Knowl Inf Syst (2012) 31:547–579
DOI 10.1007/s10115-011-0413-5

REGULAR PAPER

# On processing continuous frequent $K$-$N$-match queries for dynamic data over networked data sources

**Shih-Chuan Chiu · Jiun-Long Huang · Jen-He Huang**

Received: 9 December 2010 / Revised: 31 March 2011 / Accepted: 6 May 2011 /
Published online: 20 May 2011


**Abstract** Similarity search is one of the critical issues in many applications. When using all attributes of objects to determine their similarity, most prior similarity search algorithms are easily influenced by a few attributes with high dissimilarity. The *frequent k-n-match query* is proposed to overcome the above problem. However, the prior algorithm to process frequent $k$-$n$-match queries is designed for static data, whose attributes are fixed, and is not suitable for dynamic data. Thus, we propose in this paper two schemes to process continuous frequent $k$-$n$-match queries over dynamic data. First, the concept of *safe region* is proposed and four formulae are devised to compute safe regions. Then, scheme CFKNMatchAD-C is developed to speed up the process of continuous frequent $k$-$n$-match queries by utilizing safe regions to avoid unnecessary query re-evaluations. To reduce the amount of data transmitted by networked data sources, scheme CFKNMatchAD-C also uses safe regions to eliminate transmissions of unnecessary data updates which will not affect the results of queries. Moreover, for large-scale environments, we further propose scheme CFKNMatchAD-D by extending scheme CFKMatchAD-C to employ multiple servers to process continuous frequent $k$-$n$-match queries. Experimental results show that scheme CFKNMatchAD-C and scheme CFKNMatchAD-D outperform the prior algorithm in terms of average response time and the amount of produced network traffic.

**Keywords** Similarity search · $k$-$n$-match problem · Continuous query · Dynamic data

S.-C. Chiu · J.-L. Huang (✉) · J.-H. Huang
Department of Computer Science, National Chiao Tung University,
Hsinchu, Taiwan, ROC
e-mail: jlhuang@cs.nctu.edu.tw

S.-C. Chiu
e-mail: scchiu@cs.nctu.edu.tw

J.-H. Huang
e-mail: jenho@cs.nctu.edu.tw

## 1 Introduction

Similarity search is one of the critical issues in many applications such as pattern recognition, content-based retrieval, data mining, location-based services, and bio-informatics [2,10,15,16,24,36]. Generally, the objects are represented as points in a multi-dimensional space, and similarity search is transformed into nearest neighbor (abbreviated as NN) search which is to find the object closest to a query point. To facilitate similarity search, a similarity function is employed to aggregate all attributes of two objects into one score, which indicates how similar these two objects are. Several approaches were proposed in the literature to efficiently process NN queries [11,13,16,25,32].

In applications such as stock markets, moving objects, sensor networks, and intrusion detection systems, a large volume of data are stored in databases and the values of these data usually change frequently [27,37]. Thus, the result of a query may change as the attributes of objects change. Unfortunately, most NN query processing algorithms are designed for static data whose attributes are static and are not suitable for dynamic data. Such phenomenon calls for *continuous* NN queries that continuously monitor the dynamic data and report the latest query results as long as the changes of attributes make the query results become invalid [12,23,29].

Most prior work uses all features/dimensions to measure similarity. However, as mentioned in [30], considering all features in similarity measurement has the drawback that *a few dimensions with high dissimilarity may greatly affect the similarity*. Consider the example shown in Table 1. When Euclidean distance is used as the measurement of similarity, the nearest neighbor of object $A$ is object $C$. However, we can observe that object $B$ is more similar to object $A$ than object $C$ in seven dimensions. Since in many real applications, high dissimilarity dimensions usually result from wrong readings or noises, Tung et al. argued in [30] that one should try to reduce the influences of these high dissimilarity dimensions to make similarity search more robust. Thus, *frequent k-n-match query* was proposed in [30] to solve the problem caused by the dimensions of high dissimilarity. In addition, algorithm FKNMatchAD was proposed in [30] to process frequent $k$-$n$-match queries.[1]

Although being able to efficiently process frequent $k$-$n$-match queries, algorithm FKNMatchAD has the following two drawbacks:

(1) *Algorithm FKNMatchAD is not suitable for dynamic data.*
Taiwan government is establishing digital cameras in the intersections of important roads. Each camera is equipped with a embedded computer to recognize the attributes (e.g., color, type, direction, speed, license number) of all vehicles captured by the camera. Since the recognizing algorithm is not perfect, the recognized attributes usually consist of noises. Thus, frequent $k$-$n$-match queries are used to search for the vehicles possessing the given attributes. To track for a suspicious vehicle, one may issue a continuous frequent $k$-$n$-match query to continuously search for the vehicles possessing the given attributes. Since the vehicles captured by cameras usually change, the attributes recognized by the cameras are dynamic. Such application calls for an efficient scheme to process continuous frequent $k$-$n$-match queries on dynamic data.
Unfortunately, algorithm FKNMatchAD is designed for static data and is not suitable for dynamic data. A naive scheme to process continuous frequent $k$-$n$-match queries on dynamic data is to apply algorithm FKNMatchAD to re-evaluate all queries once the value of a data point changes. However, due to the reason that not all data changes will

[1] The definition of frequent $k$-$n$-match problem and the details of algorithm FKNMatchAD will be given in Sect. 2.

**Table 1** An example

| Object | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $d_6$ | $d_7$ | $d_8$ |
|---|---|---|---|---|---|---|---|---|
| A | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| B | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 100 |
| C | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |

affect the results of queries, the naive scheme will perform many unnecessary query re-evaluations and thus prolong average response time. Such drawback motivates us to devise an efficient algorithm to process continuous frequent $k$-$n$-match queries over dynamic data.

(2) *Algorithm FKNMatchAD is not suitable for large-scale environments.* Since algorithm FKNMatchAD processes frequent $k$-$n$-match queries in a centralized manner, the average response time of algorithm FKNMatchAD becomes much longer as the number of data points increases, thereby not suitable for large-scale environments. To solve this problem, similar to [31], we propose a decentralized scheme to utilize multiple servers to efficiently process continuous frequent $k$-$n$-match queries in large-scale environments.

In view of this, we address in this paper the problem of processing continuous frequent $k$-$n$-match queries over dynamic data. It is obvious that if the attributes of the objects do not significantly change, the changes of the similarities among objects and the query point are of highly likelihood to be insignificant. Therefore, not all changes will affect the results of queries, and there is much redundant work if we re-evaluate all queries every time a data point changes [8]. Based on this observation, we first introduce the concept of *safe regions*[2] which are defined as the intervals that the result of the query will not change as long as each attribute of each point is within its safe region. We then design scheme CFKNMatchAD-C to utilize safe regions to avoid unnecessary query re-evaluations. Algorithm INC-FKNMatchAD, which is the incremental version of algorithm FKNMatchAD, is also developed to speed up query processing. Specifically, when a user submits a continuous frequent $k$-$n$-match query, scheme CFKNMatchAD-C first applies algorithm FKNMatchAD to compute the query result. In addition, scheme CFKNMatchAD-C also computes the *safe region* of every attribute of each data object for the query. When the server receives a change of a data object, scheme CFKNMatchAD-C first checks whether the new value of the attribute is out of its safe region. If not, the query re-evaluation can be ignored since the change of the data object does not affect the result of the query. Otherwise, scheme CFKNMatchAD-C performs algorithm INC-FKNMatchAD to re-evaluate the query. As a result, scheme CFKNMatchAD-C is able to greatly reduce average response time of queries by avoiding unnecessary query re-evaluations and performing incremental query re-evaluations.

A *data source* is a device that owns or monitors the value(s) of a data point(s). When getting the new value of a data point, the data source will send a *data update* to the server to notify the server about the new value of the data point. For *networked data sources* such as sensors, sending data updates to the server will increase the load of the server and produce more network traffic. Thus, for the applications that the server need not to keep the latest

[2] The concept of safe regions has been widely used in spatial queries over moving objects [12,23] and monitoring sensor networks [28,34]. The detailed definition of safe regions used in this paper will be given in Sect. 3.

**Table 2** Notations

| Notation | Meaning |
|---|---|
| $DB$ | The database, which is a set of data points |
| $c$ | Cardinality of the database |
| $d$ | Dimensionality of the data space |
| $k$ | The number of $n$-match points to return |
| $n$ | The number of dimensions to match |
| $P$ | A data point |
| $p_i$ | The coordinate of $P$ in the $i$th dimension |
| $Q$ | The query point |
| $q_i$ | The coordinate of $Q$ in the $i$th dimension |

values of data points, a data update is called *unnecessary* if the data update will not affect the result of each query. Therefore, scheme CFKNMatchAD-C also utilizes safe regions to (1) eliminate the server's burden on handling unnecessary data updates and (2) reduce the amount of network traffic produced by data sources by means of avoiding transmissions of unnecessary data updates between data sources and the server. Finally, for large-scale environments with a lot of data points, we also design scheme CFKNMatchAD-D by extending scheme CFKMatchAD-C to use multiple servers to efficiently process continuous frequent $n$-$k$-match queries. To the best of our knowledge, there is no other prior work dealing with processing continuous frequent $k$-$n$-match queries over dynamic data. This characteristic distinguishes our paper from others.

The rest of this paper is organized as follows. Section 2 presents the definition of frequent $k$-$n$-match query and the details of algorithm FKNMatchAD. Section 3 describes the idea of safe regions and the details of scheme CFKNMatchAD-C. The design of scheme CFKNMatchAD-D is presented in Sect. 4. Performance evaluations of these schemes are given in Sect. 5, while the related work is discussed in Sect. 6. Finally, Sect. 7 makes a conclusion for this paper.

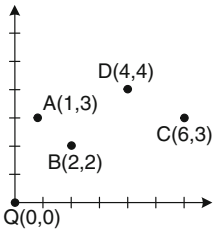
## 2 Preliminaries

### 2.1 Problem definition

The *k-n-match problem*, which is proposed in [30], is to find $k$ objects that are the most similar to the query object and $n$ is an integer which is not larger than the dimensionality of a data object (denoted as $d$). As the example shown in Table 1, some features of data items are not significant enough to present characteristics of data items. Thus, instead of comparing data objects in all dimensions, we compare data objects to query point by $n$ significant dimensions. The notations used are listed in Table 2 for better readability.

We follow the notations given in [30]. Objects from the database are considered as multi-dimensional data points.[3] A database is considered as a set of $d$-dimensional data points, where $d$ is the dimensionality. The $n$-match difference, which is defined as below, is used to measure the similarity of two data points.

[3] 'Object' and 'data point' are used interchangeably in this paper.




(a)

| Data Point | $(\delta_1, \delta_2)$ | $(\delta_1', \delta_2')$ | 1-match difference | 2-match difference |
|---|---|---|---|---|
| $A$ | (1,3) | (1,3) | 1 | 3 |
| $B$ | (2,2) | (2,2) | 2 | 2 |
| $C$ | (6,3) | (3,6) | 3 | 6 |
| $D$ | (4,4) | (4,4) | 4 | 4 |

(b)

**Fig. 1** An example of the $n$-match problem. **a** Data points. **b** Calculations of 1-match and 2-match differences

**Definition 1** (*The n-match difference*) Consider two $d$-dimensional points $P(p_1, p_2, \ldots, p_d)$ and $Q(q_1, q_2, \ldots, q_d)$. Let $\delta_i = |p_i - q_i|, i = 1, \ldots, d$. Sort $\{\delta_1, \delta_2, \ldots, \delta_d\}$ in increasing order and let the sorted result be $\{\delta_1', \delta_2', \ldots, \delta_d'\}$. The *n-match difference of point P with respect to point Q* is defined as $\delta_n'$.

With the definition of $n$-match difference, the $n$-match problem is defined as follows.

**Definition 2** (*The n-match problem*) Given a set of $d$-dimensional points in a database $DB$, an *n-match query* $\langle Q, n\rangle$, where $Q$ is the query point and $n$ is an integer ($1 \leq n \leq d$), is to search $DB$ for the data point $P$ which has the smallest $n$-match difference with respect to $Q$. $P$ is called the *n-match point* of $Q$.

We use the following example to illustrate $n$-match queries.

*Example 1* Figure 1 shows a two-dimensional space with four data points $A(1, 3)$, $B(2, 2)$, $C(3, 6)$, and $D(4, 4)$. Consider an $n$-match query $\langle Q(0, 0), 1\rangle$. The result of the query is $A$ because $A$ is of the smallest 1-match difference with respect to $Q$. When we set $n$ to 2, $B$ becomes the result because it has the smallest 2-match difference with respect to $Q$.

The $k$-$n$-match problem is then defined as below.

**Definition 3** (*The k-n-match problem*) Given a set of $d$-dimensional data points in $DB$ of cardinality $c$, a *k-n-match query* $\langle Q, n, k\rangle$, where $Q$ is a query point, $n$ is an integer ($1 \leq n \leq d$), and $k$ is an integer ($k \leq c$), is to search $DB$ for $k$ data points such that the $n$-match differences of these $k$ points are less than or equal to the other points in $DB$. These $k$ data points are called the $k$-$n$-match set of $Q$.

Unfortunately, it is difficult for users to set a proper value of $n$ for each application. Thus, Tung et al. proposed the *frequent k-n-match query* in [30] to solve this problem. In frequent $k$-$n$-match problem, instead of determining a single value for $n$, users are asked to set a range of $n$, say $[n_0, n_1]$. With $[n_0, n_1]$, the frequent $k$-$n$-match problem is to try each value of $n$ in the range and to find $k$ points that appear the most frequently in the $k$-$n$-match answer sets for all the $n$ values. The definition of *frequent k-n-match problem* is as follows.

