$ will not lie in $C$'s true (smaller) outer partition under the actual set of anchor records.

We formalize the intuition in the lemma below.

LEMMA 6.1 (TWO ANCHOR MEMBERSHIP). *Let $\mathbf{A}$ be a set of anchor records, and $Q$ be any query record. Let $X, C \in \mathbf{A}$ be two anchor records. Let $\mathbf{A}' = \{X, C\}$. Then, $Q \notin O_{\mathbf{A}'}(C) \Rightarrow Q \notin O_{\mathbf{A}}(C)$.*

PROOF. Recall that $Q \in O_{\mathbf{A}}(C)$ denotes that record $Q$ lies in the outer partition of anchor $C$ under the set of anchors $\mathbf{A}$. From the definition of home partitions and outer partitions, we have $Q \notin O_{\mathbf{A}'}(C) \Rightarrow d(Q, J) > d_{thresh} \forall J \in H_{\mathbf{A}'}(C)$. Now, consider any record, $J \notin H_{\mathbf{A}'}(C)$. Then, $J \in H_{\mathbf{A}'}(X)$ (because $J$ has to lie in the home partition of either $C$ or $X$ when they are the only two anchor records). By definition of home partition, $d(X, J) < d(C, J)$.

Since our distance function is invariant of the choice of anchor records, for any $J$, $J \notin H_{\mathbf{A}'}(C) \Rightarrow d(X, J) < d(C, J) \Rightarrow J \notin H_{\mathbf{A}}(C)$. That is, $J \notin H_{\mathbf{A}}(C) \forall J \notin H_{\mathbf{A}'}(C)$. Taking the contrapositive of the previous statement, we have, $J \in H_{\mathbf{A}'}(C) \forall J \in H_{\mathbf{A}}(C)$. Therefore, $d(Q, J) > d_{thresh} \forall J \in H_{\mathbf{A}'}(C) \Rightarrow d(Q, J) > d_{thresh} \forall J \in H_{\mathbf{A}}(C) \Rightarrow Q \notin O_{\mathbf{A}}(C)$ (again by invariance of distance with respect to anchor set). Hence, $Q \notin O_{\mathbf{A}'}(C) \Rightarrow Q \notin O_{\mathbf{A}}(C)$. This completes our proof. □

Using this idea, we derive filter rules which, given the input query record $Q$ and its nearest anchor record $X$, test each anchor point $C$ in the absence of all other anchor points for $Q$'s membership. We eliminate all anchor records satisfying our filter rules and map $Q$ to the remaining anchor points, which is guaranteed to be a superset of $M(Q, \mathbf{A})$.

## 6.2 General filter for any metric distance

We now give a generic filter rule that holds true for all metric distance functions. As discussed previously, we simplify the problem without losing correctness by considering two anchors at a time.

THEOREM 6.1 (GENERIC FILTER RULE). *Let $Q$ be an input query record, $X$ be its nearest anchor, and $C$ be any test anchor as in Figure 2. Let $x = d(X, Q), c = d(C, Q)$. If $c > x + 2d_{thresh}$, then $Q \notin O(C)$.*

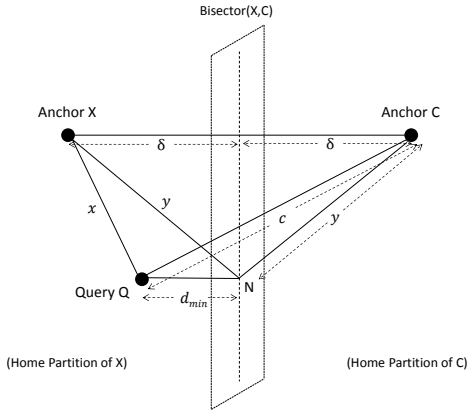



Figure 2: Generic Filter

PROOF. Let $N$ be the nearest point to $Q$ that is equidistant from points $X$ and $C$, that is, $d(X, N) = d(C, N)$ and $d(Q, N) \geq d(Q, N') \forall N'$s.t. $d(X, N') = d(C, N')$. Let $d(Q, N) = d_{min}$. Since $N$ is the closest point to $Q$ on the bisector plane of $X, C$, we have $d_{min} \leq d(Q, I) \forall I \in H(C)$ (because if $QI$ intersects the bisector plane at $J$, we have $d_{min} \leq d(Q, J) \leq d(Q, I)$).
Let $d(X, N) = d(C, N) = y$ ($N$ lies on bisector plane of $X, C$). By triangle inequality on triangles $XQN$ and $CQN$ respectively, $y \leq x + d_{min}$ and $y \geq c - d_{min}$. Combining these, we have $c \leq x + 2d_{min}$. So, if $c > x + 2d_{thresh}$, we have $x + 2d_{min} > x + 2d_{thresh} \Rightarrow d_{min} > d_{thresh}$. Therefore, if $c > x + 2d_{thresh}$, then $d(Q, I) \geq d_{min} > d_{thresh} \forall I \in H(C)$. Thus $Q \notin O_{\{X,C\}}(C)$. Using Lemma 6.1, $c > x + 2d_{thresh} \Rightarrow Q \notin O(C)$. □

We note that although we describe $X$ as the nearest anchor to $Q$, our filter rule holds true even if $X$ is an *approximate* nearest neighbor. We omit the full argument but note that this property can be used to improve efficiency when used in conjunction with Min-Hash like schemes.

## 6.3 Distance function specific filters

In this section, we look at individual distance functions and construct distance-specific filters. In particular, we have designed filters for Euclidean distance, Total Variation distance, 1-norm distance, Hamming distance, which we will describe in this section. We have also designed filters for Earth Mover distance, and $L_p$ distance in certain scenarios, which we will present in the full version of this paper in the interest of space. All these specific filters have strictly stronger pruning powers than the generic filter. Note that all the distance functions we will discuss are metric distances [9].

Like the general filter, our special filters also rely on Lemma 6.1 to consider only two anchors $X, C$ at a time to make the problem tractable. Let $B(X, C) = \{P | P \in \mathbf{D}, d(P, X) = d(P, C)\}$ be the bisector of $X, C$. The key insight for stronger filters is that when given a specific distance function, we can compute tighter lower bounds for the minimum distance from $Q$ to $B(X, C)$, denoted as $d_{min}(Q) = \min\{d(P, Q) | P \in B(X, C)\}$. We also write this as just $d_{min}$ when the context is clear.

### 6.3.1 Euclidean Distance

DEFINITION 6.1 (EUCLIDEAN DISTANCE). *The Euclidean distance between two points $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ is $d_E(A, B) = \sqrt{\sum_i (a_i - b_i)^2}$.*

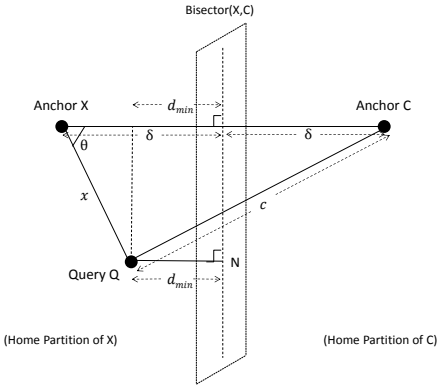



Figure 3: Filter for Euclidean Distance

Euclidean distance is a metric distance and therefore the generic filter rule applies. Here, we rely on characteristics of Euclidean distance to compute the exact minimum distance to bisector, which gives us a stronger filter rule.

