Soft Comput
DOI 10.1007/s00500-017-2561-3

METHODOLOGIES AND APPLICATION

Implementation of scalable fuzzy relational operations in MapReduce

Elham S. Khorasani[1] · **Matthew Cremeens**[1] · **Zhenge Zhao**[1]

Abstract One of the main restrictions of relational database models is their lack of support for flexible, imprecise and vague information in data representation and querying. The imprecision is pervasive in human language; hence, modeling imprecision is crucial for any system that stores and processes linguistic data. Fuzzy set theory provides an effective solution to model the imprecision inherent in the meaning of words and propositions drawn from natural language (Zadeh, Inf Control 8(3):338–353, doi:10.1016/S0019-9958(65)90241-X, 1965; IGI Global, https://books.google.com/books?id=nt-WBQAAQBAJ, 2013). Several works in the last 20 years have used fuzzy set theory to extend relational database models to permit representation and retrieval of imprecise data. However, to our knowledge, such approaches have not been designed to scale-up to very large datasets. In this paper, the MapReduce framework is used to implement flexible fuzzy queries on a large-scale dataset. We develop MapReduce algorithms to enhance the standard relational operations with fuzzy conditional predicates expressed in natural language.

Communicated by V. Loia.

Matthew Cremeens and Zhenge Zhao are contributed equally to this article.

✉ Elham S. Khorasani
esahe2@uis.edu

Matthew Cremeens
mcrem2@uis.edu

Zhenge Zhao
mcrem2@uis.edu

[1] Department of Computer Science, University of Illinois at Springfield, One University Plaza, Springfield, IL, USA

Keywords Relational operations · Fuzzy set theory · MapReduce · Fuzzy queries

1 Introduction

The mainstream relational database management systems use a Boolean logic to characterize users' queries. This means that the query condition is either satisfied or not satisfied. The use of Boolean logic poses a restriction in terms of flexibility and semantics of relational operations and does not allow expressing preference or ranking of query results. In many real-world situations, the queries consist of imprecise words and conditions and the objective is not merely to find the tuples which satisfy a given query but to determine to what extent each tuple satisfies the conditions in the query and to allow ranking of such tuples. For instance, it seems quite natural for an online real-estate company to answer questions such as:

Give me all two-bedroom apartments which are not too expensive and are close to downtown.

While such queries might be quite simple for a human real-estate agent to respond, they are too imprecise for traditional database systems to process and respond. The imprecision arises from using linguistic words such as "too expensive" and "close." Using boolean logic demands translation of such words into precise numeric values or intervals which could potentially result in loss of information. For example, suppose that we rewrite the above query in a form that can be processed by traditional databases:

"Find all two-bedroom apartments that cost <=$320K and are located <=10 miles from downtown."

This query will return an apartment which costs $320K and is located 10 miles from downtown but fails to return a $321K apartment that is located 10.5 miles from down-

Published online: 23 March 2017

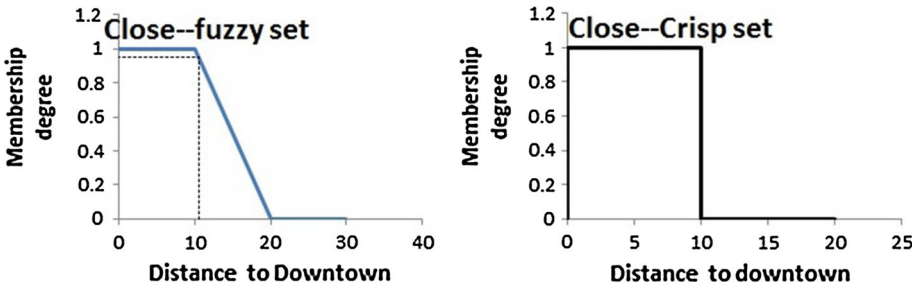

Fig. 1 Fuzzy set versus crisp set for representing the meaning of "close"

town. While the difference between the two apartments is negligible, the boolean logic imposes a rigid boundary in the meaning of the words "expensive" and "close" which results in returning only the first apartment discarding the second.

Fuzzy set theory and possibility theory (Zadeh 1965, 1999) provide an effective solution to represent and process imprecise linguistic information. As opposed to classical set theory, where an element either belongs or does not belong to a set, in fuzzy set theory an element can partially belong to a set. The partial membership of elements in a set is characterized by a membership function which takes values in the real unit interval [0,1]. Figure 1 illustrates the difference between a fuzzy set translation of the word "close to downtown" as opposed to its crisp (interval) translation. In contrast to the crisp set , the membership degree of the fuzzy set gradually decreases after 10 miles. Distance 10.5 miles to downtown is still considered close to degree 0.95.

Several works have been proposed in the last 20 years to extend relational database models to permit representation and retrieval of imprecise data using fuzzy set theory. The existing approaches to fuzzy relational database models can be divided into three main categories: 1—similarity-based models, where the ordinary equivalence relation between the domain values is replaced by similarity or proximity relations (Shenoi and Melton 1989, 1990; Buckles and Petry 1982); 2—possibility-based models (Prade and Testemale 1984; Bosc and Prade 1997; Medina et al. 1995; Galindo 2005), where an entire tuple is associated with a membership degree and an attribute value is allowed to be a fuzzy set on the attribute domain; and 3—extended possibility-based models where the fuzziness of data appears both in attribute values in the form of a fuzzy set as well as in the attribute domain in the form of a proximity relation (Ma et al. 2000; Ma and Mili 2002). For a comprehensive survey on fuzzy relational database systems, one can refer to Chen (1998), Petry (1997), Ma and Yan (2010).

The focus of this paper is to develop MapReduce algorithms to scale-up the fuzzy relational operations to large-scale crisp datasets. We formulate selection, projection, union, difference, intersection and join operations with fuzzy conditions.

Several recent papers have extended the classical MapReduce equi-join operation by replacing the rigid equality condition with a softer similarity relation (Afrati et al. 2012; Vernica et al. 2010; Metwally and Faloutsos 2012; Das Sarma et al. 2014; Wang et al. 2013; however, to our knowledge, this is the first work that utilizes fuzzy set theory to develop scalable MapReduce algorithms to perform imprecise linguistic queries on a large-scale database.

The rest of the article is organized as follows: the next section presents an overview of fuzzy relational algebra which is used to model imprecise queries on crisp datasets. Section 3 formulates MapReduce algorithms to scale-up fuzzy relational operations and analyzes their time efficiencies. Section 4 provides a discussion on fuzzy join optimization and load balancing. The algorithms are implemented and tested against a real-world dataset, and their scalability is discussed in Sect. 5. Section 6 concludes the paper and presents future directions of this research.

2 Imprecise queries on crisp datasets

Imprecise queries addressed to a crisp dataset are modeled by fuzzy relational operations. A fuzzy relational operation takes a set of crisp relations as input and produces a fuzzy relation as a result, where each tuple is associated with a degree to which the fuzzy operation is satisfied.

A fuzzy relation R is characterized by its membership function $\mu_R(t) : D \rightarrow [0, 1]$, where t is a tuple in R and D is its domain. The basic algebra on fuzzy relations is as follows (Petry 1997):

Cartesian Product The membership degree of a tuple xy in the Cartesian product $R \times S$ is the minimum of the membership degrees of tuples x in R and y in S:

$$\mu_{R\times S}(xy) = \min(\mu_R(x), \mu_S(y))$$

where $\mu_R(x)$ and $\mu_S(y)$ are degrees of membership of x in R and y in S, respectively.