**Definition 4** (*The frequent k-n-match problem*) Given a set of $d$-dimensional points in $DB$ of cardinality $c$, and a frequent $k$-$n$-match query $\langle Q, [n_0, n_1], k\rangle$ where $Q$ is a query point, $[n_0, n_1]$ is an interval within $[0, d]$, and $k$ is an integer ($k \leq c$), let $S_{n0}, \ldots, S_{n1}$ be the result sets of $k - n_0 - match, \ldots, k - n_1 - match$, respectively. The frequent $k$-$n$-match problem is to find a set $T$ of $k$ points, so that for any point $P_1 \in T$ and any point $P_2 \in DB - T$, $P_1$'s number of appearances in $S_{n0}, \ldots, S_{n1}$ is larger than or equal to $P_2$'s number appearances in $S_{n0}, \ldots, S_{n1}$. $T$ is called the frequent $k$-$n$-match set of $Q$.

**Table 3** An example database

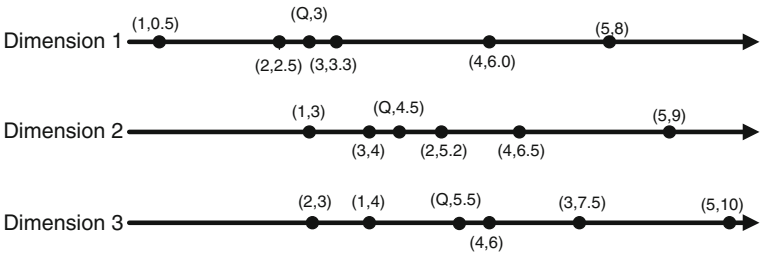
| Object ID | $d_1$ | $d_2$ | $d_3$ |
|---|---|---|---|
| 1 | 0.5 | 3.0 | 4.0 |
| 2 | 2.5 | 5.2 | 3.0 |
| 3 | 3.3 | 4.0 | 7.5 |
| 4 | 6.0 | 6.5 | 6.0 |
| 5 | 8.0 | 9.0 | 10.0 |

### 2.2 Algorithm FKNMatchAD

Algorithm AD is proposed in [30] to efficiently process the $k$-$n$-match queries and the frequent $k$-$n$-match queries. Algorithm AD consists of two versions, algorithm KNMatchAD and algorithm FKNMatchAD, which are for the $k$-$n$-match problem and the frequent $k$-$n$-match problem, respectively. Algorithm KNMatchAD uses the following data structures to maintain the necessary information while processing a query. $appear[i]$ maintains the number of appearances of point $i$ and each element of $appear[i]$ is initialized to 0. $h$ maintains the number of points appearing $n$ times and is initialized to 0. $S$ is the answer set and is initialized to $\phi$. $g[\,]$ is an array of size $2d$ to maintain the next attribute to access in each dimension in both directions. $g[\,]$ is viewed as $2d$ dimensions. The direction toward smaller values in dimension $i$ corresponds to $g[2\times(i-1)]$, while the direction toward larger values in dimension $i$ corresponds to $g[2\times(i-1)+1]$. Each element in $g[\,]$ is a triple $(pid, pd, dif)$, where $pid$ is the id of the data point, $pd$ is the dimension, and $dif$ is the difference between $q_{pd}$ and the next attribute to access in dimension $pd$.

We use the following example to illustrate how algorithm KNMatchAD processes a $k$-$n$-match query. Consider the database shown in Table 3. Suppose that a user issues a query $\langle Q(3.0, 4.5, 5.5), 2, 2\rangle$ where $k = n = 2$. The steps of algorithm KNMatchAD to process the query are as follows.

- *Step 1:* Locate $q_i$ in each dimension $i$ and the result is shown in Fig. 2a. In each dimension, the tuple $(a, b)$ indicates that the value of data point $b$ in this dimension is $a$. For example, (2,2.5) in dimensional 1 indicates that the value of point 2 in dimension 1 is 2.5, while $(Q, 3)$ indicates that the value of the query point in dimension 1 is 3.
- *Step 2:* Find the smallest difference between each data point and $Q$ in every dimension using binary search toward larger or smaller attribute directions and store in $g[\,]$. $g[\,]$ has six triples that are $\{(2, 0, 0.5), (3, 1, 0.3), (3, 2, 0.5), (2, 3, 0.7), (1, 4, 1.5), (4, 5, 0.5)\}$. Consider the triple $(2, 0, 0.5)$. The value of the second item being 0 indicates the smaller direction in dimension 1. Thus, the triple $(2, 0, 0.5)$ indicates that in dimension 1, the nearest data point toward smaller direction is data point 2 and the distance between data point 2 and the query point in dimension 1 is 0.5. Similarly, in the triple $(3, 1, 0.3)$, the second item being 1 indicates the larger direction in dimension 1. Thus, the triple $(3, 1, 0.3)$ means that in dimension 1, the nearest data point toward larger direction is data point 3 and the distance between data point 3 and the query point in dimension 1 is 0.3.
- *Step 3:* Get the triple $(3, 1, 0.3)$ with the smallest $dif$ in $g[\,]$ and $appear[3]$ is increased by one. Then, find the next smaller difference in dimension 1 toward larger attribute direction. (4, 6.0) is found, and the triple $(4, 1, 3.0)$ is placed in $g[1]$.
- *Step 4:* Get the triple $(2, 0, 0.5)$ with the smallest $dif$ and $appear[2]$ is increased by one. Find the next smaller difference in dimension 1 toward smaller attribute direction. $(1, 0.5)$ is found and the triple $(1, 0, 2.5)$ is placed in $g[0]$.




**(a)** Step 1

| | |
|---|---|
| appear[ ] | {0,0,0,0,0} |
| g[ ] | {(2,0,0.5),(3,1,0.3),(3,2,0.5),(2,3,0.7),(1,4,1.5),(4,5,0.5)} |
| S[ ] | {} |
| h | 0 |

**(b)** Step 2

| | |
|---|---|
| appear[ ] | {0,0,1,0,0} |
| g[ ] | {(2,0,0.5),(4,1,3.0),(3,2,0.5),(2,3,0.7),(1,4,1.5),(4,5,0.5)} |
| S[ ] | {} |
| h | 0 |

**(c)** Step 3

| | |
|---|---|
| appear[ ] | {0,1,1,0,0} |
| g[ ] | {(1,0,2.5),(4,1,3.0),(3,2,0.5),(2,3,0.7),(1,4,1.5),(4,5,0.5)} |
| S[ ] | {} |
| h | 0 |

**(d)** Step 4

| | |
|---|---|
| appear[ ] | {0,1,2,0,0} |
| g[ ] | {(1,0,2.5),(4,1,3.0),(1,2,1.5),(2,3,0.7),(1,4,1.5),(4,5,0.5)} |
| S[ ] | {3} |
| h | 1 |

**(e)** Step 5

| | |
|---|---|
| appear[ ] | {0,1,2,1,0} |
| g[ ] | {(1,0,2.5),(4,1,3.0),(1,2,1.5),(2,3,0.7),(1,4,1.5),(3,5,2.0)} |
| S[ ] | {3} |
| h | 1 |

**(f)** Step 6

| | |
|---|---|
| appear[ ] | {0,2,2,1,0} |
| g[ ] | {(1,0,2.5),(4,1,3.0),(1,2,1.5),(4,3,2.0),(1,4,1.5),(3,5,2.0)} |
| S[ ] | {3,2} |
| h | 2 |

**(g)** Step 7

**Fig. 2** Illustration of algorithm KNMatchAD

- *Step 5:* Get the triple (3, 2, 0.5) with the smallest $dif$ and $appear[3]$ is increased by one. Since $appear[3]$ equals 2 (i.e., $n$), data point 3 is inserted into $S$ and $h$ is increased by 1. Then, find the next smaller difference in dimension 2 toward smaller attribute direction. (1, 3.0) is found, and the triple (1, 2, 1.5) is placed in $g[2]$.
- *Step 6:* Get the triple (4, 5, 0.5) with the smallest $dif$ and $appear[4]$ is increased by one. Find the next smaller difference in dimension 3 toward larger attribute direction. (3, 7.5) is found and the triple (3, 5, 2.0) is placed in $g[5]$.

- *Step 7:* Get the triple (2, 3, 0.7) with the smallest $dif$ and $appear[2]$ is increased by one. Since $appear[2]$ equals $n$, data point 2 is inserted into $S$ and $h$ is increased by 1. Finally, algorithm KNMatchAD terminates and returns $S$ (i.e., data point 3 and data point 2) as the result of the query since the $h$ equals $k$.

Figure 2b–g show the values of all data structures used in algorithm KNMatchAD from step 2 to step 7 during processing the given $k$-$n$-match query.

**Definition 5** (*The mth match*) For a data point $pid$, we say that the $pd$th dimension of data point $pid$ contributes *one match* to a query $\mathcal{Q}$ if the triple $(pid, pd, dif)$ is popped from $g[\,]$ and makes $appear[pid]$ increase by one. In addition, we say that the $pd$th dimension of data point $pid$ contributes *the mth match* if the triple makes $appear[pid]$ increase from $m-1$ to $m$.

Algorithm FKNMatchAD is similar to algorithm KNMatchAD. The difference is that algorithm FKNMatchAD has to monitor the number of appearances of points in the interval $[n_0, n_1]$ given by the frequent $k$-$n$-match query $\langle Q, [n_0, n_1], k\rangle$. In addition, data structures $h[\,]$ and $S[\,]$ displace $h$ and $S$ to maintain the appearances of points and results. In algorithm FKNMatchAD, if $appear[i]$ is within $[n_0, n_1]$, point $i$ will be added to result set $S_{appear[i]}$ and $h_{appear[i]}$ will increase by one. Algorithm FKNMatchAD stops until $h[n_1]$ equals $k$. Finally, algorithm FKNMatchAD scans $S[\,]$ to obtain the identities of the $k$ points that appear most times in $S[\,]$. The algorithmic form of algorithm FKNMatchAD is as below. Interested readers can refer to [30] for the details of algorithm KNMatchAD and algorithm FKNMatchAD.

**Algorithm** FKNMatchAD

1: Initialize $appear[\,]$, $h[\,]$, $S[\,]$
2: **for** every dimension $i$ **do**
3: Locate $q_i$ in dimension $i$.
4: Calculate the differences between $q_i$ and its closest attributes in dimension $i$ along both directions. Form a triple $(pid, i, dif)$ for each direction. Put this triple to $g[i]$.
5: **do**
6: $(pid, pd, dif) = smallest(g)$;
7: $appear[pid]++$;
8: **if** $n_0 \le appear[pid] \le n_1$ **then**
9: $h[appear[pid]]++$;
10: $S[appear[pid]] = S[appear[pid]] \cup pid$;
11: Read the next attribute from dimension $pd$ and form a new triple $(pid, pd, dif)$. If end of the dimension is reached, let $dif$ be $\infty$. Put the triple to $g[pd]$.
12: **while** $h[n_1] < k$
13: Scan the top $k$ elements of $S_{n_0}, \ldots, S_{n_1}$ to obtain the identities of the $k$ points that appear most times in $S[\,]$.

## 3 Scheme CFKNMatchAD-C: continuous frequent $K$-$N$-match query processing in centralized environments

Since the $k$-$n$-match problem is a special case of the frequent $k$-$n$-match problem with $n_0 = n_1 = n$, we focus on the frequent $k$-$n$-match problem in the rest of this paper. Generally, there are two types of continuous queries: event based and time based. For event-based continuous

queries, the server reports the valid top-$k$ points for the queries once an attribute of a point changes. Query re-evaluation is driven by each data update. For time-based continuous query, a client will specify a report period, and the server should find the valid top-$k$ points for the query and report them for every report period. Query re-evaluation is performed periodically. In this paper, we focus on event-based continuous queries and the proposed schemes can be also applied to time-based continuous queries.

In this subsection, we propose scheme CFKNMatchAD-C to efficiently process continuous frequent $k$-$n$-match queries by avoiding unnecessary query re-evaluations and performing incremental query re-evaluations in centralized environments. To achieve this, we propose the idea of *safe region* and design a method to calculate safe regions of all attributes of points for each query. Safe regions are intervals that the result of a query will not change as long as each attribute of each point is within its safe regions. With the aid of safe regions, the query processor can determine whether a data update will affect the result of a query, and thereby be able to avoid unnecessary query re-evaluations. In addition, we also develop an incremental version of algorithm FKNMatchAD, called algorithm INC-FKNMatchAD, to facilitate fast query re-evaluations. The proposed centralized architecture is depicted in Sect. 3.1. The overview of scheme CFKNMatchAD-C is given in Sect. 3.2. Algorithm INC-FKNMatchAD is described in Sect. 3.3 while the calculation method of safe regions is depicted in Sect. 3.4. The proposed unnecessary data update elimination algorithm is shown in Sect. 3.5. Complexity analysis of scheme CFKNMatchAD-C is shown in Sect. 3.6.

### 3.1 The centralized system architecture

As shown in Fig. 3, the system for similarity search applications usually has a centralized server with many data sources. The server consists of two tables: query table and data table. The query table stores the information of the registered queries, while the data table stores the latest values of all data points. When receiving a continuous frequent $k$-$n$-match query issued by a client, the query processor will register the query into the query table. In addition, the query processor will process this query immediately and return the result to the client. If the values of a data point, say $P$, change, the corresponding data source will send a data update containing the new values of the data points to the server. The format of a data update is $\langle P, (p'_1, p'_2, \ldots, p'_d)\rangle$ where $(p'_1, p'_2, \ldots, p'_d)$ are the values of the attributes of the data point $P$. Once receiving a data update, the update processor will update the values of the data point $P$ stored in the data table. Since the change of the values of a data point may change the results of some queries, once receiving a data update, the query processor is responsible for sending the latest results of the queries to the users.