THEOREM 6.2 (EUCLIDEAN FILTER). *Let $Q$ be an input query record, $X$ be its nearest anchor, and $C$ be any test anchor as in Figure 3. Let $\delta = \frac{d_E(X,C)}{2}, x = d_E(X,Q), c = d_E(C,Q)$. If $c^2 > x^2 + 4\delta d_{thresh}$, then $Q \notin O(C)$.*

PROOF. As before, let $N$ be the nearest point to $Q$ that is equidistant from points $X$ and $C$, that is, $d_E(X,N) = d_E(C,N)$ and $d_E(Q,N) \geq d_E(Q,N')\forall N'$s.t. $d_E(X,N') = d_E(C,N')$. Let $d_E(Q,N) = d_{min}$. By triangle inequality, $d_{min} \leq d_E(Q,I)\forall I \in H(C)$. Observe that $d_{min} = \delta - x\cos\theta$. Note that $d_{min}$ is the shortest distance from $Q$ to the bisector plane of $X$ and $C$, which is equivalent to their partition boundary in the absence of other anchor records. From triangle $XQC$, we have $c^2 = x^2 + (2\delta)^2 - 2x(2\delta)\cos\theta$. Simplifying, we obtain $\cos\theta = \frac{x^2+(2\delta)^2-c^2}{2(2\delta)x}$. Now, $d_{min} = \delta - x\cos\theta = \frac{c^2-x^2}{4\delta} \Rightarrow c^2 - x^2 = 4\delta(\delta - x\cos\theta)$. Therefore, if $c^2 > x^2 + 4\delta d_{thresh} \Rightarrow \delta - x\cos\theta > d_{thresh} \Rightarrow d_{min} > d_{thresh}$. Using $d_{min} \leq d_E(Q,I)\forall I \in H(C)$, we have $d_E(Q,I) > d_{thresh}\forall I \in H(C) \Rightarrow Q \notin O_{\{X,C\}}(C)$. Using Lemma 6.1, $Q \notin O_{\{X,C\}}(C) \Rightarrow Q \notin O(C)$. Therefore, $c^2 > x^2 + 4\delta d_{thresh} \Rightarrow Q \notin O(C)$. □

### 6.3.2 Total Variation Distance (TVD)

DEFINITION 6.2 (TOTAL VARIATION DISTANCE). *Let $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ be two discrete probability distributions ($\sum_{i=1}^n a_i = 1, \sum_{i=1}^n b_i = 1$). The Total Variation Distance between $A, B$ is $d_{TVD}(A,B) = \frac{1}{2}\sum_{i=1}^n |a_i - b_i|$.*

The Total Variation Distance (TVD) between two probability distributions can be thought of as the largest possible difference between probabilities that the distributions can assign to any given event. Total Variation Distance is a metric distance function.

For the TVD function, we describe how to efficiently compute a lower bound for the distance of any record $Q = (q_1, q_2, \ldots, q_n)$ to the bisector plane of any two other records $A = (a_1, a_2, \ldots, a_n)$, $B = (b_1, b_2, \ldots, b_n)$. The intuition behind our approach is as follows. Given the three distributions (points) $Q, A, B$, where $Q$ is closer to $A$ than $B$, we wish to compute the distance from $Q$ to the bisector of $A, B$. That is, we wish to find the shortest path from $Q$ to any point $N$ such that $d_{TVD}(A,N) = d_{TVD}(B,N)$. Initially we have $d_A = d_{TVD}(A,Q) < d_{TVD}(B,Q) = d_B$. We change the probability values in $Q$ to take $Q$ closer to $B$ and farther away from $A$, thereby reducing $d_B$ and increasing $d_A$. Note that during this process, we have to be careful to maintain the constraints $\sum_{i=1}^n q_i = 1$ and $0 \leq q_i \leq 1$, therefore every increase in one $q_i$ requires a corresponding decrease in another $q_j$. Repeating this process, we eventually reach a point when $d_A = d_B$. This means that we have reached a point on the bisector of $A, B$. The total distance moved in the transformation from $Q_{old}$ to $Q_{new}$, which can be measured by the sum of the changes made to different $q_i$, is the length of this particular path from $Q$ to the bisector.

Our approach involves varying $q_i$ to take $Q$ closer to $B$ and further away from $A$ with minimum total changes to different $q_i$s. This gives us the shortest path to the bisector. We make use of a few key observations to lower bound the length of the shortest such path.

OBSERVATION 6.1 (INDEPENDENCE OF DIMENSIONS). *Let $Q = (q_1, q_2, \ldots, q_n), A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$. Changing $q_i$ only affects the contribution of the $i^{th}$ coordinate to the distances $d_A$ and $d_B$, and is independent of other coordinates.*

This follows trivially from the definition of TVD that $d_{TVD}(A,B) = \frac{1}{2}\sum_{i=1}^n |a_i - b_i|$.

OBSERVATION 6.2 (VARYING IN A SINGLE DIRECTION). *The shortest path to the bisector of $A, B$ from $Q$ will only include changes of $q_i$ in a singe direction for any $i$.*

This can be easily proven by contradiction. Suppose the shortest path to the bisector from $Q$ involved both an increase and a decrease of $q_i$ for some $i$. Then, consider two steps where $q_i$ was respectively increased and decreased by $\delta$. By not changing $q_i$ in either of these steps, we reduce the distance of our path by $2\delta$, while still reaching the same point, thereby giving a shorter path.

OBSERVATION 6.3 (PARTITIONING DIMENSIONS). *Let $Q = (q_1, q_2, \ldots, q_n)$, $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$. Consider the following partitioning or classification on the dimensions $i \in [1, n]$:*

***1(a):*** $q_i > a_i > b_i$, ***1(b):*** $q_i < a_i < b_i$
***2(a):*** $a_i \geq q_i \geq b_i$, ***2(b):*** $a_i \leq q_i \leq b_i$
***3(a):*** $q_i > b_i \geq a_i$, ***3(b):*** $q_i < b_i \leq a_i$

As before, let $d_A = d_{TVD}(A,Q)$ and $d_B = d_{TVD}(B,Q)$. Observe that the change in $d_B - d_A$ when varying $q_i$ only depends on which of the above classes $i$ belongs to. This follows from our previous observation about the independence of dimensions, and the fact that $d_B - d_A = \frac{1}{2}\sum_i(|b_i - q_i| - |a_i - q_i|)$.

First observe that increasing $q_i$ in class 3(a) has no effect on $d_B - d_A$, while decreasing $q_i$ for this class initially has no effect, and then only increases $d_B - d_A$. Similarly, decreasing $q_i$ in class 3(b) has no effect on $d_B - d_A$, while increasing $q_i$ will potentially increase $d_B - d_A$. Therefore, the shortest path to the bisector will not include any changes to $q_i$ where $i$ belongs to class 3.

Now, consider varying $q_i$ for $i$ in class 2. If $i$ is of type 2(a), then every $\delta$ decrease in $q_i$ until it reaches $b_i$ results in a corresponding decrease in $d_B - d_A$ by $\delta$ (because $\sum_i |b_i - q_i|$ decreases by $\delta$ and $\sum_i |a_i - q_i|$ increases by $\delta$). Decreasing $q_i$ any further will result in no change in $d_B - d_A$. Let $m_i = q_i - b_i$ be the maximum possible change of this nature. Increasing $q_i$ for this case is unfavourable and will not be a part of the shortest path as it increases $d_B - d_A$, thereby moving away from the bisector.

Similarly, if $i$ is of type 2(b), then every $\delta$ increase in $q_i$ until it reaches $b_i$ results in a corresponding $\delta$ decrease in $d_B - d_A$, and $m_i = b_i - q_i$ is the maximum possible change of this type.

Finally, consider varying $q_i$ where $i$ is in class 1. In class 1(a), increasing $q_i$ has no effect on $d_B - d_A$. Decreasing $q_i$ until it reaches $a_i$ does not change $d_B - d_A$ either, but it is still a potential

move to consider, because any further $\delta$ decrease $q_i$ now results in a corresponding $\delta$ decrease in $d_B - d_A$ similar to case 2(a). Let $c_i = q_i - a_i$ be the amount of distance required to reach the favorable moves, and $m_i = a_i - b_i$ be the maximum subsequent favorable change possible.

Similarly, in class 1(b), increasing $q_i$ by $c_i = a_i - q_i$ has no immediate effect on $d_B - d_A$, but every subsequent change in $q_i$ up to a maximum of $m_i = b_i - a_i$ results in a corresponding and equal change in $d_B - d_A$.