Union The membership degree of a tuple x in $R \cup S$ is the maximum of its membership degrees in R and S:

$$\mu_{R\cup S}(x) = \max(\mu_R(x), \mu_S(x))$$

Intersection The membership degree of a tuple x in $R \cap S$ is the minimum of its membership degrees in R and S:

$$\mu_{R\cap S}(x) = \min(\mu_R(x), \mu_S(x))$$

Difference The membership degree of a tuple x in $R–S$ is the minimum of its membership in R and S complement:[1].

$$\mu_{R-S}(x) = \min(\mu_R(x), \mu_{\overline{S}}(x)) = \min(\mu_R(x), 1 - \mu_S(x))$$

Selection The membership degree of a tuple x in $\sigma_\phi(R)$ (where σ is the selection operation and ϕ is a fuzzy condition) is equal to the minimum of its membership degree in R and the degree to which it satisfies the fuzzy condition ϕ:

$$\mu_{\sigma_\phi(R)}(x) = \min(\mu_R(x), \mu_\phi(x))$$

Projection The membership degree of a tuple u in $\pi_\gamma(R)(u)$ (where π is the projection operation and γ is a proper subset of attributes in R) is equal to the maximum membership of all tuples in R whose γ attributes have the same values as the γ attributes in u:

$$\mu_{\pi_\gamma(R)}(u) = \sup_{x:x.\gamma=u.\gamma} (\mu_R(x))$$

Join The membership degree of a tuple xy in $R \bowtie_{A\theta B} S$ (where $\bowtie$ is the join operation and θ is a fuzzy comparator on the join attributes $R.A$ and $S.B$) is equal to the minimum of the membership degree of x in R, y in S and the degree to which the join keys satisfy the fuzzy comparator operator:

$$\mu_{R\bowtie_{A\theta B}S}(xy) = \min(\mu_R(x), \mu_S(y), \mu_\theta(x.A, y.B))$$

Note that in a crisp dataset all tuples in a relation have a membership degree equal to one (i.e., $\mu_R(x) = 1$ for all $x \in R$). However, once fuzzy operations are applied fuzzy relations are produced as intermediate or final results in which the tuples may have a membership degree less than or equal to one.

Several fuzzy conditions in a fuzzy relational operation may be combined using logical AND ($\wedge$) and OR($\vee$) operators. Suppose ϕ_1 and ϕ_2 are two fuzzy conditions in a relational operation. We have:

- $\mu_{\phi_1\wedge\phi_2}(x) = \min(\mu_{\phi_1}(x), \mu_{\phi_2}(x)))$
- $\mu_{\phi_1\vee\phi_2}(x) = \max(\mu_{\phi_1}(x), \mu_{\phi_2}(x)))$

One can generalize the AND and OR operators by assigning a weight (w_i) to each fuzzy condition (ϕ_i) to express its importance in a query (Dubois and Prade 1986):

- $\mu_{\bigwedge_i w_i\phi_i}(x) = \min_i(\max(\mu_{\phi_i}(x), 1 - w_i))$
- $\mu_{\bigvee_i w_i\phi_i}(x) = \max_i(\min(\mu_{\phi_i}(x), w_i))$

One can also modify a fuzzy condition by applying linguistic modifiers (or hedges), such as "very," "somewhat," "extremely," "more or less." A linguistic modifier is modeled a function $m : [0, 1] \rightarrow [0, 1]$ that is applied to a fuzzy set and modifies the membership degrees. The most common types of linguistic modifiers are *concentrator* and *dilator*. Concentrators, such as: "very" and "extremely" intensify the membership function of a fuzzy set while dilators such as "more or less" and "rather" dilute it. Concentrators and dilators are typically modeled as the following function (A is a fuzzy set):

$$\mu_{mA}(x) = (\mu_A(x))^n$$

For example, $\mu_{\text{very tall}}(h) = (\mu_{\text{tall}}(h))^2$ and $\mu_{\text{some what tall}}(h) = (\mu_{\text{tall}}(h))^{1/2}$

To demonstrate fuzzy relational operations, let us assume the Faculty relation in Table 1.

Now suppose we want to find all young faculty who make a somewhat good salary. This query can be formulated as follows:

$$\mu_{\sigma_{\text{age=young AND salary=some what good}}(\text{Faculty})}(x) = \min(\mu_{\text{Faculty}}(x), (\mu_{\text{good}}(\text{salary}(x)))^{1/2}, \mu_{\text{young}}(\text{age}(x))) \quad (1)$$

where $\mu_{\text{Faculty}}(x)$ is the membership of x in the Faculty relation and it is equal to 1 for all tuples x because Faculty is a crisp relation.

Suppose that the fuzzy sets representing the imprecise words "young" and "good" are given as shown in Fig. 2.

[1] If A is a fuzzy set and $\overline{A}$ is its complement then $\mu_{\overline{A}}(x) = 1 - \mu_A(x)$.

Table 1 Faculty relation

ID	Name	Age	Salary (K)
1234234	Jones	33	70
4324364	Champaign	33	120
4354354	Jameson	41	45
8454857	Nash	50	60
8382347	Jung	29	50
8933897	Li	38	90
4354958	Zhu	40	55
8454875	Edwards	33	65
4854858	Mitchell	38	57
4985948	Kerri	48	100
8435487	Cornell	50	45

The trapezoidal fuzzy sets in this figure can be formulated as follows:

$$\mu_{\text{good}(x)} = \begin{cases} 0 & x \leq 40 \\ (x-40)/30 & 45 < x \leq 70 \\ 1 & 70 < x < 100 \\ (200-x)/100 & 100 \leq x < 200 \\ 0 & x \geq 200 \end{cases}$$

$$\mu_{\text{young}(x)} = \begin{cases} 1 & x \leq 35 \\ (55-x)/20 & 35 < x \leq 55 \\ 0 & x > 55 \end{cases}$$

Given the above fuzzy set representations, Eq. 1 can be used to compute the degree to which each tuple in Table 1 satisfies the fuzzy conditions age = young and salary = somewhat good. For example, the degree to which Nash is considered young and his salary is somewhat good is equal to 0.25:

$$\begin{aligned} &\mu_{\sigma_{\text{age = young AND salary = somewhat good}}}(\text{Faculty})(\text{Nash}) \\ &= \min(1, (\mu_{\text{good}}(60))^{1/2}, \mu_{\text{young}}(50)) \\ &= \min(1, 0.81, 0.25) \\ &= 0.25 \end{aligned}$$

Similarly, we can compute the membership value to which each faculty in Table 1 is young and makes a somewhat good salary. The result is shown in Table 2.

One can specify a threshold value (i.e., cutoff membership value) and only return a set of tuples which satisfy the fuzzy conditions to a degree greater than the threshold.

Consequently, fuzzy set theory allows us to effectively model the meaning of imprecise words in a query and provides a calculus to compute and rank the results based on the degree to which they satisfy the fuzzy conditions in the query.

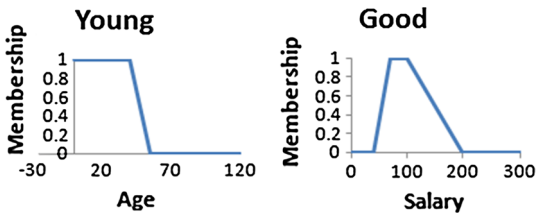

Fig. 2 Fuzzy sets representing the words "young" and "good." The domain of "age" is assumed to be [0, 120], and the domain of "salary" is assumed to be: [0, 300K]

Table 2 Faculty relation

ID	Name	Age	Salary (K)	μ_{young} & somewhat good salary
1234234	Jones	33	70	1
4324364	Champaign	33	120	0.64
4354354	Jameson	41	45	0.02
8454857	Nash	50	60	0.25
8382347	Jung	29	50	0.11
8933897	Li	38	90	0.72
4354958	Zhu	40	55	0.25
8454875	Edwards	33	65	0.7
4854858	Mitchell	38	57	0.32
4985948	Kerri	48	100	0.35
8435487	Cornell	50	45	0.02

3 Scaling flexible queries with MapReduce

3.1 MapReduce framework

MapReduce (Dean and Ghemawat 2008) is one of the most common platforms for processing big data. Many standard algorithms have been extended to comply with the shared-nothing architecture of MapReduce. MapReduce model of computation consists of two main functions: *Map* and *Reduce*. A large input file is broken into chunks and stored in a distributed file system. During the execution of a MapReduce job, the mapper tasks read an input split and call the Map function on each single record in the input split to produce a set of intermediate key-value pairs. The intermediate key-value pairs are hashed to one or more reducers based on their keys. The key-value pairs sent to each reducer are sorted and grouped by their keys. A reduce function is called for every key and all its associated values to produce a chunk of final output.

The cost of a MapReduce job is expressed in terms of $(M + C + R)$ where M is the map cost across all records, C is the communication cost of passing intermediate key-value pairs to the reducers and R the total computation cost of all reducers.

3.2 Fuzzy relational operations in MapReduce

This section describes MapReduce algorithms for fuzzy relational operations on a large crisp dataset. The selection, union, intersection, and difference operations do not require much modifications to the crisp counterparts. The join operation is more complex as it requires computing the fuzzy comparator operator for all pairs of records in R and S.

3.2.1 Fuzzy selection

The selection operator is a map-only job that reads a record r from relation R, computes the degree (μ) to which r satisfies a given fuzzy condition and emits r and μ as key and value, respectively. A dictionary file which maps the linguistic words in the query to their fuzzy set representations is stored in the memory of the nodes running the mappers.