A naive scheme to provide the latest query results to clients is to ask the query processor to re-evaluate all registered queries when the update processor receives a data update. However, due to the reason that not all data updates will affect the results of the registered queries, the naive scheme will perform many unnecessary query re-evaluations, thereby prolonging average response time. Such drawback motivates us to design scheme CFKNMatchAD-C to efficiently process continuous frequent $k$-$n$-match queries by avoiding unnecessary query re-evaluations. In essence, scheme CFKNMatchAD-C consists of the following components.

- *Procedure OnReceivingQuery:* When a user submits a continuous frequent $k$-$n$-match query $\mathcal{Q}$, the query processor executes procedure OnReceivingQuery to obtain the query result as well as the corresponding safe regions.
- *Procedure OnReceivingUpdate:* When the value of a data point changes, the update processor executes procedure OnReceivingUpdate to check whether the date update will change the result of each registered query and re-evaluate the affected queries.

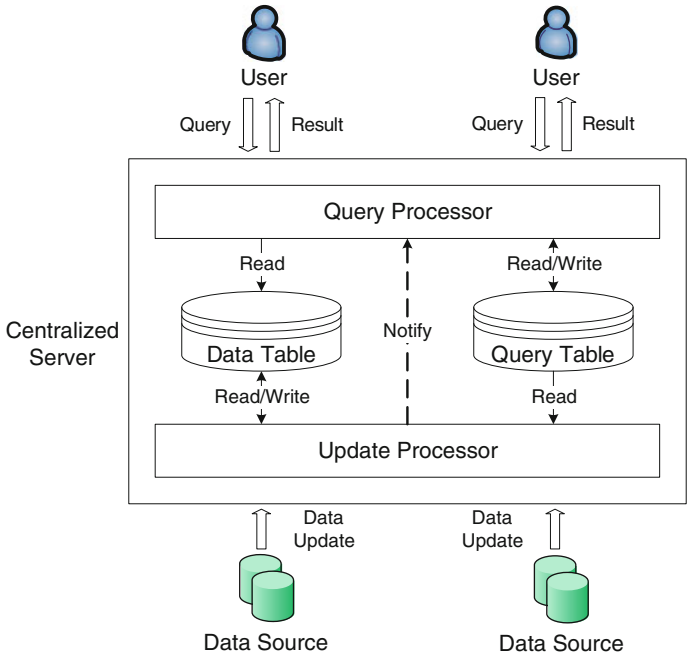



**Fig. 3** Centralized architecture

- *Algorithm INC-FKNMatchAD:* Algorithm INC-FKNMatchAD is an incremental version of algorithm FKNMatchAD. When necessary, the update processor will perform algorithm INC-FKNMatchAD for fast query re-evaluations.
- *A transmission reduction scheme:* We propose a transmission reduction scheme to eliminate the transmissions of unnecessary data updates among data sources and the server.

The details of procedure OnReceivingQuery and procedure OnReceivingUpdate are given in Sect. 3.2. Algorithm INC-FKNMatchAD is described in Sect. 3.3, while the proposed transmission reduction scheme is described in Sect. 3.5.

### 3.2 Processing continuous frequent $K$-$N$-match queries in centralized environments

Consider a query $\mathcal{Q}$ and a change of the $i$th dimension of a data point $P$. Such change may have one of the following affects.

(1) The result of $\mathcal{Q}$ is changed when the change is significant.
(2) The result $\mathcal{Q}$ is unchanged, but the corresponding internal data structures of $\mathcal{Q}$ are changed. This case usually occurs when the change is a little bit significant. Note that the internal data structures are used to facilitate safe region calculation, and the details of the internal data structures will be described in Sect. 3.4.
(3) The result and the internal data structures of $\mathcal{Q}$ are both unchanged when the change is minor.

According to the above observations, safe regions can be classified into two levels: *level-one safe regions* and *level-two safe regions*. Considering the $i$th dimension of a data point $P$ and a query $\mathcal{Q} = \langle Q, [n_0, n_1], k \rangle$, level-one safe regions and level-two safe regions are defined as follows.

**Definition 6** The *level-one safe region* of the $i$th dimension of data point $P$ with respect to query $\mathcal{Q}$ (denoted as $SR_1(\mathcal{Q}, P, i)$) is defined as the interval that if the new value of the $i$th dimension of $P$ is within $SR_1(\mathcal{Q}, P, i)$, (1) the result of query $\mathcal{Q}$ and (2) the corresponding internal data structures will be unchanged.

**Definition 7** The *level-two safe region* of the $i$th dimension of data point $P$ with respect to query $\mathcal{Q}$ (denoted as $SR_2(\mathcal{Q}, P, i)$) is defined as the interval that if the new value of the $i$th dimension of $P$ is within $SR_2(\mathcal{Q}, P, i)$, (1) the result of query $\mathcal{Q}$ will be unchanged and (2) the corresponding internal data structures will be changed.

By the above definitions, it is obvious that $SR_1(\mathcal{Q}, P, i)$ and $SR_2(\mathcal{Q}, P, i)$ do not intersect with each other.

When receiving a query $\mathcal{Q}$ issued by a user, the query processor will perform the following steps (procedure OnReceivingQuery).

- *Step 1:* Register query $\mathcal{Q}$ into the query table.
- *Step 2:* Execute algorithm FKNMatchAD to get the result of query $\mathcal{Q}$, and then report the result to the user.
- *Step 3:* Calculate the safe regions of query $\mathcal{Q}$.[4]
- *Step 4:* Store the safe regions and the data structures used in algorithm FKNMatchAD into the query table.

When the values of a data point $P$ have been changed, the corresponding data source will send a data update to the update processor to notify the server of the latest values of data point $P$. Once receiving a data update $\langle P, (p'_1, p'_2, \ldots, p'_d)\rangle$, the update processor will perform the following steps (procedure OnReceivingUpdate).

- *Step 1:* Retrieve the old values of data point $P$ (i.e., $(p_1, p_2, \ldots, p_d)$) from the data table.
- *Step 2:* For each dimension $i$ that $p'_i \neq p_i$, invoke the following three steps.
- *Step 3:* Update the value of dimension $i$ of $P$ to $p'_i$.
- *Step 4:* Keep the values of dimension $i$ of all data points sorted according to their distances to $q_i$.[5]
- *Step 5:* Notify the query processor about the change of dimension $i$.

When being notified by the update processor about the change of the $i$th dimension of data point $P$, the query processor will perform the following steps (procedure OnNotified).

- *Step 1:* For each query, say query $\mathcal{Q}$, in the query table, the query processor first retrieves the corresponding safe regions (i.e., $SR_1(\mathcal{Q}, P, i)$ and $SR_2(\mathcal{Q}, P, i)$) and the data structures from the query table, and then performs one of the following sub-steps according to the relationship among $p'_i$ and $SR_1(\mathcal{Q}, P, i)$ and $SR_2(\mathcal{Q}, P, i)$.
- *Step 2-1:* If $p'_i$ is not in $SR_1(\mathcal{Q}, P, i)$ or $SR_2(\mathcal{Q}, P, i)$, the query processor will perform algorithm INC-FKNMatchAD to re-evaluate query $\mathcal{Q}$, send the new result to the user issuing query $\mathcal{Q}$, and store the new result of query $\mathcal{Q}$ and the corresponding data structures into the query table.
- *Step 2-2:* If $p'_i$ is in $SR_2(\mathcal{Q}, P, i)$, the query processor will update the internal data structures of query $\mathcal{Q}$, and store the data structures into the query table. The details of data structure update are given in Sect. 3.4.
- *Step 2-3:* If $p'_i$ is in $SR_1(\mathcal{Q}, P, i)$, the query processor will do nothing since the result of the query is unchanged.

[4] The details of safe region calculation are described in Sect. 3.4.

[5] This step is to facilitate algorithm INC-FKNMatchAD.

**Table 4** Summary of safe regions

| Case | Condition | | | Safe regions | |
|---|---|---|---|---|---|
| | $is\ appear(\mathcal{Q}, P, i)$ | $appear[P]$ | $match(\mathcal{Q}, P, i)$ | $SR_1(\mathcal{Q}, P, i)$ | $SR_2(\mathcal{Q}, P, i)$ |
| I | FALSE | $<n_0 - 1$ | | Region 1 | Region 2 |
| II | FALSE | $\geq n_0$ | | Region 1 | $\emptyset$ |
| III | TRUE | $<n_0 - 1$ | | Region 2 | Region 1 |
| IV | TRUE | | $>n_1$ | Region 3 | Region 1 |

Region 1: $[-\infty, q_i - threshold(\mathcal{Q})) \cup (q_i + threshold(\mathcal{Q}), \infty]$
Region 2: $[q_i - threshold(\mathcal{Q}), q_i + threshold(\mathcal{Q})]$
Region 3: $[q_i - threshold(\mathcal{Q}), q_i - |p_i - q_i|] \cup [q_i + |p_i - q_i|, q_i + threshold(\mathcal{Q})]$

Finally, the algorithmic forms of the above procedures are as below.

**procedure** QueryProcessor.OnReceivingQuery($\mathcal{Q}$)
1: Register $\mathcal{Q}$ into the query table
2: Execute algorithm FKNMatchAD and send the result to the user issuing $\mathcal{Q}$
3: Store the data structures used in algorithm FKNMatchAD into the query table
4: **for each** data point, say $P$, in the data table **do**
5: **for each** dimension $i$ **do**
6: Calculate $SR_1(\mathcal{Q}, P, i)$ and $SR_2(\mathcal{Q}, P, i)$ according to Table 4
7: Store $SR_1(\mathcal{Q}, P, i)$ and $SR_2(\mathcal{Q}, P, i)$ into the query table

**procedure** UpdateProcessor.OnReceivingUpdate($\langle P, (p'_1, p'_2, \ldots, p'_d)\rangle$)
1: Retrieve the old values of $P$ (i.e., $p_1, p_2, \ldots, p_d$) from the data table
2: **for each** dimension $i$ **do**
3: **if** $p'_i \neq p_i$ **then**
4: Update the value of dimension $i$ of $P$ to $p'_i$
5: Keep the values of dimension $i$ of all data points sorted according to their distances to $q_i$
6: Notify the query processor by invoking QueryProcessor.OnNotified ($P, i, p'_i$)

**procedure** QueryProcessor.OnNotified($P, i, p'_i$)
1: **for each** query, say $\mathcal{Q}$, in the query table **do**
2: Retrieve $SR_1(\mathcal{Q}, P, i)$, $SR_2(\mathcal{Q}, P, i)$ and $\mathcal{Q}$'s data structures from the query table
3: **if** $p'_i$ is not in $SR_1(\mathcal{Q}, P, i)$ or $SR_2(\mathcal{Q}, P, i)$ **then** /* Step 2-1 */
4: Perform algorithm INC-FKNMatchAD to get the new result of $\mathcal{Q}$
5: Send the new result of $\mathcal{Q}$ to the user issuing $\mathcal{Q}$
6: Store the new result of $\mathcal{Q}$ and the corresponding data structures into the query table
7: **else if** $p'_i$ is in $SR_2(\mathcal{Q}, P, i)$ **then** /* Step 2-2 */
8: Perform procedure UpdateDataStructure($\mathcal{Q}, P, i$) to update the internal data structures of query $\mathcal{Q}$
9: Store the data structures into the query table

### 3.3 Incremental query re-evaluation

We observe that when processing a frequent $k$-$n$-match query, algorithm FKNMatchAD has to sort all data items according to their distances to $q_i$, for each dimension $i$. Let $c$ be the number of data points in the database. The sorting is of time complexity $O(d \times c \log c)$ and is a time-consuming part in algorithm FKNMatchAD. Such observation motivates us to develop algorithm INC-FKNMatchAD (an incremental version of algorithm FKNMatchAD) to facilitate fast query re-evaluations. The idea of algorithm INC-FKNMatchAD is as follows. In the first time to process $\mathcal{Q}$, algorithm FKNMatchAD is performed to get the result of $\mathcal{Q}$. In addition, the sorted orders of all data points in each dimension are stored. When the value of dimension $i$ of data point $P$ changes from $p_i$ to $p_i'$, the position of data point $P$ in dimension $i$ may become incorrect and the query processor has to adjust the position of data point $P$ in dimension $i$ to make the order of all data points become sorted. Such adjustment is of time complexity $O(\log c)$ when the values of dimension $i$ of all data points are organized by a balanced binary search tree (e.g., an AVL tree or a red-black tree). Thus, with the above adjustment method, query processor can perform algorithm INC-FKNMatchAD without sorting any data points in any dimension when re-evaluating query $\mathcal{Q}$.

### 3.4 Safe region calculation and data structure update

To facilitate the calculation of safe regions, for each query $\mathcal{Q}$, the following data structures are stored in the query table.

(1) $isAppear(\mathcal{Q}, P, i)$: $isAppear(\mathcal{Q}, P, i)$ indicates whether the $i$th dimension of point $P$ contributes one match to query $\mathcal{Q}$. In other words, $isAppear(\mathcal{Q}, P, i)$ indicates whether the $i$th dimension of point $P$ makes $appear[P]$ increase by one. $isAppear(\mathcal{Q}, P, i)$ is initially set to FALSE and will be set to TRUE when the triple $(P, i, dif)$ is popped from $g[\,]$.

(2) $match(\mathcal{Q}, P, i)$: $match(\mathcal{Q}, P, i)$ is set to $m$ if the $i$-dimension of point $P$ contributes the $m$th match of query $\mathcal{Q}$. In algorithm FKNMatchAD, data point $P$ will be inserted into $S_n$ when there exists one dimension $i$ where $match(\mathcal{Q}, P, i) = n$.

(3) $threshold(\mathcal{Q})$: $threshold$ is the $dif$ of the latest triple popped from $g[\,]$. Thus, the distance between query point $Q$ and each point in $S_{n_0}, \ldots, S_{n_1}$ is not larger than $threshold(\mathcal{Q})$.