Using our above observations, we now construct an optimization problem whose solution is the length of the shortest path from $Q$ to the bisector of $A, B$. We refer to a change that decreases $d_B - d_A$ as a favorable change. Let $P^+ = \{i | i \in$ Class 1(a)$\cup$Class 2(b)$\}$ be the set of coordinates $i$ where it is favourable to increase $q_i$. Similarly, $P^- = \{i | i \in$ Class 1(b)$\cup$Class 2(a)$\}$ is the set of coordinates $i$ where it is favourable to decrease $q_i$. The idea here (following from Observation 6.3) is that the shortest path will only consist of simultaneous increases of magnitude $\delta$ in $q_i$ for some $i \in P^+$ and decreases of magnitude $\delta$ for some $j \in P^-$ respectively.

Let $c_i$ be the cost to reach a favorable position, and $m_i$ be the maximum possible favorable change for classes 1 and 2. For $i \in$ Class 1, we have $c_i = |q_i - a_i|$ and $m_i = |a_i - b_i|$. For $i \in$ Class 2, $m_i = |b_i - q_i|$. Note that Class 2 positions only have favorable moves right from the start, and so $c_i = 0$. Class 3 positions are not considered since they do not bring any favorable changes. Let $t = \frac{1}{2}\sum_{i=1}^{n}(|b_i - q_i| - |a_i - q_i|)$ be the total change required in $d_B - d_A$ as we move $Q$ to reach the bisector. Using these, we define the following optimization problem that minimizes the distance to the bisector.

$$\text{(TVD)} \quad \min \sum_{i\in P^+} u_i c_i + \sum_{j\in P^-} v_j c_j + \sum_{i\in P^+} x_i + \sum_{j\in P^-} y_j \tag{1}$$

$$\text{s.t.} \sum_{i\in P^+} x_i + \sum_{j\in P^-} y_j \geq t \tag{2}$$

$$\sum_{i\in P^+} x_i = \sum_{j\in P^-} y_j \tag{3}$$

$$0 \leq x_i \leq m_i, \forall i \in P^+ \tag{4}$$

$$0 \leq y_j \leq m_j, \forall j \in P^- \tag{5}$$

$$u_i \geq x_i, \forall i \in P^+ \tag{6}$$

$$v_j \geq y_j, \forall j \in P^- \tag{7}$$

$$u_i, v_j \in \{0, 1\}, \forall i, j \tag{8}$$

In the above optimization problem, $x_i$ denotes the number of favorable moves for each position $i \in P^+$ (possibly after moves with no effect), similarly $y_j$ denotes the number of favorable moves for each position $j \in P^-$. Let $u_i, v_j$ be integral variables, denoting whether the cost of moves with no effect has occurred at position $i \in P^+$ or $j \in P^-$, respectively, just so that the corresponding favorable moves can happen. The objective function is the sum of the total costs of favorable moves $x_i$, $y_j$, and no-effect moves where $u_i, v_j$ are weighted by $c_i$ and $c_j$ which are the costs of no-effect moves at position $i$ and $j$ respectively.

The constraint in Equation (2) makes sure that the total favorable moves is sufficient for the reduction of difference of distance to $A$ and $B$ is at least $t$ (reaching the bisector). Equation (3) ensures that the total move in $P^+$ and $P^-$ is balanced such that $Q$ is still a valid probability distribution. Equation (4) (5) indicates that favorable moves cannot exceed the maximally allowed $m_i$ and $m_j$. Equation (6) (7) (8) ensures that if either $x_i$ or $y_j$ is greater than 0, then the initial cost of no-effect moves is added, by forcing $u_i$, $v_j$ respectively to 1.

This LP gives the exact minimum distance to the bisector. However, it is a mixed integer program that is in general intractable. So, we present an alternative linear program TVD-F, whose solution gives us a lower bound to the solution of the original optimization problem, and therefore, a lower bound to the distance to the bisector. Using the same variables as before, we define the following fractional LP TVD-F, which is easy to solve.

$$\text{(TVD-F)} \quad \min \sum_{i\in P^+} x_i(\frac{c_i + m_i}{m_i}) + \sum_{j\in P^-} y_j(\frac{c_j + m_j}{m_j}) \tag{9}$$

$$\text{s.t.} \sum_{i\in P^+} x_i + \sum_{j\in P^-} y_j \geq t \tag{10}$$

$$\sum_{i\in P^+} x_i = \sum_{j\in P^-} y_j \tag{11}$$

$$0 \leq x_i \leq m_i, \forall i \in P^+ \tag{12}$$

$$0 \leq y_j \leq m_j, \forall j \in P^- \tag{13}$$

Intuitively, this linear program assumes that the initial set-up cost of the no-effect moves, $c_i$ is distributed evenly across the favorable moves. So now each favorable move of magnitude $\delta$ at position $i$ incurs a cost of $\frac{c_i+m_i}{m_i}\delta$, and similarly for position $j$. It can be shown that the optimal value of TVD-F is at least as good as (no greater than) that of TVD.

LEMMA 6.2 (LP LOWER BOUND). *Let $OPT$ be the optimal value of LP TVD, and $OPT^f$ be the optimal value of TVD-F. We have $OPT^f \leq OPT$.*

PROOF. Let $u_i^*, v_j^*, x_i^*, y_j^*$ be the values of variables when the optimal value $OPT$ is achieved for TVD.

First, it can be verified that the same $x_i^*, y_j^*$ is also a feasible solution to TVD-F because the constraints in TVD-F is a subset of those in TVD. So we only need to show $u_i^* c_i + x_i^* \geq \frac{c_i+m_i}{m_i} x_i^*$ and similarly $v_j^* c_j + y_i^* \geq \frac{c_j+m_j}{m_j} y_j^*$, since the summation of the left side and the right side are the optimal value of TVD and one objective value of TVD-F, respectively. Note that these equations can be rewritten as $u_i^* c_i \geq c_i \frac{x_i*}{m_i}$ and $v_j^* c_j \geq c_j \frac{x_j*}{m_j}$, both of which are true given our constraints. Thus $OPT = u_i^* c_i + x_i^* + v_j^* c_j + y_i^* \geq \frac{c_i+m_i}{m_i} x_i^* + \frac{c_j+m_j}{m_j} y_j^* \geq OPT^f$. □

Note that not only is the TVD-F easy to solve, but it can also be greedily solved without using an LP-solver, by sorting $\frac{c_i+m_i}{m_i}$ for $i \in P^+$ and $\frac{c_j+m_j}{m_j}$ for $j \in P^-$ ascendingly, then picking position $i$ and $j$ simultaneously to maintain balance (Equation (11)), while gaining in Equation (10), until Equation (10) is satisfied.

Using this we derive a filter for TVD.

THEOREM 6.3 (TVD FILTER). *Let $Q$ be an input query record, $X$ be its nearest anchor, and $C$ be any test anchor. Use $X$ as $A$ or the point closer to $Q$, and $C$ as $B$ or the point further away from $Q$ to formulate TVD-F. Let $OPT^f$ be the solution to our fractional linear program TVD-F. Then, $OPT^f > d_{thresh} \Rightarrow Q \notin O(C)$.*

PROOF. First we know $OPT^f \leq OPT = d_{bisector}$ by Lemma 6.2. Using Lemma 6.1, similar to our previous specific filters, we know that if $d_{bisector} > d_{thresh} \Rightarrow Q \notin O(C)$. Combining, we get $OPT^f > d_{thresh} \Rightarrow d_{bisector} > d_{thresh} \Rightarrow Q \notin O(C)$. □

### 6.3.3 1-Norm Distance

DEFINITION 6.3 (1-NORM DISTANCE). *The 1-norm distance between two points $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ is given by $d_1(A, B) = ||A - B||_1 = \sum_{i=1}^{n} |a_i - b_i|$.*

Observe that 1-norm distance appears to be very similar in structure to Total Variation Distance. In particular, the Observation 6.1 and 6.2 for TVD still hold. However the lower bound of distance to bisector for 1-norm distance is different. The difference arises from the fact that for TVD, data points $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ are probability distributions that constrained by $\sum_i a_i = \sum_i b_i = 1$, whereas for 1-norm distance the points are unconstrained.