Algorithm 1 Fuzzy selection map-only job

1: **Map:**
Input: r: an input record, $\mu_R(r)$: membership of r in R, ϕ:fuzzy condition
2: Parse ϕ and retrieve the fuzzy sets corresponding to the linguistic words in ϕ
3: compute $\mu_\phi(r)$
4: $d = min(\mu_R(r), \mu_\phi(r))$
5: emit key=r, value=d

The cost of the selection operation is the cost of the Map function across all records, $O(|R|)$, where $|R|$ is the number of records in R.

3.2.2 Fuzzy union, intersection and difference

The Map function for fuzzy union $R \cup S$ reads an input record r (from R or S) and emits r and its membership degree (in R or S). The reduce function computes the maximum membership degree for each input record it receives as a key.

Algorithm 2 MapReduce job for Fuzzy union $R \cup S$

1: **Map:**
Input: r: an input record, $\mu(r)$: membership of r
2: emit key=r, value = $\mu(r)$

1: **Reduce:**
Input: r: an input record, value= $list_i < \mu_i(r) >$
2: emit key=r, value = $max(list_i < \mu_i(r) >)$

The Map function for fuzzy intersection $R \cap S$ is the same as fuzzy union. The reduce function emits the minimum membership degree for each record only if both R and S have the record.

Algorithm 3 MapReduce job for Fuzzy intersection $R \cap S$

1: **Map:**
Input: r: an input record, $\mu(r)$: membership of r
2: emit key=r, value = $\mu(r)$

1: **Reduce:**
Input: r: an input record, value= $list_i < \mu_i(r) >$
2: **if** value has at least two membership degrees **then**
3: emit key=r, value = $min_i(list_i < \mu_i(r) >)$
4: **end if**

The Map function for fuzzy difference R–S reads an input record r and emits r as the key. For value, it emits the name of the relation (R or S) to which r belongs as well as its membership degree. The reduce function receives a record r as the key and a list of its associated relations and degrees of membership in each relation. If r belongs to R but not S, it will emit r and its membership degree in R. If r belongs to both R and S it will emit r and the minimum of its membership degrees in R and S complement.

Algorithm 4 MapReduce job for Fuzzy difference $R - S$

1: **Map:**
Input: r: an input record, N name of the relation to which r belongs (R or S), $\mu_N(r)$: membership of r in N
2: emit key=r, value = $(\mu_N(r), N)$

1: **Reduce:**
Input: r: an input record, value= $list_N < (\mu_N(r), N) >$
2: **if** value== $\{(\mu_R(r), R)\}$ **then**
3: emit key=r, value = $\mu_R(r)$
4: **end if**
5: **if** value== $\{(\mu_R(r), R), (\mu_S(r), S)\}$ **then**
6: emit key=r, value = $min(\mu_R(r), 1 - \mu_S(r))$
7: **end if**

In all the above three algorithms, the mappers read $|R| + |S|$ input records and emit each record in the output. Each reducer performs a linear operation on the values it receives. Hence, the total cost for fuzzy union, intersection and difference is:

$$M + C + R = O(|R| + |S|) + O(|R| + |S|) + O(|R| + |S|)$$

3.2.3 Fuzzy projection

The Map function for fuzzy projection $\pi_\gamma(R)$ reads each input record r in R and emits the gamma attributes of r, $r.\gamma$, as key and the membership degree of r in R as value. A reducer receives a key $r.\gamma$, produced by any of the map tasks, and a set of membership degrees associated with it. It emits $r.\gamma$ and the maximum of its membership degrees.

To illustrate the mapreduce algorithm for fuzzy projection suppose that we would like to return the age for young faculties who make a somewhat good salary from Table 1. The

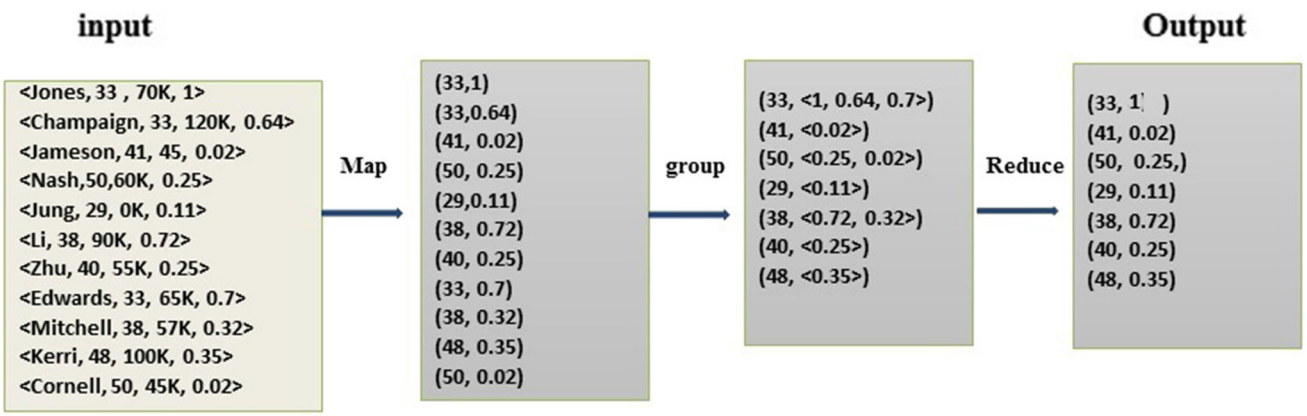

Fig. 3 The Map function returns the age and the membership value for each record. The reduce function returns the maximum membership value for each age

Algorithm 5 MapReduce job for Fuzzy projection $\pi_\gamma(r)$

1: **Map:**
Input: r: an input record, γ: projection attributes $\mu_R(r)$: membership of r in R
2: emit key=r.γ, value = $\mu_R(r)$

1: **Reduce:**
Input: u:projected attributes of an input record, value= $list_{i:r_i.\gamma=u} < (\mu_R(r_i) >$
2: emit key=u, value = $max_{i:r_i.\gamma=u}(\mu_R(r_i))$

selection operation is applied first to return the membership values in Table 2. Then the projection is applied to return the age and membership values from this table. The MapReduce data flow for projection is illustrated in Fig. 3

The cost of fuzzy projection is:

$$M + C + R = |R| + |\pi_\gamma(R)| + |\pi_\gamma(R)|$$

where $|\pi_\gamma(R)|$ is the size of relation R after eliminating the attributes not included in γ.

3.2.4 Fuzzy join

The fuzzy $R-S$ join operation takes a fuzzy comparator (such as fuzzy equal, fuzzy greater than and fuzzy less than) and computes the degree to which every pair ($r \in R, s \in S$) satisfies the join condition. The pairwise computation has a quadratic running time and does not fit well within the MapReduce framework. Hence, a heuristic partitioning and filtering is applied to avoid unnecessary computations. We demonstrate the fuzzy join algorithm via a small example.

Consider a dataset for online retail sales which includes two files that, among other attributes, have columns for customer name, age of customer, and product purchased (Table 3).

Table 3 Store X and store Y datasets

Name	Age	Product
Store X		
Person A	32	PA
Person B	35	PB
Person C	33	PC
Person D	38	Pd
Store Y		
Person E	36	PE
Person F	35	PF
Person G	32	PG
Person H	30	PH

Suppose we are interested to find all pairs of products from both stores that people of approximately the same age purchased. More formally, for every pair of records x in Store X and y in Store Y we would like to find the membership degree to which x. Age is fuzzy equal (FEQ) to y.Age:

$$\mu_{\text{StoreX}\bowtie_{(\text{x.Age FEQ y.Age})}\text{StoreY}}(xy) = \min(\mu_{\text{StoreX}}(x), \mu_{\text{StoreY}}(y), \mu_{\text{FEQ}}(x.\text{Age}, y.\text{Age})) \quad (2)$$

StoreX and StoreY are both crisp relations and so $\mu_{\text{StoreX}}(x)$ and $\mu_{\text{StoreY}}(y)$ are both equal to 1; hence,

$$\mu_{\text{StoreX}\bowtie_{(\text{x.Age FEQ y.Age})}\text{StoreY}}(xy) = \mu_{\text{FEQ}}(x.\text{Age}, y.\text{Age}) \quad (3)$$

Typically, we are interested to return only the pairs whose degree of membership in the join is above a certain threshold. For example, we would like to consider only the customers whose ages are fuzzy equal to at least 75% degree. To begin, we need a fuzzy membership function for the fuzzy comparator, fuzzy equals, to determine the degree to which two age values are approximately equal. Suppose we have a trapezoid