Because the result of query $\mathcal{Q}$ is dependent on the set $S[\,]=\{S_{n_0}\ldots S_{n_1}\}$, the result of query $\mathcal{Q}$ is unchanged as long as query $\mathcal{Q}$'s $S[\,]$ is unchanged. Consider a query $\mathcal{Q}$ and a data update that the $i$th dimension of data point $P$ changes from $p_i$ to $p_i'$. The data update will not affect $S[\,]$ if the data update does not change the value of $appear[P]$. Thus, $SR_1(\mathcal{Q}, P, i)$ *is set to the interval that if* $p_i'$ *is in the interval, the data update will not affect the result of* $\mathcal{Q}$ *and* $appear[P]$. It is possible that the data update changes $appear[P]$ but does not change $S[\,]$. Since being an important parameter in determining $S[\,]$, the value of $appear[P]$ should be kept up-to-date. Thus, $SR_2(\mathcal{Q}, P, i)$ *is set to the interval that if* $p_i'$ *is in the interval, the data update will not affect the result of* $\mathcal{Q}$ *but will affect* $appear[P]$.

To derive safe regions, we performed experiments on the Intel Lab Data[6] (the real dataset used in this paper) and observed the influence of each value change. Thus, we induced the following four cases with frequent occurrences. Note that users can derive other new cases as well as the corresponding safe regions, and it is easy to integrate these new safe regions into scheme CFKNMatchAD-C.

[6] The description of the dataset is given in http://db.csail.mit.edu/labdata/labdata.html.

**Lemma 1** (Safe Regions for Case I) *In the cases that* $isAppear(\mathcal{Q}, P, i)$ *is FALSE and* $appear[P]$ *is smaller than* $n_0 - 1$, $SR_1(\mathcal{Q}, P, i)$ *is* $[-\infty, q_i - threshold(\mathcal{Q})) \cup (q_i + threshold(\mathcal{Q}), \infty]$ *and* $SR_2(\mathcal{Q}, P, i)$ *is* $[q_i - threshold(\mathcal{Q}), q_i + threshold(\mathcal{Q})]$.

*Proof* $appear[P] < n_0 - 1$ means that $P$ is not in the result set of query $\mathcal{Q}$ while $isAppear(\mathcal{Q}, P, i)$=FALSE means that $p_i$ does not make $appear[P]$ increase. Thus, $p_i'$ will either increase $appear[P]$ by one or not affect $appear[P]$. Since being smaller than $n_0-1$, $appear[P]$ will be still smaller than $n_0$ no matter what the value of $p_i'$ is. Therefore, the change of the $i$th dimension of data point $P$ will not make $P$ become a member of the result set of query $\mathcal{Q}$. However, $appear[P]$ will increase by one only when the distance between $p_i'$ and $q_i$ (i.e., $|p_i' - q_i|$) is smaller than or equal to $threshold(\mathcal{Q})$. The internal data structures will not be affected if the distance between $p_i'$ and $q_i$ (i.e., $|p_i' - q_i|$) is larger than $threshold(\mathcal{Q})$. Hence, $SR_1(\mathcal{Q}, P, i)$ is $[-\infty, q_i - threshold(\mathcal{Q})) \cup (q_i + threshold(\mathcal{Q}), \infty]$. Since $p_i'$ will not affect the result of $\mathcal{Q}$, $SR_2(\mathcal{Q}, P, i)$ is $[-\infty, \infty] - SR_1(\mathcal{Q}, P, i) = [q_i - threshold(\mathcal{Q}), q_i + threshold(\mathcal{Q})]$. □

**Lemma 2** (Safe Regions for Case II) *In the cases that* $isAppear(\mathcal{Q}, P, i)$ *is FALSE and* $appear[P]$ *is larger than or equal to* $n_0$, $SR_1(\mathcal{Q}, P, i)$ *is* $[-\infty, q_i - threshold(\mathcal{Q})) \cup (q_i + threshold(\mathcal{Q}), \infty]$ *and* $SR_2(\mathcal{Q}, P, i)$ *is* $\emptyset$.

*Proof* $appear[P] \geq n_0$ means that $P$ is in $S[\ ]$ of query $\mathcal{Q}$, while $isAppear(\mathcal{Q}, P, i) =$ FALSE means that $p_i$ does not increase $appear[P]$. Thus, $p_i'$ will either increase $appear[P]$ by one or not affect $appear[P]$. Since the distance between $p_i$ and $q_i$ is larger than $threshold(\mathcal{Q})$, the result of $\mathcal{Q}$ will be unchanged as long as the distance between $p_i'$ and $q_i$ is still larger than $threshold(\mathcal{Q})$. Thus, we have $SR_1(\mathcal{Q}, P, i) \cup SR_2(\mathcal{Q}, P, i) = [-\infty, q_i - threshold(\mathcal{Q})) \cup (q_i + threshold(\mathcal{Q}), \infty]$. In addition, when the distance between $p_i'$ and $q_i$ is still larger than $threshold(\mathcal{Q})$, $appear[P]$ will be also unchanged. Thus, $SR_1(\mathcal{Q}, P, i)$ is $[-\infty, q_i - threshold(\mathcal{Q})) \cup (q_i + threshold(\mathcal{Q}), \infty]$ and $SR_2(\mathcal{Q}, P, i) = SR_1(\mathcal{Q}, P, i) \cup SR_2(\mathcal{Q}, P, i) - SR_1(\mathcal{Q}, P, i) = \emptyset$. □

**Lemma 3** (Safe Regions for Case III) *In the cases that* $isAppear(\mathcal{Q}, P, i)$ *is TRUE and* $appear[P]$ *is smaller than* $n_0$, $SR_1(\mathcal{Q}, P, i)$ *is* $[q_i - threshold(\mathcal{Q}), q_i + threshold(\mathcal{Q})]$ *and* $SR_2(\mathcal{Q}, P, i)$ *is* $[-\infty, q_i - treshold(\mathcal{Q})) \cup (q_i + threshold(\mathcal{Q}), \infty]$.

*Proof* $appear[P] < n_0$ means that $P$ is not in the result set of query $\mathcal{Q}$ while $isAppear(\mathcal{Q}, P, i)$=TRUE means that $p_i$ has made $appear[P]$ increase. Thus, the new value $p_i'$ will either decrease $appear[P]$ by one or not affect $appear[P]$. Since being smaller than $n_0$, $appear[P]$ will be still smaller than $n_0$ no matter what the value of $p_i'$ is. Therefore, the change of the $i$th dimension of data point $P$ will not make $P$ become a member of the result set of query $\mathcal{Q}$. However, $appear[P]$ will decrease by one if the distance between $p_i'$ and $q_i$ (i.e., $|p_i' - q_i|$) becomes larger than $threshold(\mathcal{Q})$. $appear[P]$ will not be affected as long as the distance between $p_i'$ and $q_i$ (i.e., $|p_i' - q_i|$) is smaller than or equal to $threshold(\mathcal{Q})$. Thus, $SR_1(\mathcal{Q}, P, i)$ is $[q_i - treshold(\mathcal{Q}), q_i + threshold(\mathcal{Q})]$. Since $p_i'$ will not affect the result of $\mathcal{Q}$, $SR_2(\mathcal{Q}, P, i)$ is $[-\infty, \infty] - SR_1(\mathcal{Q}, P, i) = [-\infty, q_i - treshold(\mathcal{Q})) \cup (q_i + threshold(\mathcal{Q}), \infty]$. □

**Lemma 4** (Safe Regions for Case IV) *In the cases that* $isAppear(\mathcal{Q}, P, i)$ *is TRUE, and* $match(\mathcal{Q}, P, i)$ *is larger than* $n_1$, $SR_1(\mathcal{Q}, P, i)$ *is* $[q_i - threshold(\mathcal{Q}), q_i - |p_i - q_i|] \cup [q_i + |p_i - q_i|, q_i + threshold(\mathcal{Q})]$ *and* $SR_2(\mathcal{Q}, P, i)$ *is* $[-\infty, q_i - threshold(\mathcal{Q})) \cup (q_i + threshold(\mathcal{Q}), \infty]$.

*Proof* $match(\mathcal{Q}, P, i) > n_1$ implies that $appear[P] > n_1$ and $P$ is in $S[\,]$. $isAppear(\mathcal{Q}, P, i)$=TRUE means that $p_i$ has made $appear[P]$ increase. If the distance between $p'_i$ and $q_i$ is larger than or equal to the distance between $p_i$ and $q_i$, $p'_i$ will not affect another dimension $j$ where $match(\mathcal{Q}, P, j) \leq n_1$. Since the appearances of $P$ in $S[\,]$ are affected by each dimension $j$, where $match(\mathcal{Q}, P, j) \leq n_1$, the result of $\mathcal{Q}$ will not be affected by $p'_i$ if $p'_i$ is in $[-\infty, q_i - |p_i - q_i|] \cup [q_i + |p_i - q_i|, \infty]$. On the other hand, $isAppear(\mathcal{Q}, P, i)$ will become FALSE and $appear[P]$ will decrease by one if the distance between $p'_i$ and $q_i$ becomes larger than threshold. Thus, $isAppear(\mathcal{Q}, P, i)$ and $appear[P]$ will be unchanged if $p'_i$ is in $[q_i - threshold(\mathcal{Q}), q_i + threshold(\mathcal{Q})]$. As a result, $SR_1(\mathcal{Q}, P, i)$ is $([-\infty, q_i - |p_i - q_i|] \cup [q_i + |p_i - q_i|, \infty]) \cap ([q_i - threshold(\mathcal{Q}), q_i + threshold(\mathcal{Q})]) = [q_i - threshold(\mathcal{Q}), q_i - |p_i - q_i|] \cup [q_i + |p_i - q_i|, q_i + threshold(\mathcal{Q})]$ and $SR_2(\mathcal{Q}, P, i)$ is $[-\infty, q_i - |p_i - q_i|] \cup [q_i + |p_i - q_i|, \infty] - SR_1(\mathcal{Q}, P, i) = [-\infty, q_i - threshold(\mathcal{Q})) \cup (q_i + threshold(\mathcal{Q}), \infty]$. □

For better readability, the safe regions of these four cases are summarized in Table 4. In addition, procedure UpdateDataStructure can be designed according to these lemmas, and the algorithmic form of procedure UpdateDataStructure is as below.

**Procedure** UpdateDataStructure($\mathcal{Q}$, $P$, $i$)

```
1: if isAppear(Q, P, i) is FALSE then
2:   if appear[P] < n0 − 1 then      /* Case I */
3:     appear[P] + +
4:     isAppear(Q, P, i) ←TRUE
5: else      /* isAppear(Q, P, i) is TRUE */
6:   if (appear[P] < n0) or match(Q, P, i) > n1 then      /* Case III or Case IV */
7:     appear[P] − −
8:     isAppear(Q, P, i) ←FALSE
```

### 3.5 Unnecessary data update elimination

If the centralized server is dedicated to process continuous frequent $k$-$n$-match queries, it is not necessary to keep the values of the data points in the data table up-to-date. If a data update will not change the results and the corresponding internal data structures of all queries in the query table, the data update is called *unnecessary* and the data source need not send the data update to the server. Therefore, the network traffic among data sources and the centralized server can be reduced. In view of the above observation, level-one safe regions can be used as filters to eliminate transmissions of unnecessary data updates among data sources and the server.

The idea of unnecessary data update elimination is as follows. After receiving a continuous frequent $k$-$n$-match query, procedure QueryProcessor.OnReceivingQuery is performed to compute the safe regions of the query for each dimension of each data point. Then, the centralized server sends the level-one safe region of each data point to the data source which owns or monitors the data point. Therefore, every data source can use the received level-one safe regions to determine whether a data update will change the results of queries. The data source will send a data update to the centralized server only when the new value of the data point is out of its level-one safe region.

When receiving a data update from a data source, the centralized server invokes procedure UpdateProcessor.OnReceivingUpdate to process the data update. When the new value of the data point is out of its level-two safe region, some query re-evaluations may be

**Table 5** An illustrative example

| ID | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ |
|---|---|---|---|---|---|
| 1 | 0.5 | 1.8 | 4.0 | 1.5 | 3.1 |
| 2 | 2.5 | 6.2 | 6.5 | 9.0 | 6.2 |
| 3 | 3.3 | 4.4 | 7.2 | 7.0 | 4.3 |
| 4 | 6.0 | 5.2 | 3.0 | 4.5 | 2.3 |
| 5 | 4.8 | 9.0 | 10.0 | 11.0 | 8.5 |
| Q | 3.0 | 4.5 | 5.5 | 6.0 | 3.5 |

performed. Since not keeping the latest values of all data points, when re-evaluating queries, the centralized server has to send probe messages to all data sources to ask them to report the latest values of the data points they own or monitor. In addition, after re-evaluating one query, the centralized server will send the new level-one safe regions to the data sources. Finally, the correctness of algorithm CFKMatchAD-C can be proved by the following theorem.

**Theorem 1** (Correctness of scheme CFKMatchAD-C) *Scheme CFKNMatchAD-C returns the frequent k-n-match set of the query $\mathcal{Q}$.*

*Proof* With the definitions of safe regions and Lemmas 1–4, it is obvious that this theorem is correct. □

### 3.6 Complexity analysis

In scheme CFKNMatchAD-C, for each query, only the first query evaluation is processed by algorithm FKNMatchAD and the successive query re-evaluations are processed by algorithm INC-FKNMatchAD. Suppose that scheme CFKNMatchAD is used to process an arbitrary query and there are $n_{update}$ data updates. Let $pr$ be the probability that a data update changes the query results. Thus, algorithm FKNMatchAD is executed once, while algorithm INC-FKNMatchAD has to be executed for $pr \times n_{update}$ times. According to the analysis in Sect. 3.3, the time complexities of algorithm FKNMatchAD and algorithm INC-FKNMatchAD to process a data update for a query are $O(d \times c \log c)$ and $O(\log c)$, respectively. Consider the case that algorithm INC-CFKNMatchAD is used on a query to process $n_{update}$ data updates. Since only $pr \times n_{update}$ data updates are necessary, the amortized time complexity of scheme CFKNMatchAD-C to process a data update is $O\left(\frac{d}{n_{update}} \times c \log c + pr \times \log c\right)$. When there are $|Q|$ queries registered, the amortized time complexity of scheme CFKNMatchAD-C to process a data update is $O\left(\frac{d}{n_{update}} \times c \log c \times |Q| + pr \times \log c \times |Q|\right)$.