To solve the problem of finding the distance to the bisector of two points $A = (a_1, a_2, \ldots, a_n), B = (b_1, b_2, \ldots, b_n)$ from a third given point, $Q = (q_1, q_2, \ldots, q_n)$, we again partition the coordinates $[n]$ in three classes as we did in Observation 6.3 for TVD.

Overall, we define for $i \in$ Class (1) $c_i = |q_i - a_i|$ and $m_i = |b_i - a_i|$. For $i \in$ Class (2), $c_i = 0$ and $m_i = |b_i - q_i|$. Let $t = \sum_{i=1}^{n} (|b_i - q_i| - |a_i - q_i|)$ be the total change of distance required in $d_B - d_A$ as we move $Q$ to reach the bisector. Using these, we define the following optimization problem that minimizes the distance to the bisector.

$$\text{(1Norm-F)} \qquad \min \sum_{i \in P} x_i \left(\frac{c_i + m_i}{m_i}\right) \tag{14}$$

$$\text{s.t.} \sum_{i \in P} x_i \geq \frac{t}{2} \tag{15}$$

$$0 \leq x_i \leq m_i, \forall i \in P \tag{16}$$

THEOREM 6.4 (1-NORM FILTER). *Let $Q$ be an input query record, $X$ be its nearest anchor, and $C$ be any test anchor. Use $X$ as $A$ or the point closer to $Q$, and $C$ as $B$ or the point further away from $Q$ to formulate 1Norm-F. Let $d_{min}$ be the solution to 1Norm-F. Then, $d_{min} > d_{thresh} \Rightarrow Q \notin O(C)$.*

A proof of this theorem can be derived similar to TVD.

### 6.3.4 Hamming Distance

DEFINITION 6.4 (HAMMING DISTANCE). *The Hamming distance between two points $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ is the number of positions that $A$ and $B$ differ on. It is given by $d_H(A, B) = |\{i | i \in [n], a_i \neq b_i\}|$.*

We note that the Independence of Dimensions in Observation 6.1, and Varying in a Single Direction in Observation 6.2 still hold for Hamming distance. However, dimensions need to be partitioned differently.

OBSERVATION 6.4 (PARTITIONING DIMENSIONS). *Let $Q = (q_1, q_2, \ldots, q_n)$, $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$. Consider the following partitioning or classification on the dimensions $i \in [1, n]$ such that:*

***1:*** *$a_i = b_i$*
***2:*** *$q_i = a_i \neq b_i$*
***3:*** *$q_i = b_i \neq a_i$*
***4:*** *$q_i, a_i, b_i$ are pairwise distinct*

Very briefly, the idea here is that changing $q_i$ in Class 2 will bring $Q$ fastest toward $B(X, C)$, while changing $q_i$ in Class 4 also helps but at a slower rate. Using ideas similar to the TVD filter, we get the following filter.

THEOREM 6.5 (HAMMING FILTER). *Let $Q$ be an input query record, $X$ be its nearest anchor, and $C$ be any test anchor. Let $t = d_H(C, Q) - d_H(X, Q)$, and let $P_2 = \{i | q_i = x_i \neq c_i\}$. Then, $\max\{\lceil \frac{t}{2} \rceil, t - |P_2|\} > d_{thresh} \Rightarrow Q \notin O(C)$.*

Using similar ideas, we have also designed filters for other distance functions such as Earth Mover Distance, $L_p$ distance in certain scenarios, and Euclidean distance in generalized high dimensional space. Details of these additional filters can be found in the full version of this paper.

## 6.4 Filter Strength

Before we conclude this filter section, we note that the strength of specific filters for Euclidean, TVD, 1-norm and Hamming discussed earlier is strictly stronger than the generic filter. We omit details of the proof in the interest of space.

THEOREM 6.6 (FILTER STRENGTH). *Given query point $Q$, its closest anchor $X$, and the test anchor $C$. Let GEN $(X, Q, C)$ denote whether we can use the generic filter to prune away $C$ as in Theorem 6.1. Let BISECTOR_FILTER$^d(X, Q, C)$ denote whether we can use specific filters for distance $d$ to prune away $C$, where $d \in \{$Euclidean, TVD, 1-norm, EMD, Hamming$\}$. Then, we have $GEN(X, Q, C) \Rightarrow$ BISECTOR_FILTER$^d(X, Q, C)$.*

# 7. OUR APPROACH: CLUSTERJOIN

After describing the partitioning scheme in Section 5 and candidate filters in Section 6, we are now in a position to introduce the ClusterJoin algorithm.

Our algorithm consists of three main phases: (1) Sampling phase. In this step, we randomly sample records as anchor points. We further sample query points to estimate the expected size of anchor partitions to decide if an overloaded partition needs to be split. (2) Mapping phase. We apply the appropriate filter rule to map each data point to an appropriate subset of anchor partitions. (3) Verification. Here, we verify records in each outer partition by performing pairwise comparisons, to compute the final output.

We describe each phase of our algorithm in turn.

## 7.1 The sampling phase

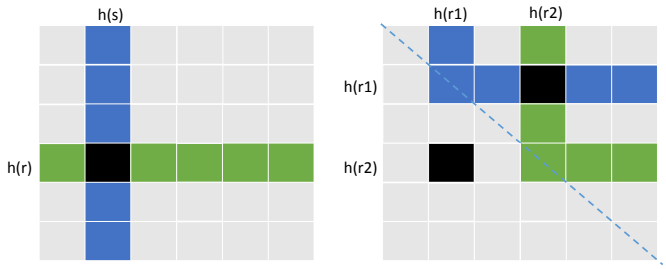
In this step, we randomly sample anchor points, $\mathbf{A}$, from the data set $\mathbf{R}$ with probability $p_A$. Note that using data samples to partition the data space tends to distribute data evenly across home partitions, because the sampled anchors represent the underlying data distribution with more anchors in dense regions and fewer in sparse regions. We also simultaneously sample a separate set of query points, $\mathbf{Q}$, from $\mathbf{R}$ with probability $p_Q$, which is used to estimate the expected size of each anchor partition when using the full data set.

The samples $\mathbf{A}$ and $\mathbf{Q}$ are sent to a common reducer, where we use candidate filters to decide which anchor partitions each query point needs to be sent to. Let query points distributed to anchor $A$ be $S(A, \mathbf{Q}) = \{Q | Q \in \mathbf{Q}, Filter(Q, A) = false\}$. Then $e_A = \frac{|S(A,\mathbf{Q})|}{p_Q}$ gives us an estimate of the size of each anchor partition when the full data set is mapped.

Note that for large $\mathbf{A}$ and $\mathbf{Q}$, this estimation can be easily parallelized, by broadcasting $\mathbf{A}$ to all reducers, and hash partitioning $\mathbf{Q}$ to all reducers, so that each reducer only produces estimates for a chunk of $\mathbf{Q}$. The estimates for each anchor can then be aggregated using another MapReduce round.

Based on the available computation resource of each machine and the cost of computing similarity between records, we can predetermine a threshold $T$, say 1000 records, that the estimated size of each anchor partition $e_A$ should not exceed. We split partitions that are estimated to be larger than $T$, to ensure load balancing, using the 2-dimensional hash partitioning idea in [21] that is reminiscent of Blocked Nested Loop Join.

**2D hashing**. The idea of 2D hashing can be illustrated pictorially. Figure 4a demonstrates the case of an R-S join. In this case we can created $k^2$ cells in a 2D matrix. For each $r \in R$ we produce an integral hash value $h(r) \in [k]$, and map $r$ into all cells in row number $h(r)$. We produce hash values similarly for each $s \in S$ and map $s$ into all cells in column number $h(s)$. Each pair of $(r, s)$ will be hashed to the same cell exactly once regardless of their hash values.