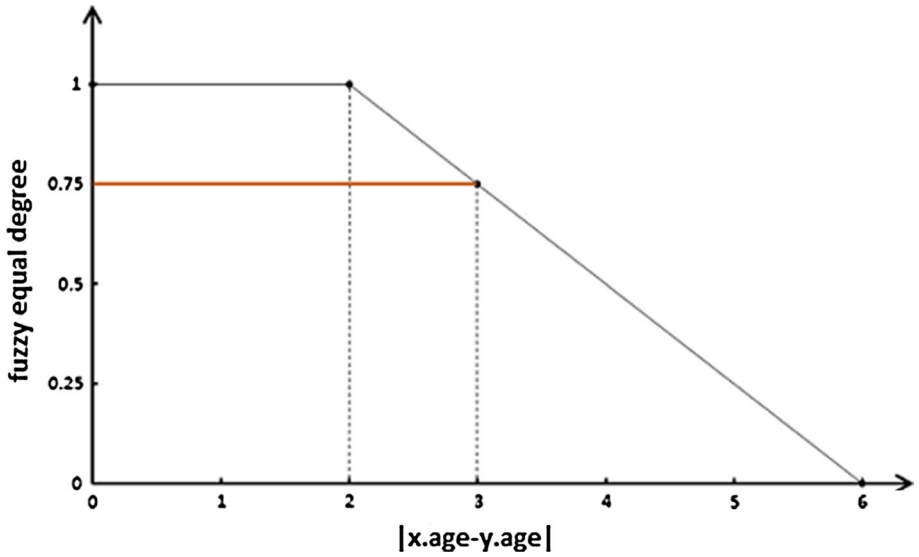

Fig. 4 membership function for the fuzzy comparator "approximately equal" for attribute age. The x-axis shows the absolute difference between two age values x.Age and y.Age, and the y-axis is the degree to which $|x.\text{Age} - y.\text{Age}|$ is considered fuzzy equal

membership function as displayed in Fig. 4. This membership function is defined on the absolute distance between two age values (i.e., $|x.\text{Age} - y.\text{Age}|$) and takes the following form:

$$\mu_{\text{FEQ}}((x.\text{Age}, y.\text{Age})) = \begin{cases} 1 & 0 <= |x.\text{Age} - y.\text{Age}| <= 2 \\ (6 - |x.\text{Age} - y.\text{Age}|)/4 & 2 < |x.\text{Age} - y.\text{Age}| <= 6 \\ 0 & 6 < |x.\text{Age} - y.\text{Age}| \end{cases}$$

Since we are only interested in getting all pairs of age values which are fuzzy equal to a degree greater than 0.75, we can take the α-cut (Buckley and Eslami 2002) of the membership function at point 0.75. The α-cut of a fuzzy set at point $\alpha \in [0, 1]$ is the set of domain elements whose membership degrees are greater than or equal to α (Klir et al. 1997). The α-cut of fuzzy equal as defined above is the interval [0, 3] as depicted in Fig. 4. This means that if $|x.\text{Age} - y.\text{Age}| <= 3$ then x.Age and y.Age are considered fuzzy equal to a degree of at least 0.75 or 75%.

The overall idea of scalable fuzzy equal join algorithm is to take the domain of the join attribute (in this case age) and partition it into equal-sized intervals with a length equal to the α-cut of the fuzzy comparator at the threshold value. The mappers send each input record to its home partition (i.e., the interval it falls into) as well as the partition to its immediate right.

For example, let us assume that the domain of the Age attribute in Table 3 is the interval [18,75]. The length of the α-cut of the fuzzy comparator "fuzzy equal" at threshold 0.75 is 3. Hence, the interval [18,75] is partitioned into 19 sub-intervals of size 3, that is: $p_0 = [18, 21]$, $p_1 = (21, 24]$, ..., $p_{17} = (69, 72]$, $p_{18} = (72, 75]$.

Each mapper receives an input record as well as the name of the relation that contains the record and sends the record to its home partition, p_h, and its immediate right partition, p_{h+1}. The mapper also emits a flag with each record that distinguishes between the home and its immediate right partition and signifies if the record falls within the first half or the second half of the range of its home partition.

For instance, a mapper receives the input record (Person A, 32, PA) from Store X in Table 3 and computes the index of its home partition as $p_h = \lfloor \frac{32-18}{3} \rfloor = 4$. The mapper then emits the following two key-value pairs:

map((store X, Person A, 32, PA) →
{(key = p_4, value = [(Store X, 32, PA), HS]),
(key = p_5, value = [(Store X, 32, PA), Ns])}

where HS indicates that p_4 is the home partition for this record and the record lies within the second half of its home partition, while NS indicates that p_5 is the partition to the immediate right of the home partition and the record lies within the second half of its home partition.

Similarly, for each record in store X and store Y two key-value pairs are produced containing the index of the home partition and its immediate right neighbor as their keys, respectively. The intermediate key-value pairs emitted by mappers are grouped by the partition number and sent to their corresponding partition (i.e., reducer).

Each reducer receives a partition number as a key and all the records sent to that partition as values. Based on their flags, it divides the records into four groups as shown in Fig. 5

- *Home First (HF)* The set of all records flagged as home and fall within the first half of the home partition.
- *Home Second (HS)* The set of all records flagged as home and fall within the second half of the home partition.

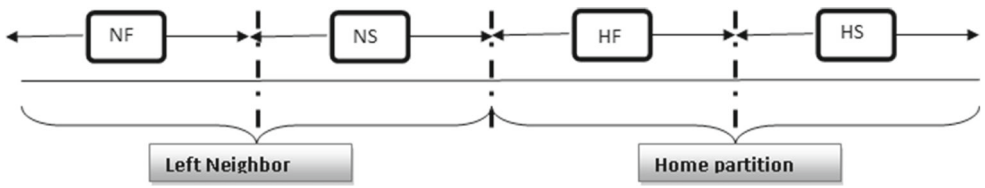

Fig. 5 Every reducer receives all records that fall within a partition and its left neighbor and divides the records into four groups: HF, HS, NF, NS. The lengths of the home partition and its left neighbor are equal to the length of α-cut of the fuzzy equal

- *Neighbor First (NF)* The set of all records flagged as neighbor and fall within the first half of the left neighbor.
- *Neighbor Second (NS)* The set of all records flagged as neighbor and fall within the second half of the left neighbor.

The reducer then combines the following records for each partition to produce the join result:

- All records from store X in (HF $\cup$ HS) must be combined with all records from store Y that fall within the same. This is because every two records that fall within the home partition have a distance less than the length of the α-cut.
- All records from store X (store Y) in HF must be combined with all records from store Y (store X) in NS.
- All records from store X (store Y) in HF must be combined with all records from store Y (store X) in NF if and only if their distance is less than the length of α-cut.
- All records from store X (store Y) in HS must be combined with all records from store Y (store X) in NS if and only if their distance is less than the length of α-cut.

The reducer also computes the degree to which the combined records satisfy the fuzzy equal condition based on Eq. 3.

Figure 6 illustrates the MapReduce data flow for fuzzy equal join on the two datasets in Table 3. The join is performed on the age column, and the reducers' outputs show all pairs of products that are purchased by people of an approximately same age (to degree $\geq$0.75). For each record it also shows the degree to which the joint result satisfies the fuzzy equal condition.

The MapReduce job for fuzzy equal join is summarized in algorithm 6. The Map function receives a single record r from relation R or S and, in line 3, it computes the home partition of r based on the length of the α-cut of fuzzy equal (L) and the lower bound on the domain of the join attribute (A.min). If r lies within the first half of its home partition, the Map function sends the record to its home partition and tags the record as HF (line 5). It also sends the record to the partition to the immediate right of the home partition and marks it as "NF" (line 6). Otherwise, if r lies within the second half of its home partition, it is marked as HS and sent to its home partition (line 8). It is also marked as NS and sent to the partition to the immediate right of the home partition (line 9). Therefore, each input record in R and S is sent to exactly two reducers.

Each reducer receives a set of records that are sent to partition for that reducer. Each record comes with the name of the relation it belongs to (R or S) as well as a flag which indicates if this reducer is the home or neighbor partition for this record and which half of the home partition the record falls into. It then runs a nested loop which combines all records from R with all the records from S which are tagged as HF or HS. In addition, it combines all R records tagged as HF with all S records tagged as NS (lines 5–7). Similarly, all R records tagged as HF are combined with all S records tagged as NF and all R records tagged as HS are combined with all S records tagged as NS if and only if their membership in fuzzy equal is at least α (lines 8–11). The membership of the joint tuple is equal to the minimum of the memberships of each tuple in its own relation as well as the degree to which the joint tuple satisfies the fuzzy equal condition (lines 6 and 10).