The major memory consumption is in the table storing the values of all data points, the three auxiliary tables for each registered query (described in Sect. 3.4), and two tables storing safe regions of all registered queries. The space complexity of the table storing the values of all data points is $O(c \times d)$. The space complexity of each auxiliary table is $O(|Q| \times c \times d)$. Similarly, the space complexity of each table storing safe regions is also $O(|Q| \times c \times d)$. Thus, the space complexity of scheme CFKNMatchAD-C is $O(|Q| \times c \times d)$.

In this subsection, we use the database shown in Table 5 as an example to illustrate the behavior of scheme CFKNMatchAD-C. Suppose that a user issues a $k$-$n$-match query

**Table 6** Content of the data structures after performing scheme CFKNMatchAD-C

| *appear*[] | {3,3,5,4,1} |
|---|---|
| $S_3$ | {3,4,2,1} |
| $S_4$ | {3,4} |
| $S$ | {3,4} |
| *threshold* | 2.5 |

(a)

*isAppear*($\mathcal{Q}$,$P$,$i$)

| | | $i$ | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| $P$ | 1 | T | F | T | F | T |
| | 2 | T | T | T | F | F |
| | 3 | T | T | T | T | T |
| | 4 | F | T | T | T | T |
| | 5 | T | F | F | F | F |

T: TRUE F: FALSE

(b)

*match*($\mathcal{Q}$,$P$,$i$)

| | | $i$ | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| $P$ | 1 | 3 | 0 | 2 | 0 | 1 |
| | 2 | 1 | 3 | 2 | 0 | 0 |
| | 3 | 2 | 1 | 5 | 4 | 3 |
| | 4 | 0 | 1 | 4 | 3 | 2 |
| | 5 | 1 | 0 | 0 | 0 | 0 |

(c)

Case

| | | $i$ | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| $P$ | 1 | | II | | II | |
| | 2 | | | | II | II |
| | 3 | | | IV | | |
| | 4 | II | | | | |
| | 5 | III | I | I | I | I |

(d)

$SR_1(\mathcal{Q},P,i)$

| | | $i$ | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| $P$ | 1 | | $[-\infty,2)\cup(7,\infty]$ | | $[-\infty,3.5)\cup(8.5,\infty]$ | |
| | 2 | | | | $[-\infty,3.5)\cup(8.5,\infty]$ | $[-\infty,1)\cup(6,\infty]$ |
| | 3 | | | $[3,3.8]\cup[7.2,8]$ | | |
| | 4 | $[-\infty,0.5]\cup[5.5,\infty]$ | | | | |
| | 5 | $[0.5,5.5]$ | $[-\infty,2)\cup(7,\infty]$ | $[-\infty,3)\cup(8,\infty]$ | $[-\infty,3.5)\cup(8.5,\infty]$ | $[-\infty,1)\cup(6,\infty]$ |

(e)

$SR_2(\mathcal{Q},P,i)$

| | | $i$ | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| $P$ | 1 | | | | | |
| | 2 | | | | | |
| | 3 | | | $[-\infty,3)\cup(8,\infty]$ | | |
| | 4 | | | | | |
| | 5 | $[-\infty,0.5)\cup(5.5,\infty]$ | $[2,7]$ | $[3,8]$ | $[3.5,8.5]$ | $[1,6]$ |

(f)

$\mathcal{Q} = \langle Q(3.0, 4.5, 5.5, 6.0, 3.5), [3,4], 2\rangle$ to a server. Once receiving the query, the query processor performs procedure QueryProcessor. OnReceivingQuery to register the query in the query table, performs algorithm FKNMatchAD, and returns the result of $\mathcal{Q}$ to the user. The data structures used by scheme CFKNMatchAD-C are shown in Table 6. Suppose that the update processor receives the following two data updates sequentially. The update processor performs procedure UpdateProcessor.OnReceivingUpdate to handle each data update, and the behavior of update processor is as follows.

- *Update 1:* The value of point 5 changes from (4.8, 9, 10, 11, 8.5) to (7.8, 10, 10, 11, 6.3). Since three dimensions are changed in this update, the update processor will first decompose the update into three sub-updates (each sub-update represents the change of one dimension), and notify the query processor once for each sub-update.

- *Sub-update 1-1:* The value of the first dimension of point 5 is changed from 4.8 to 7.8.
  The new value of the first dimension of point 5 is in $SR_2(\mathcal{Q}, 5, 1)$ and matches Case III, according to Lemma 3, $appear[5]$ decreases by one and becomes 0. In addition, $isAppear(5, 1)$ is set to FALSE and the result of $\mathcal{Q}$ is unchanged.
- *Sub-update 1-2:* The value of the second dimension of point 5 is changed from 9 to 10.
  Since the new value of the second dimension of point 5 is in $SR_1(\mathcal{Q}, 5, 2)$, $isAppear(5, 2)$, $appear[5]$, and the result of $\mathcal{Q}$ are unchanged.
- *Sub-update 1-3:* The value of the 5th dimension of point 5 is changed from 8.5 to 6.3.
  Since the 5th dimension of point 5 is in $SR_1(\mathcal{Q}, 5, 5)$, $appear[5]$, $isAppear(5, 1)$, and the result of $\mathcal{Q}$ are unchanged.

- *Update 2:* The value of point 2 is changed from (2.5, 6.2, 6.5, 9, 6.2) to (2.5, 6.2, 6.5, 7.2, 6.2).
  Only the fourth dimension is changed, and therefore, the update processor will notify the query processor about the change of the fourth dimension. Since the new value of the fourth dimension is out of $SR_1(\mathcal{Q}, 2, 4)$ and $SR_2(\mathcal{Q}, 2, 4)$, the result of $\mathcal{Q}$ may change. Thus, algorithm INC-FKNMatchAD is performed to obtain the result of $\mathcal{Q}$ and the new values of all internal data structures.

## 4 Scheme CFKNMatchAD-D: continuous frequent $K$-$N$-match query processing in decentralized environments

### 4.1 The decentralized system architecture

Since the centralized server needs to process all queries, the centralized architecture does not scale well as the numbers of queries and data points increase. In this subsection, we consider de-centralized environments which employ multiple servers to process users' queries. Consider a de-centralized environment with $m + 1$ servers (one master server and $m$ slave servers $N_1, N_2, \ldots, N_m$) in Fig. 4. Suppose that the database consists of $c$ data points, $P_1, P_2, \ldots, P_c$. Without loss of generality, we assume that $c$ is divisible by $m$. Then, each slave server $N_i$ ($i = 1, 2, \ldots, m$) handles $\frac{c}{m}$ data points: $P_{(i-1)\frac{c}{m}+1}, P_{(i-1)\frac{c}{m}+2}, \ldots P_{i\times\frac{c}{m}}$.

### 4.2 Processing continuous frequent $K$-$N$-match queries in de-centralized environments

In this subsection, we propose scheme CFKNMatchAD-D to process continuous frequent $k$-$n$-match queries in de-centralized environments. The idea of scheme CFKNMatchAD-D is as follows. When a client registers a continuous frequent $k$-$n$-match query $\mathcal{Q} = \langle Q, [n_0, n_1], k\rangle$ to the master server $N_0$, $N_0$ registers query $\mathcal{Q}$ into its query table and forwards query $\mathcal{Q}$ to all slave servers. Then, each slave server, say $N_i$, performs scheme CFKNMatchAD-C to get its $S[\,]$[7] (called the *local* $S[\,]$) of query $\mathcal{Q}$ on data points $P_{(i-1)\frac{c}{m}+1}, P_{(i-1)\frac{c}{m}+2}, \ldots P_{i\times\frac{c}{m}}$ and sends the local $S[\,]$ to the master server $N_0$. In addition, $N_i$ also sends the safe regions of the data points $P_{(i-1)\frac{c}{m}+1}, P_{(i-1)\frac{c}{m}+2}, \ldots P_{i\times\frac{c}{m}}$ to the corresponding data sources to eliminate transmissions of unnecessary data updates. After receiving the local $S[\,]$s of all

[7] $S[\,]$ is an internal data structure used by scheme CFKNMatchAD-C. Interested readers can refer to Sect. 2.2 for details.

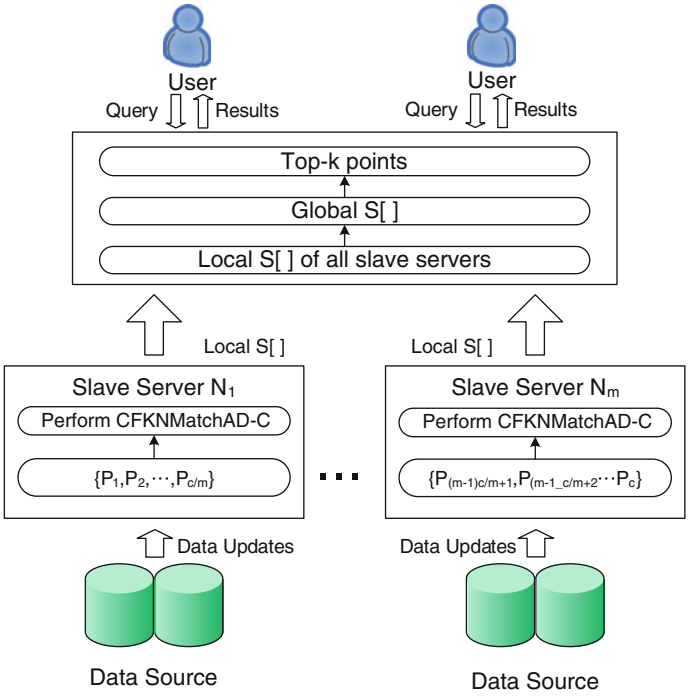



**Fig. 4** Decentralized architecture

slave servers, the master server first constructs its $S[\,]$ (called *global* $S[\,]$) by setting the $i$th element in the global $S[\,]$ to be the union of the $i$th element of each slave server's local $S[\,]$, where $i = n_0, n_0 + 1, \ldots, n_1$. Then, the master server scans the global $S[\,]$ to count the number of appearances of each data point and picks the top $k$ points that appear most times in the global $S[\,]$ as the result of query $\mathcal{Q}$.

When a slave server $N_i$ receives a data update indicating that the value of a data point $P$ gets out of its level-one safe region, $N_i$ first checks whether the new value of $P$ will change $N_i$'s local $S[\,]$ (i.e., checks whether the value of $P$ is out of its level-two safe regions). If yes, $N_i$ performs scheme CFKNMatchAD-C to get the new local $S[\,]$ and safe regions and sends the new local $S[\,]$ to the master server to update the master server's global $S[\,]$. $N_i$ also send the new safe regions to the data points monitored by $N_i$. Then, the master server determines the new result of query $\mathcal{Q}$ according to the global $S[\,]$. Otherwise, $N_i$ performs procedure UpdateDataStructures to update its internal data structures.

Finally, we prove the following lemma and theorem to show that for each slave server, sending its local $S[\,]$ to the master server is enough for master server to get the result of query $\mathcal{Q}$.

**Lemma 5** *Considering a continuous frequent k-n-match query $\mathcal{Q}$, a data point that does not appear in the local $S[\,]$ of the corresponding slave server will not appear in result of query $\mathcal{Q}$.*

*Proof* We prove this lemma by contradiction. Assume that there exists a data point $P$ which is in the result of query $\mathcal{Q}$ and is not in the local $S[\,]$ of the corresponding slave server. Since $P$ is not in the local $S[\,]$ of the slave server which $P$ is belonging to, we know that the

slave server owns/monitors at least $k$ data points having $n$-match differences smaller than $P$. However, $P$ in the result of query $\mathcal{Q}$ means that there are less than $k$ data points having $n$-match differences smaller than $P$. Such contradiction shows that no such $P$ exists and hence proves this lemma. □

**Theorem 2** (Correctness of Scheme CFKNMatchAD-D) *Scheme CKNMatchAD-D returns the frequent k-n-match set of the query $\mathcal{Q}$.*

*Proof* Since the data points owned or monitored by all slave servers are disjoint, according to Lemma 5, all data points in $S[\,]$ of scheme CFKNMatchAD-C will be also in the global $S[\,]$ of scheme CFKNMatchAD-D. Thus, scheme CFKMatchAD-D gets the same result as scheme CFKNMatchAD-C does. □

### 4.3 Complexity analysis

It is obvious that in scheme CFKNMatchAD-D, the behavior of each slave server is similar to executing scheme CFKNMatchAD-C with $\frac{c}{m}$ data points. Let $pr$ be the probability that a data update changes the query results. Thus, under the case that $|Q|$ queries are registered, the amortized time and space complexities of each slave server are $O\left(\frac{d}{n_{update}} \times \frac{c}{m} \times \log \frac{c}{m} \times |Q| + pr \times \log \frac{c}{m} \times |Q|\right)$ and $O\left(\frac{c}{m} \times d \times |Q|\right)$, respectively.