(a) R-S Join (b) Self Join

Figure 4: 2D hashing illustration

Figure 4b shows a similar hashing scheme adapted for self-join, where we essentially only consider the upper right half of the matrix by reflecting it along its diagonal. Let $C(r)$ be the set of cells that $r$ needs to be mapped to as indexed by $(row, col)$ in this hashing scheme. Then, we have $C(r) = \{(i, h(r))|\ 0 < i \leq h(r)\} \bigcup \{(h(r), j)|\ h(r) < j \leq k\}$. Note that in the self-join case, pairs with the same hash value will only be compared in diagonal cells.

It can be seen that when using 2D hashing for a data set of size $|R|$, each sub-partition (cell) has an expected size of $\frac{2|R|}{k}$.

**Partition splitting**. Given that the estimated size of each partition is $e_A = \frac{|S(A,\mathbf{Q})|}{p_Q}$, if $e_A > T$, we can use 2D hash partition to further sub-partition each over-sized anchor partition. The 2D hashing factor $k$ required to meet the threshold is $k = \lceil \frac{2e_A}{T} \rceil$.

EXAMPLE 7.1 (SAMPLING). *Recall our example in Figure 1 with 10 records and Euclidean distance. Suppose the anchor sampling rate $p_A = 0.3$ and are $R_1$, $R_9$ and $R_{10}$ are selected as anchors, or $\mathbf{A} = \{R_1, R_9, R_{10}\}$. Suppose the query sampling rate $p_Q$ is also 0.3, such that the query set is $\mathbf{Q} = \{R_2, R_6, R_8\}$.*

*By applying the Euclidean filter rule on $\mathbf{Q}$, we know that $O(R_1)$ receive two records $\{R_2, R_6\}$, whereas the other two partitions $O(R_9)$ and $O(R_{10})$ receive one record each ($R_6$ and $R_8$ respectively).*

*Suppose the size threshold of each partition is $T = 5$. The estimated size of $O(R_1)$ is $\frac{2}{0.3} > 5$, whereas the estimated size of $O(R_9)$ and $O(R_{10})$ is $\frac{1}{0.3} < 5$. Thus, $O(R_1)$ will be the only partition that needs a 2D hashing split.*

Using the partition size estimate $e_A$ and 2D hashing on partitions whose estimate exceeds the set threshold, we can guarantee that each real partition will not exceed the threshold with high probability.

THEOREM 7.1. *Let the sampling rate of query points be $p = \frac{c}{T}$, and $n$ be the partition size of any anchor partition after the complete data set is mapped. By using the partition splitting procedure described above, we can guarantee $P(n \geq 4T) \leq e^{-c}$. That is, the probability that the real partition size is 4 times larger than threshold $T$ is exponentially small as $c$ increases.*

PROOF. Given any (outer) anchor partition $A$, $n$ is the true size of $A$ when the full data set is mapped. $p$ is the query point sampling rate. Let $X_i$ be the binary random variable used to denote whether data record $R_i \in A$ is sampled in the query point set $\mathbf{Q}$ or not. That is, $X_i = 1$ if $R_i \in \mathbf{Q}$ and $X_i = 0$ if $R_i \notin \mathbf{Q}$

First consider the case where $A$ does not need to be split using 2D hashing, that is, the estimated partition size $e_A \leq T$. Let $S = \sum_{i=1}^{n} X_i$ and $\mu = np$ be the expected value of $S$. Using Chernoff bound we have the following

$$P(S \leq (1-\delta)\mu) \leq \exp(\frac{-\mu\delta^2}{2}), \forall 1 > \delta > 0$$

Putting $\delta = 3/4$ and $\mu = np$, we get

$$P(S \leq \frac{1}{4}np) \leq \exp(\frac{-9np}{32})$$

$$P(n \geq \frac{4S}{p}) \leq \exp(\frac{-9np}{32})$$

Since no 2D hashing split is necessary, we know $T \geq e_A = \frac{S}{p}$. Thus $P(n \geq 4T) \leq \exp(\frac{-9np}{32}) \leq \exp(\frac{-9Tp}{8})$. Let $p = \frac{c}{T}$, we then have $P(n \geq 4T) \leq \exp(\frac{-9c}{8}) \leq \exp(-c))$.

The same inequality can be produced for the case where the estimated partition size $e_A > T$ such that 2D hashing is employed. We derive this result by repeating the previous argument using binary random variables $X_{ij}$ to denote that $R_i$ is both, sampled in $\mathbf{Q}$, and hashed to a sub-partition in the 2D matrix cell $j$.

Overall, the probability that the real partition size is a constant factor larger than threshold $T$ is exponentially small as the sampling rate grows. □

To make this guarantee more concrete, consider the following example. Let our partition size $T$ be fixed at 1000. Suppose we set $c = 10$, which makes the sampling probability $p = \frac{10}{1000} = 1\%$. At this modest sampling rate, we can guarantee that the real size of any anchor partition using the full data set is no greater than $4T = 4000$ with probability $1 - e^{-10} > 0.99\%$.

One might wonder why 2D hashing is not applied on the entire data set for partitioning and load balancing. We note that this strategy would be very similar to the Theta-Join [21], which uses 2D hashing to handle general joins on MapReduce. There are two main reasons why using 2D hashing on the entire data set for similarity join is not efficient. First, it does not prune out any record pairs and performs a lot of unnecessary computation in the form of pairwise comparisons. We on the other hand take advantage of the characteristics of similarity join to perform both partitioning and candidate pruning simultaneously. Second, 2D hashing is known to be inefficient in handling large data sets due to the communication cost of broadcasting to $\sqrt{m}$ nodes, where $m$ is the total number of machines. In comparison, we only use 2D hashing on a small set of records that have already been mapped to a common anchor partition. Here, the cost of broadcasting is insignificant, while the benefit of balancing load is quite significant.

We also note that although splitting skewed partitions has been used for similarity join [23, 25], previous approaches split partitions in ad-hoc manners that cannot provide an upper bound for the size of each partition. In fact, when more than $T$ records are mapped to the same partition, they sometimes represent a dense cluster of records in a small region. Previous approaches may not be able to reduce the size of such partitions, because all records are so close, that they have to be hashed to all sub-partitions in order to ensure correctness.

**Partition merging**. We also observe that for high dimensional data where records are sparsely distributed across the data space, it is possible that there exist many small partitions that have only a handful of records mapped to them. As a result we also considered the possibility of merging anchor partitions at the sampling stage using the partition size estimates. The benefit here is that outer partition records now only need to be mapped to one merged partition, instead of many smaller ones, which saves on communication and job startup costs. However, the problem of partition merging is related to Set Union Bin Packing, which is NP hard [19]. We implement a greedy approach of merging partitions with significant overlap as we scan anchor partitions. We observe that the empirical performance gain from merging is not significant, and the overhead of merging sometimes outweighs the cost savings. Accordingly, we will not discuss partition merging in the rest of this paper.

## 7.2 The mapping phase

In this step, the set of anchors and their respective 2D hashing factors from the previous step are available at all machines. Mappers read data records in parallel and decide for each record its home anchor partition (by comparing the distance to all anchors), as well as the outer partitions to which it belongs (using the filer rules discussed in Section 6). If a record is mapped to an anchor partition that requires 2D hashing (because the estimated size of the partition is larger than $T$), sub-partitions-ids will be produced in place of the anchor partition id using the 2D hashing scheme discussed in Section 7.1.