Lemma 1 *suppose that a pair of records $r \in R$ and $s \in S$ are fuzzy equal to a degree greater than or equal to a threshold α. Using algorithm 6, r and s are sent to at least one common reducer and the pair (r, s) is produced exactly once in the output.*

Proof Suppose L is the length of α-cut of fuzzy equal then s must be within the interval $[r - L, r + L]$. suppose p_h is the home partition for r. Hence, r is sent to reducers r_h (its home reducer) and r_{h+1} (the reducer to the immediate right of the home reducer). There are three possible cases:

1 The home partition of s is also p_h. In this case, s and r are both sent to r_h and r_{h+1}, but they are only combined in their home reducer and hence, (r, s) is produced only once in the output.
2 The home partition of s is p_{h-1}. In this case, s is sent to r_{h-1} and r_h. Hence, r and s go to exactly one common reducer, r_h, and the pair (r, s) is produced only once in the output.
3 The home partition of s is p_{h+1}. In this case, s is sent to r_{h+1} and r_{h+2}. Hence, r and s go to exactly one common reducer, r_{h+1}, and the pair (r, s) is produced only once in the output. □

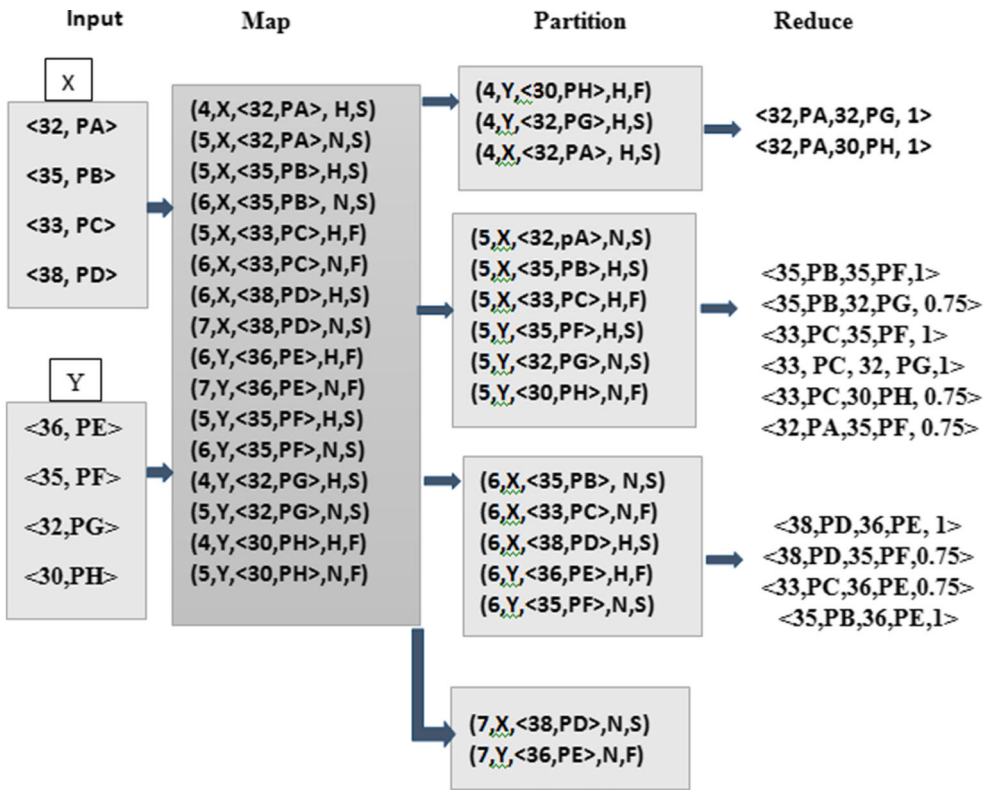

Fig. 6 MapReduce data flow for fuzzy equal join to return all pairs of products, from datasets X and Y, which were purchased by people of approximately the same age. The first and second columns in the input are "age" and "product_id," respectively. The last column in the output is the degree to which the joint record satisfy the fuzzy equal condition. For example, product PC from store X and PH from store Y are purchased by people approximately the same age to 0.75°

The Map and communication cost for algorithm 6 are both $M = C = 2(|R| + |S|)$ as each input record in R or S is sent to exactly two reducers. The reducer performs a cross product only on a subset of the values it receives. If we assume that the input data are distributed uniformly, every reducer on average receives about $\frac{2(|R|+|S|)}{n}$, where n is the number of partitions (i.e., the number of reducers). The reducer then performs a quadratic operation on four subpartitions: $\{HF, HS\}$, $\{HF, NS\}$, $\{HF, NF\}$ and $\{HS, NS\}$. Hence the total cost of all reducers is:

$$r = n \times O\left(4 \times \left(\frac{2(|R| + |S|)}{4n}\right)^2\right) = O\left(\frac{(|R| + |S|)^2}{n}\right)$$

Hence, the total cost of algorithm 6 is:

$$M + C + R = O((|R| + |S|)) + O((|R| + |S|)) + O\left(\frac{(|R| + |S|)^2}{n}\right)$$

In general case, however, the join key may not be partitioned uniformly across the reducers. In the worst case scenario, where all pairwise combination of R and S tuples satisfies the fuzzy joint condition, all tuples end up landing at the same partition and hence the worst cost will be $(|R| + |S|)^2$ as if the entire join is running on one partition. To alleviate this situation, when the dataset is skewed, the data must be repartitioned to balance the load among reducers. This will be discussed in the following section.

4 Join optimization and load balancing

A main drawback of most reduce-side join algorithms is their lower performance when the skewness is present in the dataset. The fuzzy equal join algorithm in the previous section divides the domain of the join attribute into a number of equal ranges. Each range is called a partition and records from each relation get assigned to these partitions based on which range their join attribute value falls into. If the join attributes are distributed uniformly, then it is reasonable to assume roughly the same number of records would be assigned to each partition, with each reducer handling an equal load. If the dataset is small, then one could also assume that each reducer could handle the data sent to it in short order. Unfortunately, all too often neither are the case

Algorithm 6 MapReduce job for Fuzzy equal join $R \bowtie S$

1: **Map:**
Input: r: an input record, K name of the relation to which r belongs (R or S), $\mu_K(r)$: membership of r in K, α: threshold value, L: length of α-cut of fuzzy equal, A: the join attribute, A.min: a predefined lower bound on the values of A
2: **if** $\mu_K(r) >= \alpha$ **then**
3: $p_h = \lfloor \frac{r.A - A.\min}{L} \rfloor$ {Compute the home partition}
4: **if** $(r.A - r.\min) \mod L < L/2$ **then**
5: emit key=p_h, value= <K,r,HF>
6: emit key=p_{h+1}, value= <K,r,NF>
7: **else**
8: emit key=p_h, value= <K,r,HS>
9: emit key=p_{h+1}, value= <K,r,NS>
10: **end if**
11: **end if**

1: **Reduce:**
value
Input: $= list < (K, r, flag) >$
2: **for** v1: value **do**
3: **for** v2: value **do**
4: **if** (v1.K= R **and** v2.K=S) **then**
5: **if** (v1.flag=(HF | HS) **and** v2.flag=(HF | HS)) **or** (v1.flag=HF **and** v2.flag=NS) **then**
6: emit key=(v1.r, v2.r)
value= $\min(\mu_{v1.K}(v1.r), \mu_{v2.K}(v2.r), \mu_{FEQ}(v1.r, v2.r))$
7: **end if**
8: **if** (v1.flag=HF **and** v2.flag=NF) OR (v1.flag=HS **and** v2.flag=NS) **then**
9: **if** $(\mu(FEQ(v1.r, v2.r)) >= \alpha)$ **then**
10: emit key=(v1.r, v2.r)
value= $\min(\mu_{v1.K}(v1.r), \mu_{v2.K}(v2.r), \mu_{FEQ}(v1.r, v2.r))$
11: **end if**
12: **end if**
13: **end if**
14: **end for**
15: **end for**

and some subpartitioning of the data is needed to ensure load balancing.