We now consider the space and time complexities of the master server. To obtain the query results, the master should store the local $S[\,]$ of all slave servers. The local $S[\,]$ of each slave server consists of $n_1 - n_0 + 1$ lists (i.e., $S_{n_0}, S_{n_0+1}, \ldots, S_{n_1}$). In practice, only the top-$k$ elements in $S_i, i = n_0, n_0 + 1, \ldots, n_1$, are used in calculating the query results. Thus, each slave server only needs to send the top-$k$ elements of $S_i, i = n_0, n_0+1, \ldots, n_1$, to the master server, and the master server will have enough information to obtain the query results. Thus, each list stored in the master server consists of at most $k$ elements. In addition, the master server should also store the global $S[\,]$. Thus, the space complexity of the master server is $O((n_1 - n_0 + 1) \times k \times (m + 1) \times |Q|) = O((n_1 - n_0) \times k \times m \times |Q|)$. For each data update that may affect the query results, the master server has to scan the local $S[\,]$ of all slave servers and the global $S[\,]$ to obtain the query results. Therefore, the amortized time complexity of the master server to process a data update when $|Q|$ queries are registered is $O(pr \times (n_1 - n_0) \times k \times m \times |Q|)$.

## 5 Performance evaluation

In this section, we evaluate the performance of scheme CFKNMatchAD-C and scheme CFKNMatchAD-D. Scheme FKNMatchAD, which executes algorithm FKNMatchAD for each data update, is also implemented for comparison purposes. The *average response time* of queries and the *amount of produced packets* are used as the performance metrics. We first apply these schemes on a real dataset in Sect. 5.1. However, since many parameters such as the number of data points and the number of dimensions are small and fixed in the real dataset, only the experimental result with different $k$ is given. To measure the effect of these parameters on large-scale environments, we also apply these schemes on a synthetic dataset and the experimental results are given in Sect. 5.2. For each experiment, one random generated query is executed for each run and six runs are performed. The average results of the six runs are taken as the experiment results.

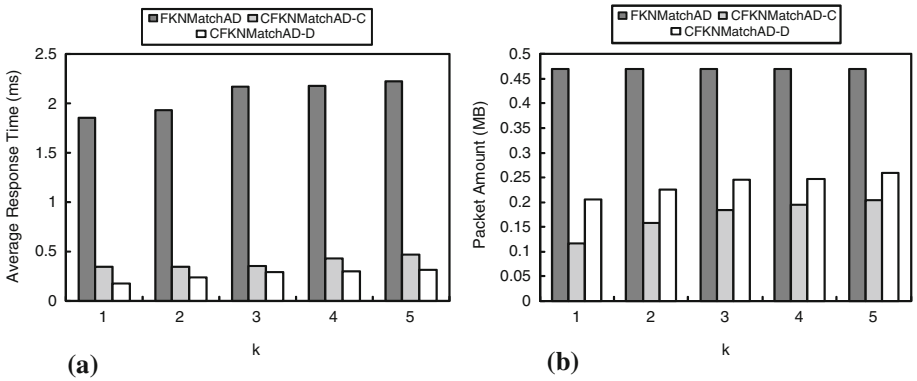



**Fig. 5** Experimental results on real dataset. **a** Average response time, **b** Amount of produced packets

### 5.1 Experiments on the real dataset

In this subsection, we use the Intel Lab Data[8] to measure the performance of these schemes. Intel Lab Data are the data collected from 54 sensors deployed in the Intel Berkeley Research Lab. The dataset consists of about 2.3 million records and each record consists of three readings: temperature, humidity, and light. Thus, the dataset consists of 54 three-dimensional dynamic data. In this experiment, one master server and two slave servers are used in scheme CFKNMatchAD-D.

Figure 5 shows the effect of $k$ on average response time and the amount of produced packets. The values of $n_0$ and $n_1$ are set to 2 and 3, respectively. As shown in Fig. 5a, scheme CFKNMatchAD-C is able to reduce average response time by utilizing safe regions to avoid unnecessary query re-evaluations. When $k$ becomes large, the value of $threshold$ also becomes large, thereby shortening the lengths of safe regions. Thus, the average response time of schemes utilizing safe regions increases as $k$ increases. It is interesting that although scheme CFKNMatchAD-D uses more servers than scheme CFKNMatchAD-C, the improvement of scheme CFKNMatchAD-D over scheme CFKNMatchAD-C in average response time is not significant. The reason is that when the number of data points is small (only 54 data points in the dataset), the improvement gained from the decentralized architecture does not significantly exceed the overhead caused by the decentralized architecture. We will show that scheme CFKNMatchAD-D is able to greatly reduce average response time in large-scale environments in the following experiments. Similarly, since increasing $k$ will reduce the effect of safe regions, as shown in Fig. 5b, the amounts of packets produced by schemes using safe regions to eliminate transmissions of unnecessary data updates (e.g., scheme CFKNMatchAD-C and scheme CFKNMatchAD-D) increase as $k$ increases. However, it is interesting that scheme CFKNMatchAD-D produces more packets than scheme CFKNMatchAD-C. It is also due to the reason that the number of data points in the dataset is small. We will show in Sect. 5.2 that scheme CFKNMatchAD-D is able to reduce much more traffic than scheme CFKNMatchAD-C in large-scale environments.

[8] The description of the dataset is given in http://db.csail.mit.edu/labdata/labdata.html.

**Table 7** System parameters

| Parameter | Default setting |
|---|---|
| Number of points ($N$) | 20,000 |
| Number of dimensions | 40 |
| $k$ | 30 |
| Monitor interval ($[n_0,n_1]$) | [4,32] |
| Number of fluctuated dimensions ($m$) | 8 |
| Inter-arrival time of data updates | Exponential dist. with mean = 10 s |
| Simulation time | 3,000 s |

## 5.2 Experiments on the synthetic dataset

### *5.2.1 Simulation model*

The synthetic dataset consists of $N$ data points with $d$ attributes. We assume that each attribute is normalized and is distributed uniformly in [0, 1]. In the centralized environment, there is one centralized server receiving data updates of $N$ data points from the data sources. In the de-centralized environment, there are 10 slave servers and one master server. Each slave server monitors $\frac{N}{10}$ data points. The inter-arrival time among data updates is modeled as an exponential distribution with mean 10 seconds. In every data update, at most $m$ attributes of a randomly selected data point are changed. After receiving a data update, each scheme is performed to update query results. To investigate the impact of various parameters, we conduct several experiments with different data variation rates. The data variation rate is defined as the upper bound of $\frac{|p_i'-p_i|}{p_i}$ for each data update of an arbitrary data point $P$. Thus, higher data variation rate indicates that the changes of values are more significant. The data variation rate is set to 5, 10, and 15%, respectively. Since the results on different data variation rates are similar, only results on 10% are given in the following subsections. The number of dimensions is set to 40. According to [30], the values of $n_0$ and $n_1$ are set to 4 and 32, respectively. The simulation time of each run is set to 3,000 s. Table 7 lists the default setting of the system parameters.

### *5.2.2 Impact of the number of data points*

In this subsection, we investigate the impact of the number of data points. Figure 6a shows the average response time of all schemes as the number of data points increases from 10,000 to 30,000. It is obvious that the average response time of all schemes increases as the number of data points increases. Since the average response time of scheme FKNMatchAD is much longer than that of other schemes, the average response time of scheme CFKNMatchAD-C and scheme CFKNMatchAD-D are given in Fig. 6b for better readability.

For better understanding of the behavior of all schemes, the numbers of executions of algorithm FKNMatchAD and algorithm INC-FKNMatchAD in the experiment with 20,000 data points are given in Table 8. There are 1,377 data updates in this experiment. Since each data update triggers one execution of algorithm FKNMatchAD, including initial execution

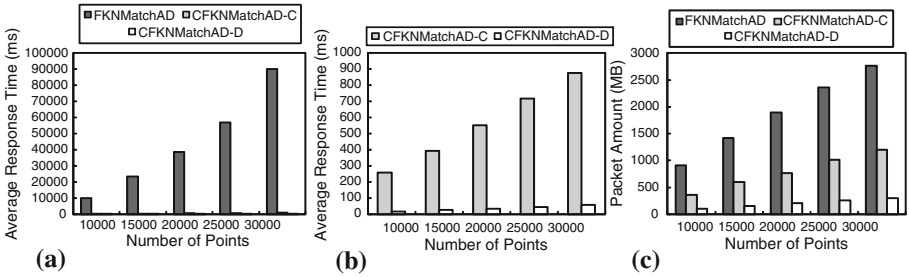



**Fig. 6** Experimental results with different number of points. **a** Average response time (I). **b** Average response time (II). **c** Amount of produced packets

**Table 8** Numbers of executions of algorithms

| Scheme | No. of executions of algorithms | |
|---|---|---|
| | FKNMatchAD | INC-FKNMatchAD |
| FKNMatchAD | 1,378 | 0 |
| CFKNMatchAD-C | 1 | 832 |
| CFKNMatchAD-D | 10 | 908 |

of algorithm FKNMatchAD, scheme FKNMatchAD executes algorithm FKNMatchAD for 1378 times. In scheme CFKNMatchAD-C, with the aid of safe regions, there are 545 unnecessary data updates not triggering any execution of algorithm FKNMatchAD and algorithm INC-FKNMatchAD. In addition, algorithm FKNMatchAD is executed only once (when the query processor receives the query) while algorithm INC-FKNMatchAD is executed for 832 times. Since algorithm INC-FKNMatchAD is of lower time complexity than algorithm FKNMatchAD, scheme CFKNMatchAD-C is much faster than scheme FKNMatchAD. Scheme CFKNMatchAD-D executes algorithm FKNMatchAD for 10 times since each slave server should execute algorithm FKNMatchAD once when the query processor receives the query. These executions of algorithm FKNMatchAD in scheme CFKNMatchAD-D are performed on fewer data points because each slave server is in charge of a portion of data points. Thus, the execution of algorithm FKNMatchAD in scheme CFKNMatchAD-D is faster than the execution in centralized schemes. Similarly, the executions of algorithm INC-FKNMatchAD in scheme CFKNMatchAD-D is also faster than the executions in centralized schemes. Thus, the average response time of scheme CFKNMatchAD-D is much shorter than that of other schemes.

Figure 6c shows the amounts of packets produced by all schemes. It is intuitive that the amounts of packets increase as the number of data points increases. Scheme CFKNMatchAD-C produces less packets than scheme FKNMatchAD does using safe regions to eliminate unnecessary data updates not affecting the results of queries. In addition, scheme CFKNMatchAD-D produces less amount of packets than scheme CFKMatchAD-C does. We use the following example to explain such result. Consider the case that a data point $p$ is getting out of its safe regions and suppose that in scheme CFKNMatchAD-D, $p$ is monitored by slave server $N_i$. In scheme CFKNMatchAD-C, the centralized server has to ask *all* data points to submit their latest values to get the latest query results and safe regions. In addition, the centralized server has to send the latest safe regions to all data points. On the contrary, in scheme CFKNMatchAD-D, only the data points monitored by $N_i$ will be requested to

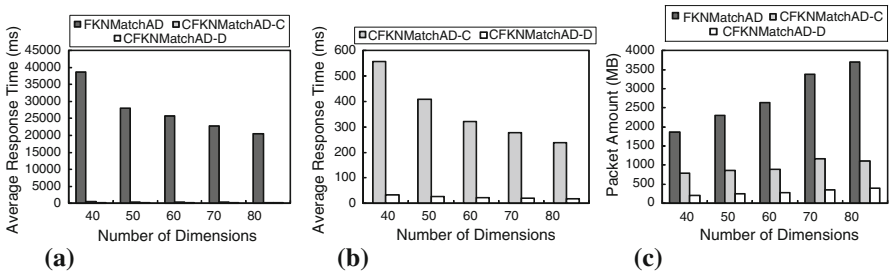



**Fig. 7** Experimental results with different number of dimensions. **a** Average response time (I). **b** Average response time (II). **c** Amount of produced packets

submit their latest data values, and only $N_i$ has to send the latest safe regions to the data points monitored by $N_i$. Due to the above reasons, scheme CFKNMatchAD-D is able to produce less packets than scheme CFKNMatchAD-C does.

### *5.2.3 Impact of the number of dimensions*

This subsection shows the effect of the number of dimensions which is set from 40 to 80. The average response time is shown in Fig. 7a, b while the amounts of produced packets are shown in Fig. 7c. Since the monitor interval $[n_0, n_1]$ is fixed, when the number of dimensions becomes large, a data point has more likelihood to appear in $g[\,]$ more times. Thus, a smaller portion of data points will be processed and the value of $threshold$ will become smaller. In each query re-evaluation, for each dimension, only the data points, whose distances to the corresponding attribute of the query point are shorter than $threshold$, will be reorganized. As a result, the average response time of all schemes decreases when number of dimensions becomes large.

As shown in Fig. 7c, the amount of packets produced by scheme CFKNMatchAD-C increases as the number of dimensions increases. It is because that when the number of dimensions is large, the size of each data update also increases. Moreover, since scheme CFKNMatchAD-C and scheme CFKNMatchAD-D use safe regions to eliminate unnecessary data updates, the increase in the number of dimensions does not have significant effect on the amounts of packets produced by scheme CFKNMatchAD-C and scheme CFKNMatchAD-D.

### *5.2.4 Impact of the number of fluctuant dimensions*

Figure 8 shows the impact of the number of fluctuant dimensions by setting the number of fluctuant dimensions from 4 to 16. As shown in Fig. 8a, the average response time of scheme FKNMatchAD is not significantly affected by the increase in the number of fluctuant dimensions because scheme FKNMatchAD re-evaluates queries for each data update. On the other hand, as shown in Fig. 8b, the average response time of schemes using safe regions increases as the number of fluctuant dimensions increases. The reason is that when the number of fluctuant dimensions increases, the probability that a data point gets out of its safe region increases, thereby increasing the number query re-evaluations and average response time.