Note that a pair of records belonging to different home partitions may be members of both corresponding outer partitions. That is, consider a pair of records $R_1, R_2$ such that $R_1 \in H(A_1), R_2 \in H(A_2)$, and $R_1 \in O(A_1), O(A_2), R_2 \in O(A_1), O(A_2)$. Pairs like this, where $R_1 \in H(A_1) \cap O(A_2)$ and $R_2 \in H(A_2) \cap O(A_1)$ introduce unnecessary communication and computation costs since both $R_1$ and $R_2$ will be sent to partitions $A_1$ and $A_2$ respectively. To remove this redundancy, we map either all records in $H(A_1) \cap O(A_2)$ to partition $A_2$, or all records in $H(A_2) \cap O(A_1)$ to partition $A_1$, but not both. We use an approach similar to that in [25] to decide the direction of this mapping, and send $H(A_1) \cap O(A_2)$ to partition $A_2$ if $A_1$.id $< A_2$.id and $h(A_1.id + A_2.id)\%2 = 0$ for some chosen hash function $h(\cdot)$.

A home partition flag that indicates whether the record is in the target anchor's home partition is also sent along with the record to the verification phase. This is used so that we can avoid comparing two records both of which belong to an anchor's outer partition but neither of which belongs to the home partition of the same anchor.

## 7.3 The verification phase

Each machine will work on a separate, possibly hash-split anchor partition, to perform the verification record pairs. All records will be pre-sorted using the home partition / outer partition flags in a shuffle stage, to make sure that home partition records will be read first into the reducers. Each home partition record will be compared with existing home partition records already read, and added to the list of home partition records in memory. Outer partition records will be read after all home partition records are read. They will be compared with all home partition records and discarded. The union of the output of all reducers is the result of the similarity join.

In parallel to our work, authors in [25] also developed an algorithm with similar mapping/verification phases. However, our sampling and candidate filtering techniques are significantly different, and are experimentally shown in the next section to be more efficient, especially for high dimensional data with low distance thresholds.

# 8. EXPERIMENTAL RESULTS

We present an experimental evaluation of the proposed algorithm. The goals of our experimental study are:

• To evaluate the effectiveness of the filters designed in this paper and compare against previous work for metric space.

• To compare the scalability of different algorithms discussed in this paper using end-to-end execution time.

• To evaluate the sensitivity of the proposed ClusterJoin algorithm to different parameter settings.

## 8.1 Experimental Setup

### *8.1.1 Data set*

The first data set used in our experiments is from LinkedGeoData [1], which curates geo-spatial data used in OpenStreetMap [2]. We use the "Place" data set, which contains location information of 2.5 million of Points-Of-Interest and is processed into two dimensional coordinates. Since this is already one of the largest real world spatial data sets that we can find, in order to test algorithms using even larger data sets, we synthetically generate a larger data set based on this. Specifically, for each data record we add 9 synthetic records by perturbing both coordinates of the original record using Gaussian distributions ($\mu = 0, \sigma = 10$ miles), which produces a total of 25 million records. We perform similarity join using Euclidean distance on this data set, which can be useful for finding, for instance, all pairs of POIs that are within 1 miles of each other.

Our second data set consists of 430K news articles extracted from a recent index snapshot of Microsoft Bing search engine in the English domain. The average document size is 7KB. We model each news document using the Vector Space Model, and compute similarity joins for different distance functions. This can be used in a variety of applications such as identifying near duplicate news articles, or clustering related stories. Note that the News data set has a much higher dimensionality (each distinct word is seen as a dimension) than the 2 dimensional spatial data set.

We experiment with four metric distance functions: we use Euclidean distance on the first spatial data set, and Total Variation distance (TVD), Angular distance (Cosine similarity), and Jensen Shannon distance (JSD) [15] on the second document data set.

We have defined and developed specific filters for the first two distance functions in Section 6. We use our generic filter for Angular distance, and also Jensen Shannon distance (JSD), which is a statistical metric distance defined as follows [15].

DEFINITION 8.1. *Let $P$, $Q$ be two probability distributions,*

$$JSD(P|Q) = \frac{1}{2}KLD(P|M) + \frac{1}{2}KLD(Q|M)$$

*where $KLD(X|Y) = \sum_i X(i) \ln\left(\frac{X(i)}{Y(i)}\right)$ is the KL divergence, and $M = \frac{1}{2}(P+Q)$.*

### *8.1.2 Compared Methods*

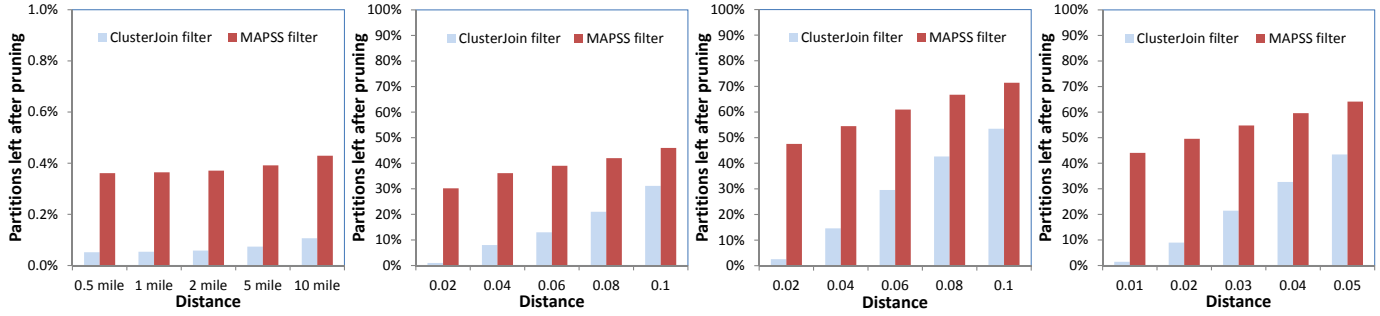
In order to evaluate the performance of different algorithms, we compare the end-to-end runtime of the following methods.

• **MAPSS [25].** This recently published approach handles joins with arbitrary metric distances, and is most similar to our method. We compare both filter pruning effectiveness across different distance functions, and end-to-end runtime with this approach.

• **V-Smart-Join [18].** V-Smart-Join first maps the record-id of each record to all tokens in the record. Each token will then be handled by a separate reducer that emits all pairs of records that share the same token. The score contribution for the same pair of records are then aggregated across tokens to obtain the similarity value. This approach can also handle a large class of similarity functions, and is shown to work well for data sets with a large alphabet and sparse records.

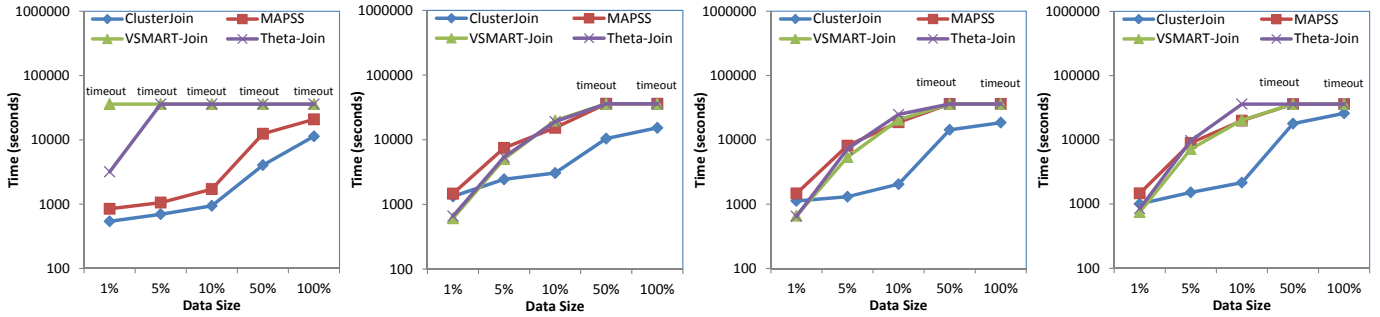
• **Theta-Join [21].** The Theta-Join approach splits work across reducers using two-dimensional hash partitioning (outlined in Figure 4 and Section 7.1). The two-dimensional hashing ensures that each pair of records meets at least once, and at the same time avoids broadcasting the full data set to all machines. This approach can handle arbitrary complex join conditions, including similarity join for metric distances, and can balance load well.