There are quite a few studies which have addressed the problem of load balancing in the presence of data skew. Some studies, such as (Kwon et al. 2012; Elmeleegy et al. 2014; Gufler et al. 2012; Ramakrishnan et al. 2012) extended the MapReduce framework to mitigate the skewness problem. Other papers (Atta et al. 2011; Zhang et al. 2012; Kyritsis et al. 2012; Hassan et al. 2014) modified the join algorithm to enforce load balancing. For the fuzzy equal join algorithm, we use the partitioning approach proposed in Zhang et al. (2012) to balance reducers' loads when data are skewed.

We assume a threshold value for the maximum number of records that a reducer can handle before being overloaded. Such threshold is determined based on the hardware properties of each reduce slot. The reducers that are expected to receive more records than the threshold value are repartitioned in such a way that all potentially similar values are able to be compared, but the load of an overloaded reducer is divided into a number of subpartitions. The more records

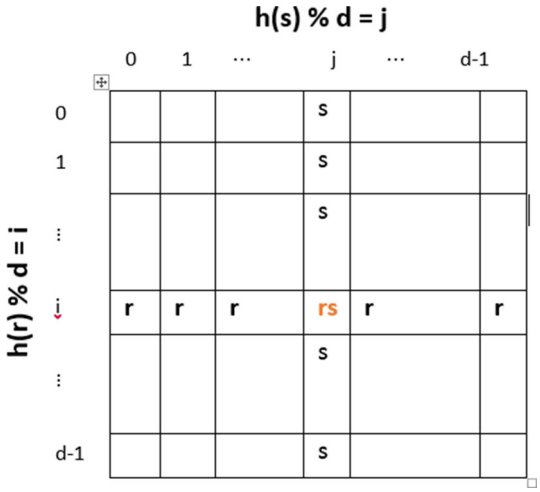

Fig. 7 Subpartitioning of an overloaded reducer, every record $r \in R$ is hashed to a row of subpartition matrix and every record $s \in S$ is hashed to a column of subpartition matrix. The pair (r, s) meet in exactly one subpartition

being sent to an overloaded reducer, the greater the number of subpartitions. The number of output records produced by an overloaded reducer is distributed into a matrix of $d \times d$ subpartitions such that the expected number of input records to each subpartition will be less than the threshold.

Suppose that an overloaded reducer receives $|r|$ and $|s|$ records from relations R and S, respectively. The maximum number of output records that can be produced by this reducer is $|r| \times |s|$. So, each subpartition is expected to produce at least $\frac{|r|\times|s|}{d\times d}$ records. To produce such output, each subpartition should at least receive $2 * \frac{\sqrt{|r|\times|s|}}{d}$ input records (Zhang et al. 2012). We want the number of input records to each subpartition to be less than a threshold; that is, $2 * \frac{\sqrt{|r|\times|s|}}{d} <= t$, where t is the threshold value. Consequently, to distribute the output of an overloaded reducer to $d \times d$ subpartitions such that each partition receives less than t records, d must be at least equal to $2 * \frac{\sqrt{|r|\times|s|}}{t}$.

A hash function is used to determine which subpartition every record is sent to in an overloaded reducer. If a record belongs to relation R in a RS join, then it is hashed to a row of the subpartition matrix, but if it belongs to relation S, it is hashed to a column of the subpartition matrix. This is illustrated in Fig. 7. This method guarantees that every pair of records from $r \in R$ and $s \in S$ will end up going to exactly one common subpartition and hence are joined in that subpartition (Fig. 7).

For example, suppose that in fuzzy equal join of Fig. 6 the reducer threshold is four records so the second and third partitions are overloaded and need to be repartitioned. The second partition has three records from relation x and three records from relation y; hence, this partition will be subpartitioned into $d \times d$ matrix of subpartitions where $d = \lceil 2 * \frac{\sqrt{3\times3}}{4} \rceil = 2$

. The third partition contains three records from relation x and two records from relation y; hence, this partition will be divided into $d \times d$ matrix of subpartitions where $d = \lceil 2 * \frac{\sqrt{3\times 2}}{4} \rceil = 2$.

If h is a hash function, suppose that $h(30) \mod 2 = 0$, $h(32) \mod 2 = 0$, $h(33) \mod 2 = 1$, $h(35) \mod 2 = 1$, $h(36) \mod 2 = 0$ and $h(38) \mod 2 = 0$. Then records in the second and third partitions are repartitioned as shown in Fig. 8.

Each record in relation x is sent to a row of the subpartition matrix, while each record in relation y is sent to a column of the subpartition matrix. For example, record (5, x,<32, PA>, N, S) is sent to the two subpartitions in row 0, while record (5, Y, <35, PF>, H, S) is sent to the two subpartitions in column 1. The records in each subpartition are then joined in a similar way to the non-overloaded partitions as explained in the previous section.

To implement the repartitioning in MapReduce, a simple MapReduce job is run to preprocess the input and count the number of R and S records being sent to each reducer in order to find the overloaded reducers. This is described in algorithm 7. The output of the preprocessing job is copied into the distributed cache for fast retrieval in the subsequent job.

Algorithm 7 preprocessing job: Finding the overloaded reducers

Map:

Input: r: an input record, K the name of the relation to which r belongs, $mu_K(r)$: membership of r in its relation, L: length of α-cut of fuzzy equal, A: the join attribute, A.min: a predefined lower bound on the values of A

2: **if** $\mu(r) >= \alpha$ **then**
   $p_h = \lfloor \frac{r.A - A.min}{L} \rfloor$ {Compute the home partition}
4: emit key=p_h, value=K
   emit key=p_{h+1}, value=K
6: **end if**

Reduce:

Input: key =p:a partition number value= $list < K >$
2: num_R_records = $count(K \in value : K = R)$
   num_S_records = $count(K \in value : K = S)$
4: **if** (num_R_records+num_S_records)>=threshold **then**
   emit key= p , value= (num_R_records, num_S_records)
6: **end if**

The Map function in the fuzzy equal join algorithm is modified as shown in algorithm 8 to subpartition the overloaded reducers. Lines 1–12 in this algorithm are the same as algorithm 6. Line 13 loads the set of overloaded partitions obtained from the preprocessing job. If the home partition or its immediate right neighbor belong to the overloaded set, they are repartitioned. The dimension of the subpartition matrix is computed based on the number of records sent to each partition and the threshold value (lines 15 and 28). Tecord r is then hashed and sent to a row of the subpartition matrix if it belongs to relation R; otherwise, it is sent to a column of the subpartition matrix (lines 16–22 and 29–36).

Algorithm 8 MapReduce job for Fuzzy equal join $R \bowtie S$ with load balancing

1: **Map:**
Input: r: an input record, K name of the relation to which r belongs (R or S), $\mu_K(r)$: membership of r in K, α: threshold membership value, L: length of α-cut of fuzzy equal, A: the join attribute, $A.min$: a predefined lower bound on the values of A, t:reducer threshold
2: **if** $\mu_K(r) <= \alpha$ **then**
3: return
4: **end if**
5: $p_h = \lfloor \frac{r.A - A.min}{L} \rfloor$ {Compute the home partition}
6: **if** $(r.A - r.min) \mod L < L/2$ **then**
7: value1= <K,r,HF>
8: value2= <K,r,NF>
9: **else**
10: value1= <K,r,HS>
11: value2= <K,r,NS>
12: **end if**
13: O= retrieve, from distributed cache, the set of overloaded partitions produced as the result of the preprocessing job
14: **if** $p_h \in O$ {The home partition is overloaded and requires repartitioning} **then**
15: d= $2 \times \frac{\sqrt{p_h.\text{num\_R\_records} \times p_h.\text{num\_S\_records}}}{t}$
16: $i = hash(r) \mod d$
17: **for** j=0...d **do**
18: **if** $r \in R$ **then**
19: emit key= $P_{h_{ij}}$ value= value1
20: **else**
21: emit key= $P_{h_{ji}}$ value= value1
22: **end if**
23: **end for**
24: **else**
25: emit key=p_h value=value1 {The home partition is not overloaded}
26: **end if**
27: **if** $p_{h+1} \in O$ {The right neighbor is overloaded and requires repartitioning.} **then**
28: d= $2 \times \frac{\sqrt{p_{h+1}.\text{num\_R\_records} \times p_{h+1}.\text{num\_S\_records}}}{t}$
29: $i = hash(r) \mod d$
30: **for** j=0...d **do**
31: **if** $r \in R$ **then**
32: emit key= $P_{h+1_{ij}}$ value= value2
33: **else**
34: emit key= $P_{h+1_{ji}}$ value= value2
35: **end if**
36: **end for**
37: **else**
38: emit key=p_{h+1} value=value2 {The right neighbor is not overloaded}
39: **end if**