Figure 8c shows the effect of the number of fluctuant dimensions on the amounts of produced packets. We can see that the increase in the number of fluctuant dimensions does not significantly affect the amounts of packets produced by scheme FKNMatchAD and scheme CFKNMatchAD-C. It is because that in scheme FKNMatchAD and scheme

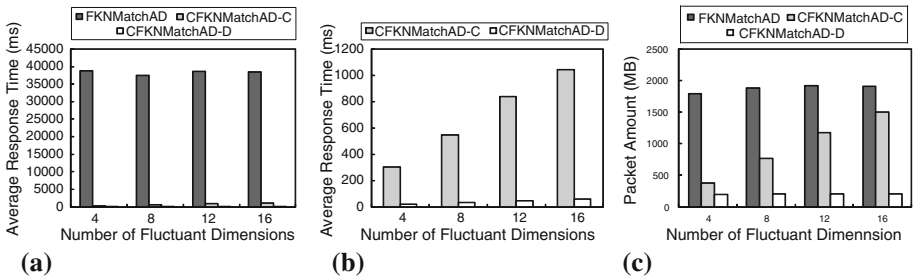



**Fig. 8** Simulation with different number of fluctuated dimension. **a** Average response time (I). **b** Average response time (II). **c** Amount of produced packets

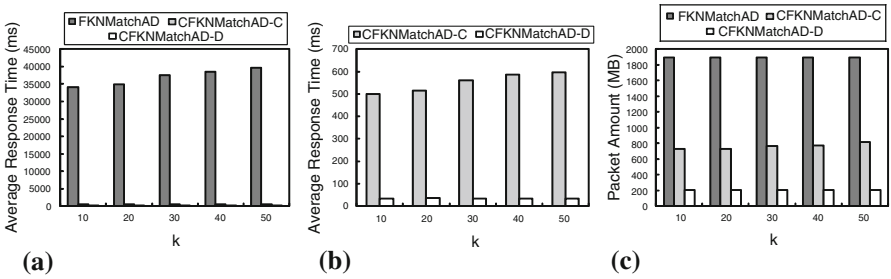



**Fig. 9** Simulation with different $k$. **a** Average response time (I). **b** Average response time (II). **c** Amount of produced packets

CFKNMatchAD-C, the data sources send all data updates to the server. Since increasing the number of fluctuant dimensions increases the probability that a data point gets out of its safe regions, the number of unnecessary data updates decreases. Thus, the amount of packets produced by scheme CFKNMatchAD-C increases as the number of fluctuant dimensions increases. Since in scheme CFKNMatchAD-D, each slave server manages only a portion of data points, the lengths of safe regions of scheme CFKNMatchAD-D are longer than those of scheme CFKNMatchAD-C. Thus, the influence of increasing the number of fluctuant dimensions in scheme CFKNMatchAD-D is quite insignificant.

### *5.2.5 Impact of the value of K*

Figure 9 shows the impact of the value of $k$ by setting $k$ from 10 to 50. It is intuitive that increasing the value of $k$ will increase the average response time since the loop (lines 5–12) in algorithm FKNMatchAD will iterate more runs. Thus, as shown in Fig. 9a, b, the average response time of scheme FKNMatchAD and scheme CFKNMatchAD-C increases as $k$ increases. In scheme CFKNMatchAD-D, since each slave server handles only a portion of data points, the effect of $k$ is quite slight. Figure 9c shows the effect of $k$ on the amounts of produced packets. As observed, the increase in $k$ does not significantly affect the amounts of packets produced by all schemes in large-scale environments.

### *5.2.6 Impact of monitor interval* $[n_0, n_1]$

In this experiment, we evaluate the performance of all schemes under different monitor intervals $[n_0, n_1]$. This experiment consists of two sub-experiments. In the first experiment, the

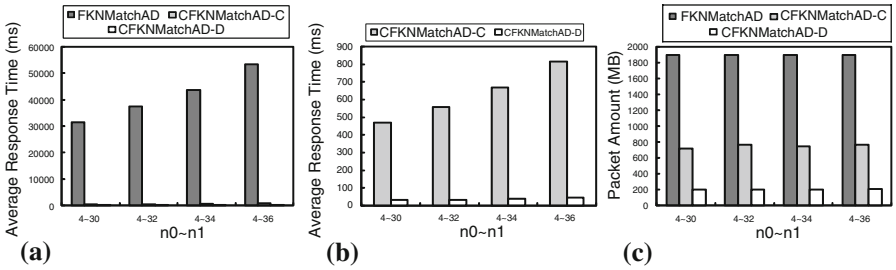



**Fig. 10** Simulation with different $n_1$. **a** Average response time (I). **b** Average response time (II). **c** Amount of produced packets

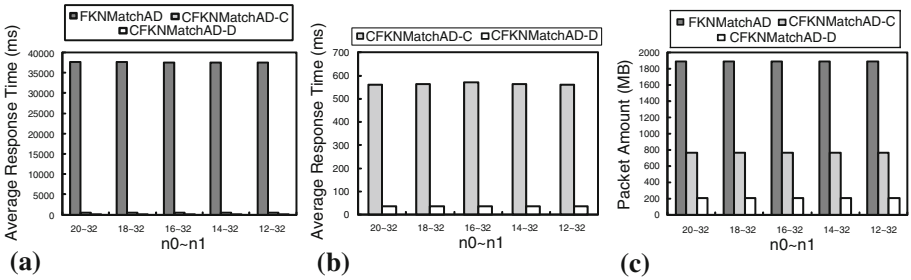



**Fig. 11** Simulation with different $n_0$. **a** Average response time (I). **b** Average response time (II). **c** Amount of produced packets

value of $n_0$ is set to four while the value of $n_1$ increases from 30 to 36. In the second sub-experiment, the value of $n_0$ increases from 12 to 20 while the value of $n_1$ is set to 32. The experimental results of the first and the second sub-experiments are shown in Figs. 10 and 11, respectively.

Figure 10a and b show the average response time of all schemes in the first sub-experiment. When $n_1$ becomes larger, the loop (lines 5–12) in algorithm FKNMatchAD will iterate more runs, thereby making the response time of each query evaluation longer. As a result, average response time of each scheme becomes longer as $n_1$ increases. Moreover, in addition to using safe regions, scheme CFKNMatchAD-D also employs multiple servers to process queries, and thus, the influence of increasing $n_1$ on scheme CFKNMatchAD-D is slighter than that on scheme CFKNMatchAD-C. Consider the average response time of all schemes in the second sub-experiment which is shown in Fig. 11a, b. Since the behavior of the loop (lines 5–12) in algorithm FKNMatchAD is not affected by $n_0$, the change of $n_0$ does not explicitly affect the average response time of all schemes. We can also observe from Figs. 10c and 11c that the increase in $n_0$ and $n_1$ does not significantly affect the amounts of packets produced by all schemes.

### *5.2.7 Impact of the number of slave servers*

This subsection shows the impact of the number of slave servers which is set from 2 to 12. Since only scheme CFKNMatchAD-D is of slave servers, the performance of other schemes is not affected by the number of slave servers. It is intuitive that utilizing more slave servers will shorten the average response time, and such intuition is shown in Fig. 12b. In addition, it is not surprising that the marginal improvement decreases as the number of slave servers

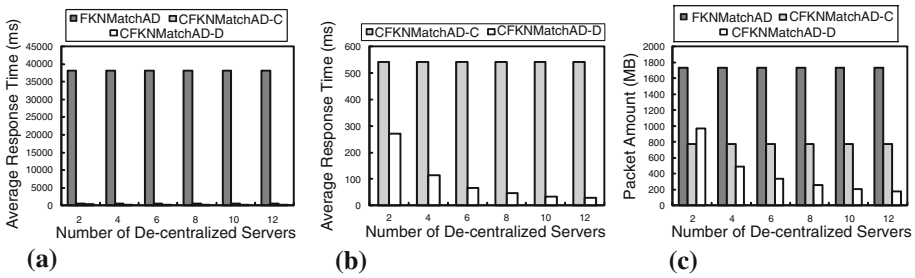



**Fig. 12** Simulation with different number of servers. **a** Average response time (I). **b** Average response time (II). **c** Amount of packets

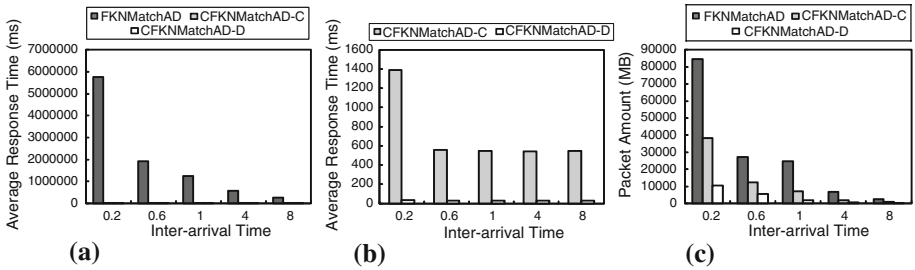



**Fig. 13** Simulation with different mean of inter-arrival time. **a** Average response time (I). **b** Average response time (II). **c** Amount of packets

increases. The reason is that in scheme CFKNMatchAD-D, only local $S[\ ]$ can be computed distributedly in each slave server, and the result of the query should be computed centrally in the master server. Thus, employing more slave servers can only speed up the computation of local $S[\ ]$ and cannot reduce the computation load of the master server. It is obvious that utilizing more servers will unavoidably increase communication traffic among servers. On the other hand, utilizing more servers will make each server handle less data points, make safe regions more effective, and thus reduce the number of necessary data updates. As shown in Fig. 12c, the reduction in data updates is enough to pay off the extra communication traffic only when the number of servers is large enough (larger than or equal to four in this experiment).

### *5.2.8 Impact of the inter-arrival time of data updates*

This experiment investigates the impact of inter-arrival time of data updates on all schemes, and the experimental results are shown in Fig. 13. It is obvious that the smaller the inter-arrival time is, the more data updates arrive. Thus, as shown in Fig. 13c, the packet amounts increase as the mean of inter-arrival time decreases. Since safe regions are used in scheme CFKMatchAD-C and scheme CFKMatchAD-D to eliminate the transmissions of unnecessary data updates, the amounts of packets produced by scheme CFKMatchAD-C and scheme CFKMatchAD-D only slightly increase as the mean of inter-arrival time decreases.

Obviously, decreasing the inter-arrival time of data updates will increase the number of query re-evaluations. When the inter-arrival time becomes shorter to some extent, some data updates may be queued and the average response time will increase. As shown in Fig. 13a, the average response time of scheme FKNMatchAD increases significantly when the

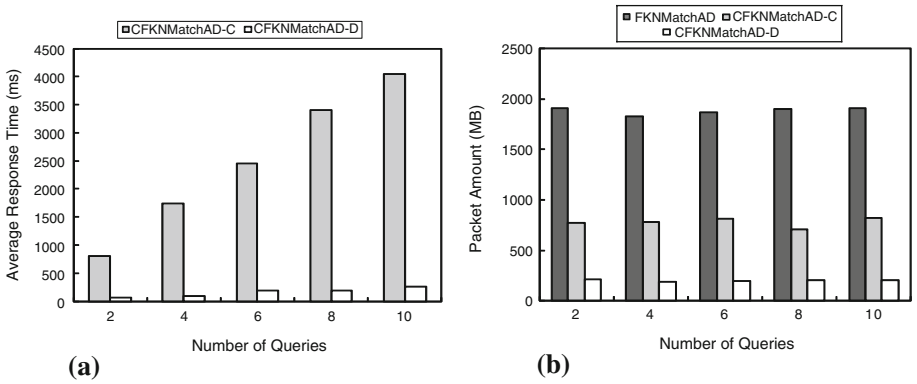



**Fig. 14** Simulation with different number of queries. **a** Average response time. **b** Amount of packets

inter-arrival time decreases from 8 to 0.2 s. Due to the effect of algorithm INC-FKNMatchAD, the queueing time in scheme CFKNMatchAD-C becomes significant only when the inter-arrival time becomes shorter than 0.6 s. When the inter-arrival time is larger than or equal to 0.6 s, the queueing time is quite insignificant. Thus, as shown in Fig. 13b, the average response time of scheme CFKNMatchAD-C is almost unchanged when the inter-arrival time is from 8 seconds to 0.6 s, and increases significantly when the inter-arrival time becomes 0.2 s. As to scheme CFKNMatchAD-D, due to using multiple servers, the queueing time is insignificant even when the inter-arrival time is set to 0.2 s, showing the scalability of scheme CFKNMatchAD-D.

#### *5.2.9 Impact of the number of registered queries*

This experiment investigates the impact of the number of registered queries. It is intuitive that the server(s) will consume more time to process queries when there are more registered queries. When the number of registered queries is larger than or equal to two, the rate that the centralized server running scheme FKNMathAD finishes all registered queries is smaller than the arrival rate of data updates. The average response time of scheme FKNMatchAD will drastically increase as the simulation time increases, and thus, we do not show the average response time of scheme FKNMatchAD and only show the average response time of scheme CFKNMatchAD-C and scheme CFKNMatchAD-D in Fig. 14a. We can see that the average response time of scheme CFKNMatchAD-C and scheme CFKNMatchAD-D is in proportion to the number of registered queries. Since the number of registered queries does not affect the number of data updates, as shown in Fig. 14b, there is no significant influence of the number of registered queries on the amount of produced packets for all schemes.