• **Prefix-Join [24].** We also implement the Prefix-Join in [24], which uses prefix-filter and PP-Join. Note that this approach is designed for set similarity joins, including Jaccard similarity and the set-based Cosine similarity. Since their set-based Cosine similarity and the unmodified vector-based Cosine similarity used in our experiments have different semantics (the scores they compute




(a) Euclidean (b) Angular (Cosine) (c) Total Variation (d) Jensen Shannon

Figure 5: Filter effectiveness comparison, using LinkedGeoData in (a) and News Data in (b), (c), (d)




(a) Euclidean (b) Angular (Cosine) (c) Total Variation (d) Jensen Shannon

Figure 6: Time comparison: vary data size, using LinkedGeoData in (a) and News Data in (b), (c), (d)

are also different), we do not compare with Prefix-Join in our main results. We did try set-based Jaccard similarity and discuss our findings in Section 8.2.4.

• **ClusterJoin.** This is our method discussed in Section 7.

There is also a very recent work MASS-Join [13] that handles set-based and string-based similarities, which are not the focus of this work.

We implement all algorithms described above and conduct experiments in a production MapReduce system [10]. All algorithms were executed concurrently with other production jobs, at the normal cluster workload, using a fixed amount of virtual resources.

## 8.2 Experimental evaluation

### *8.2.1 Filter effectiveness*

Since like our ClusterJoin method, the MAPSS [25] approach also uses the idea of filtering candidate pairs that are impossible to meet the distance threshold, before we evaluate end-to-end runtime it is interesting to directly compare the effectiveness of the filters proposed in our work and the one generic filter studied in [25]. Filter effectiveness apparently has a direct impact on the runtime, because the more effective filters are, the less time is needed for verification, which typically is the most expensive part of the job.

Figure 5 compares the MAPSS filter and the set of filters proposed in this work across four distance functions. Here the x-axis is the distance threshold, and y-axis is the average percentage of partitions that a record needs to be sent to after applying filters, where lower numbers are better (recall that without filters, each record needs to be sent to all partitions to guarantee correctness).

Figure 5a shows the effect of filtering for Euclidean distance on the 2-dimensional spatial data. Notice that since this is a low dimensional data set, filtering candidate pairs that cannot join is relatively easy. In fact both MAPSS filter and ClusterJoin filter can prune away a vast majority of candidate pairs for this low dimensional data. Still, we observe that with reasonably small distance thresholds, the ClusterJoin filter produces up to an order of magnitude fewer number of pairs for verification than the MAPSS filter.

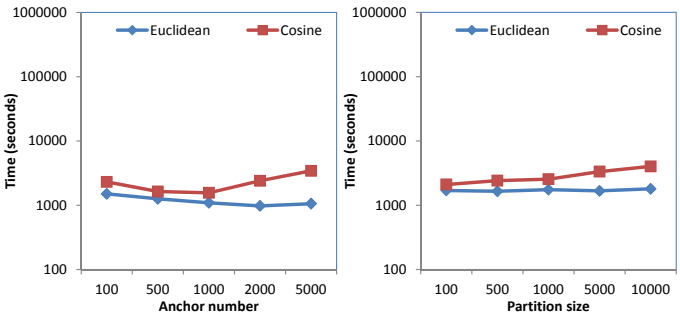
Figures 5b, 5c and 5d respectively plot the filtering effectiveness for Angular distance (Cosine), Total Variation distance (TVD) and Jensen Shannon distance (JSD), using the high dimensional document data set. Recall that we have developed and use distance-specific filters for TVD, while for Cosine and JSD we use our general filter. Notice that for all three cases our filter outperforms the MAPSS filter. However the relative difference between these two methods decreases as the distance threshold increases.

We observe that in all these three cases the filters are generally less effective when compared to the 2-dimensional Euclidean filter in Figure 5a, and produce good pruning only with low distance (high similarity) thresholds. This is partly attributable to the curse of dimensionality [7] – high dimensional data may just be inherently hard to prune away. However, we argue that this is still very useful, because in many real applications people are more interested in finding pairs of records with high similarity (low distance threshold), or alternatively those pairs that are most similar to each other. Applications using a low similarity (high distance) threshold, such as finding all POI pairs within 100 miles, or document pairs with only insignificant tokens overlapping, are possibly less natural, because the large number of matching pairs makes subsequent human consumption difficult.

### *8.2.2 Scalability test*

In this section, we vary the size of the data set and evaluate the end-to-end execution time to understand the scalability of each algorithm, which is one of the most important aspects of similarity join algorithms.

In Figure 6 we vary the size of the data set by sampling $1\%, 5\%,$




(a) Vary number of anchors (b) Vary partition size

Figure 7: Sensitivity Analysis

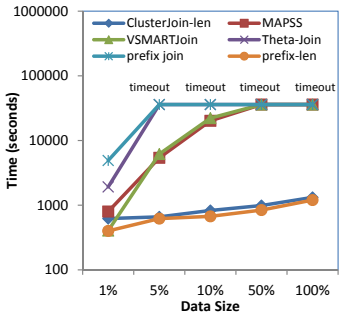



Figure 8: Jaccard similarity

10%, and 50% of the original data set, for Euclidean distance, Angular (Cosine) distance, TVD and JSD, respectively. We report runtime in seconds on a log-scale. Furthermore we limit the runtime of each job at 10 hours, after which we simply abort the job. We conservatively report the corresponding job time as 36000 seconds and place a *timeout* label next to the data point.

Figure 6a shows that for Euclidean distance, although both ClusterJoin and MAPSS are efficient and scalable as data size grows, ClusterJoin is 30% to 50% more efficient than MAPSS. The runtime gap is not as significant as the pruning effectiveness suggested in Figure 5a. This is because the 2-dimensional spatial data is relatively easy, and both approaches are already effective in pruning candidates such that verification does not dominate the job runtime. Also, the verification for pairs not pruned away is relatively cheap, since each record only has two coordinates, making distance computations inexpensive.

Also in Figure 6a, we note that V-SMART-Join is the least scalable approach for Euclidean distance, as it times-out for even the smallest data set. This is not surprising, as V-SMART-Join maps each record to constituent tokens in order to aggregate score contribution by tokens, and relies on sparse token occurrence in a large alphabet to be efficient. While this approach works well for the IP/cookie data set experimented in [18] which has a sparse alphabet (cookies), in our particular data set with only two tokens ($x$ and $y$ coordinates), this approach is expected to be the least efficient. Theta-Join also times-out with all but the smallest data set.

Figures 6b, 6c, 6d show scalability for Angular, TVD, and JSD, respectively. With the exception of the smallest, 1% data, ClusterJoin is consistently the most efficient approach. And as the data grows to 50% all algorithms except ClusterJoin time out at the 10-hour mark. This underscores the efficiency of our proposed approach that combines powerful filters and dynamic load balancing.

### 8.2.3 *Sensitivity analysis*

Our ClusterJoin algorithm has two parameters: the number of anchors, and the expected number of tuples beyond which a partition needs to be split. In this section we analyze the impact of these two parameters on performance and show our results in Figure 7. We pick two distance functions on the two different data sets: Euclidean on the spatial data and Cosine on the document data. In both cases we use 10% of the original data sets for efficiency considerations. Results for TVD/JSD are similar to Cosine and are omitted.

Figure 7a shows the performance of ClusterJoin for Euclidean and Cosine distance functions using different numbers of anchors. Execution time for Euclidean distance is relatively insensitive, and goes down slightly as we increase the number of anchors. This is because for low dimension spatial data, having a larger number of anchors will likely make the "closest" anchor even closer, thus improving the effectiveness of our filtering procedure. We observe the same trend for Cosine distance up to 1000 anchors, beyond which the total runtime increases as more anchors are added. There may be two reasons for this increase. First, in high dimensional space, adding more anchors does not significantly reduce the average distance to the "closest" anchor. As a result the benefit in pruning is not as pronounced. Second, as more anchors are introduced, the amount of time spent on partition size estimation also increases. The effect is more apparent for Cosine perhaps because for the document data set pairwise distance computation is expensive. Overall, picking $0.1\% - 1\%$ of the original data set seems to be a good empirical setting for anchors.