5 Experimental results

To show the scalability of the MapReduce algorithms proposed in this paper for fuzzy relational operations, we used

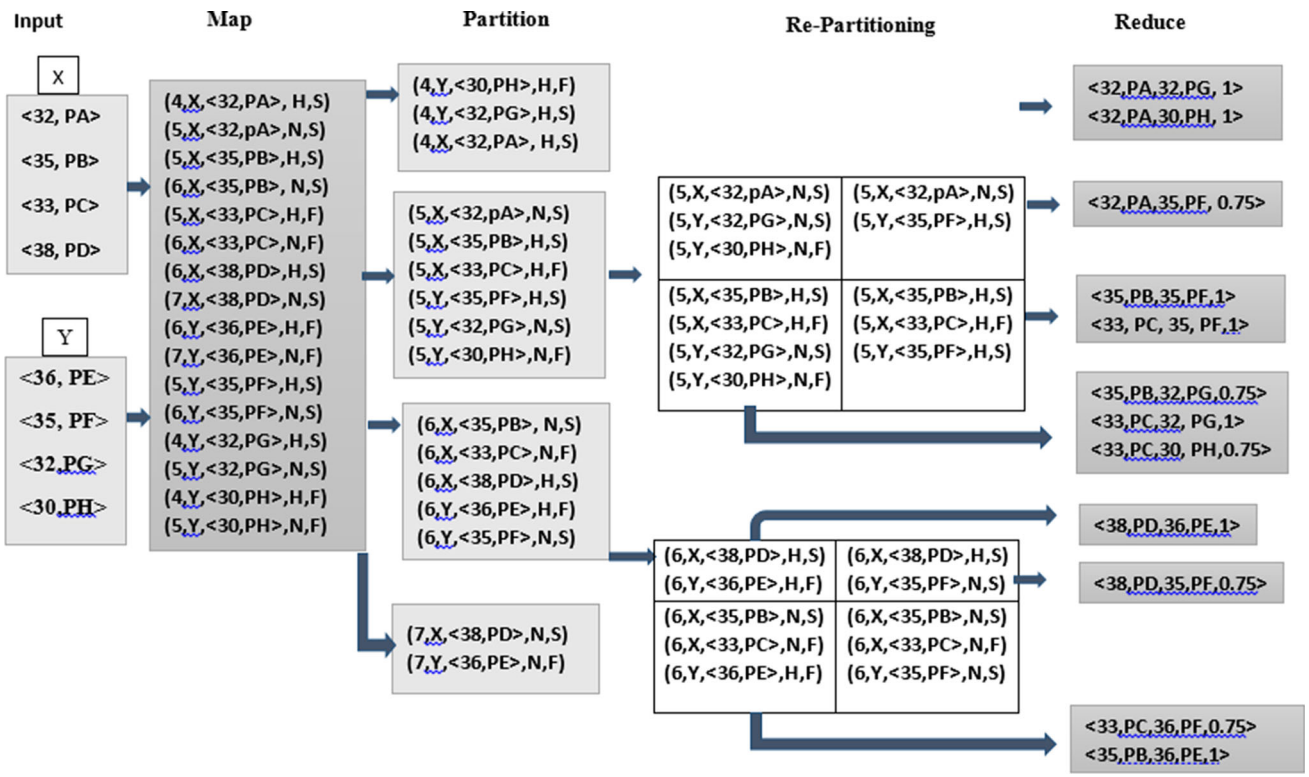

Fig. 8 MapReduce data flow of fuzzy equal join after repartitioning. The threshold is four records, the second and third partitions are repartitioned into four subpartitions

the flight dataset made available by the US department of transportation US (2016). The dataset contains flight information for nonstop domestic flights by major air carriers and includes items such as departure and arrival delays, origin and destination airports, flight numbers, nonstop distance. The dataset contains 164,219,718 records ranging from year 1987 to 2015.

The MapReduce algorithms for fuzzy selection, projection, union, intersection, and difference are all clearly linear as explained in Sect. 3.2; hence, this section is only dedicated to show the scalability of the fuzzy equal join algorithm. The join query that was posed to the dataset was "find all pairs of flights with an approximately equal flight distances." More formally,

$$\mu_{\text{flight}\bowtie_{\text{AED}}\text{flight}}(xy) = \min(\mu_{\text{flight}}(x), \mu_{\text{flight(y)}}, \mu_{\text{AE}}(x.\text{distanc}, y.\text{distance}))$$

where AE represents approximately equal and AED represents approximately equal distances and a fuzzy trapezoid membership is used to represent the linguistic term "approximately equal" for flight distance and the membership threshold is set to 0.9 to return all pairs of flights with distances of at least 90% approximately equal.

The fuzzy join algorithms with and without load balancing were run on varying data sizes on Amazon Elastic MapReduce, and the running time was measured for each data size. A total of 28 cores with 210 GB memory and 5 Terabyte SSD are used to run the experiment.

The query running time for each data size with and without load balancing is shown in Figs. 9 and 10, respectively. As illustrated in these figures, the running time of both fuzzy equal join algorithms grows linearly with the number of output records.[2] Both algorithm produced the same result; however, the running time of the fuzzy equal join algorithm with load balancing is slightly longer than the running time of the algorithm without load balancing, but this is to be expected as increasing the number of reducers in the algorithm with load balancing imposes some overhead. The fuzzy equal join algorithm without load balancing fails after 2 billion output records due to straggling reducer, while the algorithm with load balancing continues to grow linearly with the number of output records.

[2] As proven in Sect. 3.2.4 each output records is generated only once; therefore, it seems reasonable to measure the growth in terms of the number of output records.

No. InputRecords	No. OutputRecords	InputSize(bytes)	OutputSize(bytes)	Elasped Time(min)
100kX100k	116,201,179	5,333,084	97,348,434	1
200kX200k	540,769,476	10,714,919	583,100,024	3
300kX300k	1,275,951,978	16,122,782	1,548,704,801	7
400kX400k	2,187,183,711	21,531,197	2,447,384,786	11
500k X500k	The algorithm crashes			

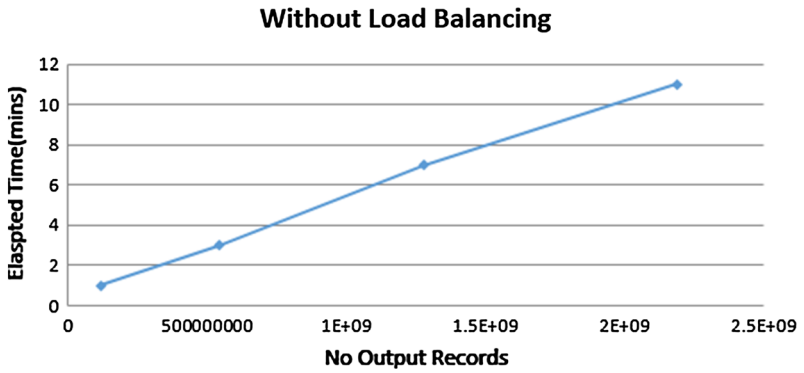

Fig. 9 The running time of the fuzzy equal join algorithm without load balancing for varying data sizes. The fuzzy equal join algorithm without load balancing fails after approximately 2 billion output records due to a skewed reducer running out of memory

No. InputRecords	No. OutputRecords	InputSize(bytes)	OutputSize(bytes)	Elasped Time(min)
100kX100k	116,201,179	53,33,084	97,348,434	1
200kX200k	540,769,476	10,714,919	583,100,024	4
300kX300k	1,275,951,978	16,122,782	1,548,704,801	9
400kX400k	2,187,183,711	21,531,197	2,447,384,786	17
500kx500k	3,362,703,589	26,965,653	1,909,288,520	28
1million X 1 million	13,738,358,722	53,921,834	3,320,396,742	111
1.5 million X 1.5 million	30,977,280,168	80,836,454	11,793,680,126	227
2 million X 2 million	55,812,622,460	107,701,367	21,248,999,677	374