## 6 Related work

In the last decade, several indexing structures such as R-tree [9], R+-tree[26], R*-tree[4], SS-tree [33] were proposed to index data objects in order to facilitate NN/kNN query processing. Unfortunately, as mentioned in [32], all R-tree-like indexing structures suffer from the dimensionality curse and are not suitable for high-dimensional data. Thus, some indexing structures were proposed to index high-dimensional data [1,5,13,14,20,32]. Weber et al. proposed in [32] a vector approximation scheme, called VA-file, to efficiently process NN queries on high-dimensional data. The VA-file divides the data space into $2^b$ rectangles where

$b$ is a user-specified number of bits. Each cell is assigned a unique bit-string of length $b$, and each data point is approximated by the bit-string of the cell where the data point falls. NN queries are performed by scanning the VA-file. During scanning, a large amount data can be skipped based on the approximations. Therefore, the VA-file is able to speed up NN query processing by reducing the amount of data that must be read from disks. Aggarwal et al. argued in [1] that a more flexible similarity function is needed for high-dimensional data and proposed a similarity function which is more meaningful than Euclidean distance. With the new similarity function, Aggarwal et al. then proposed IGrid-index, which is based on the inverted index on the grid representation of the data, to speed up similarity query processing. In [17], Koudas et al. introduced the ekNN ($e$-approximate kNN) problem which finds the approximate kNN answers and the error is bounded by $e$. Koudas et al. then proposed a scheme called DISC to optimize memory utilization to process ekNN queries. Jagadish et al. proposed a B+tree-based indexing method called iDistance [13] for kNN search on high-dimensional data. In iDistance, data are first divided into several partitions and each partition is assigned a reference point. With simple mapping effort, the distance of each data point to the reference point of its partition is able to be indexed by a B+-tree. In addition, Jagadish et al. also proposed an algorithm to process kNN queries on iDistance.

Similar to frequent $k$-$n$-match search, in subspace similarity search, not all dimensions are used in measuring similarity between a query point and a data point. The difference between subspace similarity search and frequent $k$-$n$-match search is as follows. For a subspace similarity query, the user should specify a subset of dimensions which the user is interested in, and the similarity between the query point and each data object should consider only these user-specified dimensions. On the other hand, for a frequent $k$-$n$ match query, the user only specifies the preferred number of dimensions (i..e, $n$), and the similarity between the query point and the data object should consider the $n$ most similar dimensions. In [18], Kriegel et al. first addressed this problem using the partial VA-file as an adaptation of the VA-file. The original VA-file is split into $N$ partial VA-files, where $N$ is the dimensionality. Each partial VA-file stores the approximation of the original full-dimensional VA-file in that dimension. Subspace similarity queries are processed by scanning only the file corresponding to the selected dimensions. In [19], Lian et al. proposed a multi-pivot-based method which first preprocesses the data objects to select appropriate pivots to maximize the effect of pruning and then processes the subspace similarity queries according to these pivots. In [6], Bernecker et al. proposed the projected R-tree to index the data objects and designed a best-first based algorithm to process kNN queries in arbitrary subspaces with the aid of the projected R-tree.

Continuous kNN search was proposed for dynamic data such as moving objects. In [29], kNN queries are performed periodically at predefined sample points. This algorithm has a trade-off between the sampling rate and the computation cost. If the sampling rate is set too low, the answer is of high likelihood to be incorrect. Otherwise, setting the sampling rate too high will result in much computational overhead. Moreover, it has no accuracy guarantee because the sampling rate cannot always match the split points perfectly. Prabhakar et al. proposed in [23] an indexing method called Q-index to index continuous spatial queries. Each moving object is assigned a safe region so that the results of the continuous queries will not change as long as the position of each object is within its safe region. The unnecessary query re-revaluations can be avoided with the aid of safe regions. Hu et al. proposed in [12] a generic framework to utilize safe regions to monitor continuous spatial queries over moving objects. Different from [23], the proposed framework also takes the costs of location updates and location probes in to consideration. Nutanong et al. proposed in [22] the V*-Diagram and the associated algorithm to process moving kNN queries. V*-Diagram utilizes the safe regions of both data objects and queries to reduce I/O and computation costs. Continuous

kNN search can be viewed as a special case of top-$k$ monitoring. Babcock et al. addressed in [3] the problem of processing top-$k$ monitoring queries over distributed data streams. Babcock et al. first derived two invariants and then proposed a three-phase algorithm for distributed top-$k$ monitoring. Mouratidis et al. addressed in [21] the problem of monitoring top-$k$ queries on a fixed window of the most recent data. Different from [3], the study of [21] dealt with ordinary top-$k$ queries over a single multi-dimensional stream. In [7], Deng et al. addressed the same problem that [3] dealt with and proposed algorithm MR to reduce the number of messages.

The idea of safe region is also used in query processing over sensor networks. Silberstein et al. proposed in [28] a query processing algorithm to monitor extreme values (MIN or MAX) of sensor networks. After the first execution of an extreme value monitoring query, a threshold-based filter is sent to each sensor and each sensor will update its reading to the root sensor only when its reading is out of its filter or it receives a query message from the root. Xu et al. [34] proposed an algorithm called FILA to use the similar concept to process top-$k$ monitoring queries on wireless sensor networks. In addition to filter setup and update procedures, a query re-evaluation procedure utilizing filters is also proposed in [34]. Both algorithms are able to significantly reduce energy consumption of sensors by reducing unnecessary sensor updates. Yeo et al. argued in [35] that FILA consumes much energy on sensor reading probing and filter updating especially when the variation rate of top-$k$ results is high. A data priority assignment algorithm was proposed to determine the priorities of all sensors. An priority-based algorithm was then proposed to process top-$k$ queries in an energy-efficient manner.

## 7 Conclusions

In this paper, we addressed the problem of processing continuous frequent $k$-$n$-match queries over dynamic data. We first devised four formulae to calculate safe regions, and then proposed scheme CFKNMatchAD-C to utilize safe regions to avoid unnecessary query re-evaluations. Algorithm INC-FKNMatchAD was also proposed to facilitate fast query re-evaluations. To reduce the amount of data transmissions, scheme CFKNMatchAD-C also uses safe regions to eliminate transmissions of unnecessary data updates. Moreover, for applications in large-scale environments, we further proposed scheme CFKNMatchAD-D by extending scheme CFKMatchAD-C to employ multiple servers to speed up query processing. Experimental results showed that scheme CFKNMatchAD-C and scheme CFKNMatchAD-D are able to greatly reduce average response time by utilizing safe regions. Besides, the use of safe regions in scheme CFKNMatchAD-C and scheme CFKNMatchAD-D is able to reduce network traffic among the server(s) and the data sources. Scheme CFKNMatchAD-D is of the fastest average response time due to the use of multiple servers to process continuous frequent $k$-$n$-match queries.

**Acknowledgments** This work was supported in part by Industrial Technology Research Institute of Taiwan, ROC, under contract A301AR2210, and by National Science Council of Taiwan, ROC, under contracts 99-2221-E-009-140-MY2 and 99-2219-E-002-029.

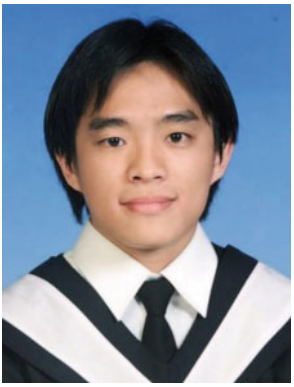
## References

1. Aggarwal CC, Yu PS (August 2000) The IGrid index: reversing the dimensionality curse for similarity indexing in high dimensional space. In: Proceedings of the ACM international conference on knowledge discovery and data mining, pp 119–129

2. Agrawal R, Lin K-I, Sawhney HS, Shim K (1995) Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In: Proceedings of the international conference on very large data bases, pp 490–501
3. Babcock B, Olston C (June 2003) Distributed Top-K monitoring. In: Proceedings of the ACM international conference on management of data
4. Beckmann N, Kriegel H-P, Schneider R, Seeger B (1990) The R*-tree: an efficient and robust access method for points and rectangles. In: Proceedings of the ACM international conference on management of data, pp 322–331
5. Berchtold S, Keim DA, Kriegel H-P (1996) The X-Tree: an index structure for high-dimensional data. In: Proceedings of the international conference on very large data bases, pp 28–39
6. Bernecker T, Emrich T, Graf F, Kriegel H-P, Króger P, Renz M, Schubert E, Zimek A (2010) Subspace similarity search: efficient k-NN queries in arbitrary subspaces. In: Proceedings of the international conference on scientific and statistical database management
7. Deng B, Jia Y, Yang S (June 2006) Supporting efficient distributed Top-k monitoring. In: Proceedings of the 7th international conference on Web-Age information management
8. Gao L, Yao Z, Wang X (2002) Evaluating continuous nearest neighbor queries for streaming time series via pre-fetching. In: Proceedings of the ACM international conference on information and knowledge management, pp 485–492
9. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of the ACM international conference on management of data, pp 47–57
10. Hammouda KM, Kamel MS (2004) Document similarity using a phrase indexing graph model. Knowl Inf Syst 6(6):710–727
11. Hjaltason GR, Samet H (1998) Distance browsing in spatial databases. ACM Trans Database Syst 24(2):265–318
12. Hu H, Xu J, Lee DL (2005) A generic framework for monitoring continuous spatial queries over moving objects. In: Proceedings of the ACM international conference on management of data, pp 479–490
13. Jagadish HV, Ooi BC, Tan K-L, Yu C, Zhang R (2005) iDistance: an adaptive B+-tree based indexing method for nearest neighbor search. ACM Trans Database Syst 30(2):364–397
14. Katayama N, Satoh S (1997) The SR-tree: an index structure for high-dimensional nearest neighbor queries. In: Proceedings of the ACM international conference on management of data, pp 369–380
15. Kelil A, Wang S, Jiang Q, Brzezinski R (2010) A general measure of similarity for categorical sequences. Knowl Inf Syst 24(2):197–220
16. Korn F, Sidiropoulos N, Faloutsos C, Siegel E, Protopapas Z (1996) Fast nearest neighbor serach in medical image databases. In: Proceedings of the international conference on very large data bases, pp 215–226
17. Koudas N, Ooi BC, Tan K-L, Zhang R (2004) Approximate NN queries on streams with guaranteed error/performance bounds. In: Proceegings of the 30th internatvonal confeyence on very large data bases
18. Kriegel H-P, Kroger P, Schubert M, Zhu Z (July 2006) Efficient query processing in arbitrary subspaces using vector approximations. In: Proceedings of the international conference on scientific and statistical database management
19. Lian X, Chen L (April 2008) Similarity search in arbitrary subspaces under Lp-Norm. In: Proceedings of the IEEE international conference on data engineering
20. Lin K-I, Jagadish HV, Faloutsos C (1994) The TV-tree: an index structure for high-dimensional data. VLDB J 3(4)
21. Mouratidis K, Bakiras S, Papadias D (June 2006) Continuous monitoring of Top-k queries over sliding windows. In: Proceedings of 25th ACM international conference on management of data
22. Nutanong S, Zhang R, Tanin E, Kulik L (2008) The V*Diagram: a querydependent approach to moving KNN queries. In: Proceedings of the 34th internatvonal conference on very large data bases
23. Prabhakar S, Xia Y, Kalashnikov D, Aref WG, Hambrusch S (2002) Query indexing and velocity constrained indexing: scalable techniques for continuous queries on moving objects. IEEE Trans Comput 55
24. Quan X, Liu G, Lu Z, Ni X, Wenyin L (2010) Short text similarity based on probabilistic topics. Knowl Inf Syst 25(3):473–491
25. Seidl T, Kriegel H-P (1998) Optimal multi-step K-nearest neighbor search. In: Proceedings of the ACM international conference on management of data, pp 154–165
26. Sellis T, Roussopoulos N, Faloutsos C (1987) The R+-tree: a dynamic index for multi-dimensional objects. In: Proceedings of the international conference on very large data bases, pp 507–518
27. Shah S, Dharmarajan S, Ramamritham K (2003) An efficient and resilient approach to filtering and disseminating streaming data. In: Proceedings of the international conference on very large data bases

28. Silberstein A, Munagala K, Yang J (2006) Energy efficient monitoring of extreme values in sensor networks. In: Proceedings of the 26th ACM international conference on management of data, pp 169–180
29. Song Z, Roussopoulos N (2001) K-nearest neighbor search for moving query point. In: Proceedings of the international symposium on advances in spatial and temporal databases, pp 79–96
30. Tung AKH, Zhang R, Koudas N, Ooi BC (2006) Similarity search: a matching based approach. In: Proceedings of the international conference on very large data bases, pp 631–642
31. Wang C, Zhou BB, Zomaya AY (2009) A decentralized method for scaling up genome similarity search services. IEEE Trans Parallel Distribut Syst 20(3):303–315
32. Weber R, Schek H-J, Blott S (1998) A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proceedings of the international conference on very large data bases, pp 194–205
33. White DA, Jain R (1996) Similarity indexing with the SS-tree. In: Proceedings of the IEEE international conference on data engineering, pp 516–523
34. Xu J, Tang X, Lee W-C, Wu M (2007) Top-k monitoring in wireless sensor networks. IEEE Trans Knowl Data Eng 19(7):962–976
35. Yeo MH, Seong DO, Yoo JS (October 2008) PRIM: priority-based Top-k monitoring in wireless sensor networks. In: Proceedings of international symposium on computer science and its applications, pp 326–331
36. Zhang M, Alhajj R (2010) Effectiveness of NAQ-tree as index structure for similarity search in high-dimensional metric space. Knowl Inf Syst 22(1):1–26
37. Zhou Y, Ooi BC, Tan K-L (2008) Disseminating streaming data in a dynamic environment: an adaptive and cost-based approach. VLDB J 17

## Author Biographies



**Shih-Chuan Chiu** received the B.S. degree in Computer Science and Information Engineering from Tamkang University and the M.S. degree in Computer Science from National Chengchi University, in 2002 and 2004, respectively. He is currently working toward the Ph.D. degree in the Department of Computer Science at National Chiao Tung University. His current research interests include data mining, computer music and mobile computing.

**Jiun-Long Huang** received his B.S. and M.S. degrees in Computer Science and Information Engineering Department in National Chiao Tung University in 1997 and 1999, respectively, and his Ph.D. degree in Electrical Engineering Department in National Taiwan University in 2003. Currently, he is an assistant professor in Computer Science Department in National Chiao Tung University. His research interests include: mobile computing, wireless networks and data mining.

**Jen-He Huang** received the B.S. and M.S. degrees in Department of Computer Science in National Chiao Tung University, Taiwan, in 2006 and 2008, respectively. His research interests include sensor networks and wireless networks.