Figure 7b illustrates the runtime with different split conditions that impact partition size. For Euclidean distance the runtime is again insensitive to partition size, because pairwise distance computation is inexpensive. For Cosine distance the runtime grows with more than 1000 anchors. This is as expected because of the expensive computation for documents similarity, which perhaps magnifies the straggler effect. While this parameter depends on factors such as the available computation resource on worker machines, a partition size of 1000 seems to be a good empirical number to use.

### 8.2.4 *Discussions: other distances and data sets*

Although the focus of this work is to support general metric distance including distribution- and vector-based distances, we notice that some set-based similarity functions have direct metric distance counterparts. For instance Jaccard similarity has a metric distance equivalent. Given that Jaccard similarity is heavily studied in the literature [5, 6, 24, 27], and we don't have a special filter for Jaccard yet, it would be interesting to see how our approach performs using the general filter. Our results using the news data set and Jaccard similarity of 0.9 are reported in Figure 8.

In addition to the techniques compared previously, we also compare with PrefixJoin [24] which is one of the pioneering works on similarity joins in MapReduce and is shown to be scalable for set-similarity metrics like Jaccard similarity. To our surprise, PrefixJoin turns out to be less scalable than expected. We believe this is because of the characteristics of the data used in our experiments. Each record in our news document data is considerably longer (7KB in size) than the data previously used in [24], including DBLP records and paper titles/abstracts. This news data set is unfavorable to a prefix-based approach for two reasons. First, since each record is much longer, the size of prefixes grows considerably, requiring the same record to be mapped to many more reducers. Second, unlike DBLP records, which have very distinct, unique tokens (like authors' last name) and when used as a prefix can be very selective, the news documents does not have as many unique tokens proportionate to the size of the record, thus reducing

the effectiveness of PrefixJoin and overwhelming reducers responsible for common tokens.

We notice that compared to the previously used data sets, the news data set is more heterogeneous with great variability in document sizes. This inspires us to develop a length-based filter used during the Mapper phase. In particular, we observe that documents of size $l$ cannot possible join with documents of size less than $sl$, or documents of size greater than $\frac{l}{s}$, where $s$ is the similarity threshold (or $1-$distance threshold) for Jaccard. Given this, we bucketize documents by length into consecutive ranges $\{b_1 : [0, k), b_2 : [k, \frac{k}{s}), b_3 : [\frac{k}{s}, \frac{k}{s^2}), \ldots\}$, where the observation is that documents in bucket $b_i$ can only join with documents in buckets $b_{i+1}$ and $b_{i-1}$, and nothing else. Based on this, we put consecutive buckets into groups, that is $g_i = \{b_i \cap b_{i+1}\}$. We use the group id, $g_i$, in conjunction with the anchor partition id in our Mapping Phase 7.2 as the reducer key, thereby further reducing the size of each partition. This approach still guarantees correctness, because in each anchor partition, a pair of documents in adjacent buckets $b_i$, $b_{i+1}$ will be mapped to exactly one sub-partition in group $g_i$. This new partitioning scheme greatly improves scalability for both ClusterJoin and PrefixJoin, as shown by ClusterJoin-len and prefix-len curves in Figure 8.

While the idea of a length-based filter is not new and has been used, for instance in PP-Join [27], to the best of our knowledge the length bucketization scheme has not been used in mapping partitions for similarity join in Map Reduce.

**Discussion.** The point of this exercise is not to show that our approach is the best for Jaccard similarity. In fact, we believe that for short homogeneous data sets with distinct tokens, PrefixJoin [27] and a very recent work MassJoin [13] (shown to be more scalable than PrefixJoin) may be more suitable.

The upshot of this is that while ClusterJoin is effective for a large number of scenarios, it is not always the best approach. In fact, we believe that it is unlikely that one technique can dominate every other algorithm across all possible settings, especially given the complexity of the problem under study. As a result, different techniques like ClusterJoin need to be developed that are optimized for different distance functions and data sets with different characteristics. Our ClusterJoin approach, for example, is perhaps more suitable for vector- or distribution-based distance functions with low distance thresholds and high dimensional data. As another example, even though the V-SMART-Join approach is not very efficient in our experiments using high dimensional document data sets, is likely to be the most scalable for sparse data sets with a large token alphabet, as the authors rightfully argue and show in [18] using the IP/Cookie data set.

Perhaps just as Hash Join, Index Join and Nested Loop Join work well under different circumstances for relational join, understanding the relative performance of different similarity-join algorithms in different scenarios will allow us to develop a cost model that ultimately can be used to select the right algorithm based on the data and distance function in question. We believe this is an interesting area for future research.

# 9. CONCLUSIONS

In this paper, we propose a ClusterJoin framework for similarity joins using MapReduce. We design a set of strong filters using a novel bisector-based reasoning, and a dynamic partitioning scheme that guarantees load balancing with high probability. Our approach is scalable and our experiments show that it outperforms the current state-of-the-art techniques significantly for a variety of distance functions with low distance thresholds.

# 10. REFERENCES


[1] Linkedgeodata: http://linkedgeodata.org.

[2] Openstreetmap: www.openstreetmap.org.

[3] F. Afrati, A. Das Sarma, A. Rajaraman, P. Rule, S. Salihoglu, and J. Ullman. Anchor points algorithms for hamming and edit distance. In *Proceedings of ICDT*, 2014.

[4] F. N. Afrati, A. D. Sarma, D. Menestrina, A. G. Parameswaran, and J. D. Ullman. Fuzzy joins using mapreduce. In *Proceedings of ICDE*, 2012.

[5] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *Proceedings of VLDB*, 2006.

[6] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proceedings of WWW*, 2007.

[7] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proceedings of ICDT*, 1999.

[8] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *Computer Networks*, 1997.

[9] V. Bryant. *Metric Spaces: Iteration and Application*. Cambridge University Press, 1985.

[10] R. Chaiken, B. Jenkins, P. Larson, and B. Ramsey. Scope: Easy and efficient parallel processing of massive data sets. In *Proceedings of VLDB*, 2008.

[11] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of ICDE*, 2006.

[12] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Communication of ACM*, 2010.

[13] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng. Massjoin: A mapreduce-based algorithm for string similarity joins. In *Proceedings of ICDE*, 2013.

[14] A. Doan, A. Halevy, and Z. Ives. Principles of data integration. In *Morgan Kaufmann*, 2012.

[15] D. M. Endres and J. E. Schindelin. A new metric for probability distributions. In *IEEE Trans. Inf. Theory*, 2003.

[16] M. R. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of SIGIR*, 2006.

[17] T. C. Hoad and J. Zobel. Methods for identifying versioned and plagiarized documents. In *JASIST 54(3)*, 2003.

[18] A. Metwally and C. Faloutsos. V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. In *Proceedings of VLDB*, 2012.

[19] D. A. Nehme-Haily. The set-union knapsack problem. *University of Texas at Austin, Thesis*, 1995.

[20] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. Spatial tessellations: Concepts and applications of voronoi diagrams. 2009.

[21] A. Okcan and M. Riedewald. Processing theta-joins using mapreduce. In *Proceedings of SIGMOD*, 2011.

[22] V. Satuluri and S. Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. In *Proceedings of VLDB*, 2012.

[23] Y. N. Silva and J. M. Reed. Exploiting mapreduce-based similarity joins. In *Proceedings of SIGMOD*, 2012.

[24] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of SIGMOD*, 2010.

[25] Y. Wang, A. Metwally, and S. Parthasarathy. Scalable all-pairs similarity search in metric spaces. In *Proceedings of KDD*, 2013.

[26] W. Winkler. The state of record linkage and current research problems. In *Technical Report, US Bureau of Census*, 1999.

[27] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *Proceedings of WWW*, 2008.