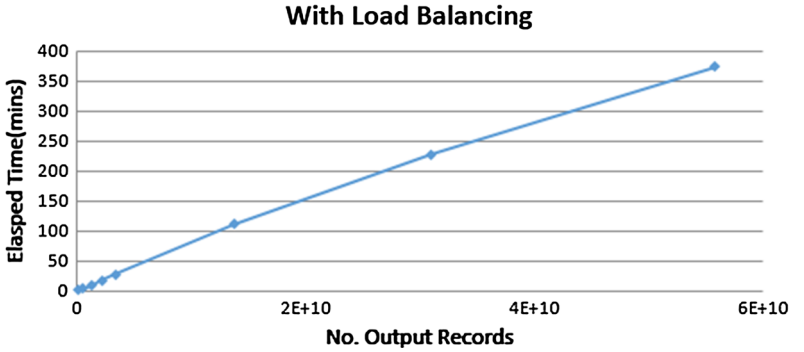

Fig. 10 The running time of the fuzzy equal join algorithm with load balancing for varying data sizes. The running time grows linearly in terms of the number of output records

6 Summary and future work

This paper reports the implementation of a scalable flexible relational algebra in MapReduce based on fuzzy set theory. The work provides a hybrid methodology which integrates fuzzy operations and big data. The algorithms proposed in the paper allow users to pose linguistic queries to a large-scale crisp dataset. The fuzzy operations discussed are fuzzy selection, fuzzy projection, fuzzy union, fuzzy intersection, fuzzy difference and fuzzy equal join. The cost of each algorithm is discussed in terms of the cost of the map and reduce functions as well as the communication cost. For future direction of this work, we will focus on two possible extensions of the current scalable fuzzy relational algebra:

1 The fuzzy join algorithm presented in this paper can only be applied when the join condition is fuzzy equal. An extension of the current framework will be considered when the join condition includes fuzzy greater than or fuzzy less than comparators.
2 The relational algebra presented in this work assumes that the linguistic queries are applied to a fuzzy relation where the attribute values are crisp but an entire tuple in the relation is associated with a membership degree. A more general framework can be developed where attribute values are also allowed to be linguistic or fuzzy. For example, the attribute "price" in a "product" relation can take linguistic values such as "expensive," "cheap," "pricey," "average" or any arbitrary fuzzy set on the price domain. Developing a scalable fuzzy join algorithm for such framework will be particularly challenging as the join will require partitioning the space of fuzzy sets on a given domain. The application of such framework will be in running linguistic queries against datasets gathered from weblogs, news or social media where data are mostly linguistic rather than numeric. In this case, the join is not merely performed by simple text similarity, but rather by the meaning of the linguistic values modeled by fuzzy sets.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This work does not contain any studies with human participants or animals performed by any of the authors.

References

Afrati FN, Sarma AD, Menestrina D, Parameswaran A, Ullman JD (2012) Fuzzy joins using mapreduce. In: 2012 IEEE 28th international conference on data engineering (ICDE). IEEE, pp 498–509

Atta F, Viglas SD, Niazi S (2011) Sand join: a skew handling join algorithm for google's mapreduce framework. In: 2011 IEEE 14th international multitopic conference (INMIC), pp 170–175. doi:10.1109/INMIC.2011.6151466

Bosc P, Prade H (1997) An introduction to the fuzzy set and possibility theory-based treatment of flexible queries and uncertain or imprecise databases. In: Motro A, Smets P (eds) Uncertainty management in information systems. Springer, New York, pp 285–324

Buckles BP, Petry FE (1982) A fuzzy representation of data for relational databases. Fuzzy Sets Syst 7(3):213–226. doi:10.1016/0165-0114(82)90052-5

Buckley JJ, Eslami E (2002) An introduction to fuzzy logic and fuzzy sets, vol 13. Springer, New York

Chen G (1998) Fuzzy logic in data modeling: semantics, constraints, and database design. Kluwer Academic Publishers, Norwell

Das Sarma A, He Y, Chaudhuri S (2014) Clusterjoin: a similarity joins framework using map-reduce. Proc VLDB Endow 7(12):1059–1070. doi:10.14778/2732977.2732981

Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

Dubois D, Prade H (1986) Weighted minimum and maximum operations in fuzzy set theory. Inf Sci 39(2):205–210. doi:10.1016/0020-0255(86)90035-6

Elmeleegy K, Olston C, Reed B (2014) Spongefiles: mitigating data skew in mapreduce using distributed memory. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. SIGMOD '14, pp. 551–562. ACM, New York. doi:10.1145/2588555.2595634

Galindo J (2005) Fuzzy databases: modeling, design and implementation: modeling, design and implementation IGI Global

Gufler B, Augsten N, Reiser A, Kemper A (2012) Load balancing in mapreduce based on scalable cardinality estimates. In: 2012 IEEE 28th international conference on data engineering (ICDE), pp 522–533. doi:10.1109/ICDE.2012.58

Hassan MAH, Bamha M, Loulergue F (2014) Handling data-skew effects in join operations using mapreduce. Proc Comput Sci 29:145–158. doi:10.1016/j.procs.2014.05.014. 2014 International conference on computational science

Klir GJ, Clair UHS, Yuan B (1997) Fuzzy set theory: foundations and applications. Prentice Hall. https://books.google.com/books?id=DNxQAAAAMAAJ

Kwon Y, Balazinska M, Howe B, Rolia J (2012) Skewtune: mitigating skew in mapreduce applications. In: Proceedings of the 2012 ACM SIGMOD international conference on management of data. SIGMOD '12, pp. 25–36. ACM, New York. doi:10.1145/2213836.2213840

Kyritsis V, Lekeas P, Souliou D, Afrati F (2012) A new framework for join product skew. In: Lacroix Z, Vidal M (eds) Resource discovery, vol 6799., Lecture notes in computer scienceSpringer, New York, pp 1–10

Ma ZM, Yan L (2010) A literature overview of fuzzy conceptual data modeling. J Inf Sci Eng 26(2):427–441

Ma ZM, Zhang WJ, Ma WY (2000) Semantic measure of fuzzy data in extended possibility-based fuzzy relational databases. Int J Intell Syst 15(8):705–716. doi:10.1002/1098-111X(200008)15:8705::AID-INT23.0.CO;2-4

Ma ZM, Mili F (2002) Handling fuzzy information in extended possibility-based fuzzy relational databases. Int J Intell Syst 17(10):925–942. doi:10.1002/int.10057

Medina JM, Vila MA, Cubero JC, Pons O (1995) Towards the implementation of a generalized fuzzy relational database model. Fuzzy Sets Syst 75(3):273–289. doi:10.1016/0165-0114(94)00380-P

Metwally A, Faloutsos C (2012) V-smart-join: a scalable mapreduce framework for all-pair similarity joins of multisets and vectors. Proc VLDB Endow 5(8):704–715

Petry FE (ed) (1997) Fuzzy databases: principles and applications. Kluwer Academic Publishers, Norwell

Prade H, Testemale C (1984) Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries. Inf Sci 34(2):115–143. doi:10.1016/0020-0255(84)90020-3

Ramakrishnan SR, Swart G, Urmanov A (2012) Balancing reducer skew in mapreduce workloads using progressive sampling. In: Proceedings of the 3rd ACM symposium on cloud computing. SoCC '12, pp 16–11614. ACM, New York. doi:10.1145/2391229.2391245

Shenoi S, Melton A (1989) Proximity relations in the fuzzy relational database model. Fuzzy Sets Syst 31(3):285–296. doi:10.1016/0165-0114(89)90201-7

Shenoi S, Melton A (1990) An extended version of the fuzzy relational database model. Inf Sci 52(1):35–52. doi:10.1016/0020-0255(90)90034-8

US (2016) Department of transportation. Online; accessed 23 Feb 2016. https://www.transportation.gov/

Vasant P (2013) Handbook of research on novel soft computing intelligent algorithms: theory and practical applications. Advances in computational intelligence and robotics (ACIR) book series. IGI Global. https://books.google.com/books?id=nt-WBQAAQBAJ

Vernica R, Carey MJ, Li C (2010) Efficient parallel set-similarity joins using mapreduce. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data, pp. 495–506. ACM

Wang Y, Metwally A, Parthasarathy S (2013) Scalable all-pairs similarity search in metric spaces. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '13, pp. 829–837. ACM, New York. doi:10.1145/2487575.2487625

Zadeh LA (1965) Fuzzy sets. Inf Control 8(3):338–353. doi:10.1016/S0019-9958(65)90241-X

Zadeh LA (1999) Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets Syst 100 Suppl 1(0):9–34. doi:10.1016/S0165-0114(99)80004-9

Zhang C, Li J, Wu L, Lin M, Liu W (2012) Sej: an even approach to multiway theta-joins using mapreduce. In: 2012 Second international conference on cloud and green computing (CGC), pp 73–80. doi:10.1109/CGC.2012.9