World Wide Web (2011) 14:465–494
DOI 10.1007/s11280-011-0125-5

Distributed processing of continuous sliding-window k-NN queries for data stream filtering

Krešimir Pripužić · Ivana Podnar Žarko · Karl Aberer

Received: 15 May 2010 / Revised: 15 November 2010 /
Accepted: 25 February 2011 / Published online: 24 March 2011

Abstract A sliding-window k-NN query (*k-NN/w query*) continuously monitors incoming data stream objects within a sliding window to identify *k* closest objects to a query. It enables effective filtering of data objects streaming in at high rates from potentially distributed sources, and offers means to control the rate of object insertions into result streams. Therefore k-NN/w processing systems may be regarded as one of the prospective solutions for the information overload problem in applications that require processing of structured data in real-time, such as the Sensor Web. Existing k-NN/w processing systems are mainly centralized and cannot cope with multiple data streams, where data sources are scattered over the Internet. In this paper, we propose a solution for distributed continuous k-NN/w processing of structured data from distributed streams. We define a k-NN/w processing model for such setting, and design a distributed k-NN/w processing system on top of the Content-Addressable Network (CAN) overlay. An extensive evaluation using both real and synthetic data sets demonstrates the feasibility of the proposed solution because it balances the load among the peers, while the messaging overhead within the P2P network remains reasonable. Moreover, our results clearly show the solution is scalable for an increasing number of queries and peers.

Keywords k nearest neighbor queries · sliding windows · data streams · peer-to-peer system

K. Pripužić (✉) · I. Podnar Žarko
Faculty of Electrical Engineering and Computing, University of Zagreb,
Unska 3, 10000 Zagreb, Croatia
e-mail: kresimir.pripuzic@fer.hr

I. Podnar Žarko
e-mail: ivana.podnar@fer.hr

K. Aberer
School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne,
EPFL IC IIF LSIR, Station 14, 1015 Lausanne, Switzerland
e-mail: karl.aberer@epfl.ch

1 Introduction

Nowadays, there is an emerging need for data stream filtering systems that are able to process huge amounts of continuously generated data stemming from heterogeneous sources dispersed over the Web. Such data has to be processed and delivered to end users in real-time, while data filtering is a crucial processing step to prevent information overload. The requirements of data filtering services have been identified even 15 years ago [3]: Such systems need to be effective in supplying users with useful information in a timely fashion, and need to handle large throughput of data publications for a large number of queries. The listed requirements pose a serious problem for emerging Internet-scale systems such as the Sensor Web [2]. The vision of a Sensor Web interconnecting millions of sensors placed in the environment to collect data from its surroundings is becoming a reality because a number of relevant projects are underway, such as sensor-based monitoring of air and water quality for pollution warning. The ongoing standardization efforts, e.g., the OGC's Sensor Web Enablement initiative [8] is designing standard models and schema for encoding sensor measurements along with a suite of protocols to handle sensor data in real-time. In particular, it specifies protocols for requesting, filtering, and retrieving sensor measurements, and for subscribing to specific sensor alerts. Such protocols need efficient techniques for processing sensor data in real-time: We put forward continuous processing of top-k queries over data streams as a viable technique for effective filtering of streaming objects in environments such as the Sensor Web.

In this paper, we study a particular type of continuous top-k queries which monitor k-nearest neighbor data objects over sliding windows (*k-NN/w queries*) to find the k nearest objects to a given query point continuously over time. Such queries are encountered in, for example, Geographic Information Systems (GIS) to find k-NN objects with respect to a given point in space while continuously monitoring incoming objects. Sliding windows are used in application domains that regard recent data objects as more important than older ones, and restrict the temporal scope of query processing in the absence of explicit deletions of data objects [20]. They are commonly defined as either the number of most recent data stream objects (count-based windows), or time intervals (time-based windows). The parameter k defines the number of matching data objects by restricting it to k nearest objects within a sliding window of size w, and therefore controls the rate of data object insertions into the output data stream. Note that since only k nearest objects within the window are inserted into an output stream, a k-NN/w data stream processor performs data filtering.

Consider, for example, the following motivating scenarios. (1) A large number of wireless sensor networks are placed along the coast of the Adriatic sea, which is well-known for its indented coastline and numerous islands, to monitor the quality of the sea water in real-time. Environmental scientists would like to identify and monitor up to ten sites with the largest pollution readings over the course of a single day, so that special teams may be alerted to investigate on-site causes of pollution. They may also identify ten sensors closest to a particular location measuring the largest pollution levels over time. Note that the top-10 readings would be provided on, e.g., hourly bases. (2) Sensor Web technology is used to monitor smart grids for efficient energy supply and distribution which integrates renewable energy sources, e.g. numerous solar panels over a large region. As power grids are highly interrelated, it is vital to monitor them in real-time. Power grid operators would like to monitor over time 100 sites with the largest or the lowest power production by using solar panel current and voltage readings so that they identify power grid hot-spots. (3) The last scenario is not from the Sensor Web domain, but can also benefit from a k-NN/w processing solution over structured data. Consider an "auction site super-network" spanning over many online auction sites. This network enables a user to define his/her ideal product of interest, and to receive, e.g., top-10 offers within the course of a day that are the most similar to his/her ideal product.

The listed motivating scenarios require a solution for k-NN/w processing which spans over distributed data sources generating streams of structured data at high rates. Two types of systems can be applied for the implementation of k-NN/w processing: data stream processing systems (DSPSs), and publish/subscribe systems. DSPS are designed for effective and efficient data filtering by executing continuous queries over data streams [22]. Publish/subscribe systems process subscriptions, i.e. continuous queries, in distributed environments to find matching publications, i.e. data objects, published by heterogeneous and distributed data sources [30]. The major challenges arising in continuous real-time processing of data streams are related to transmission, real-time computation, and storage of data objects. Centralized DSPSs are able to cope with the latter two challenges, but in case of high streaming rates and highly distributed data sources, the transmission of objects to a centralized stream processor becomes the major bottleneck of the whole system. On the contrary, in distributed DSPSs, queries are distributed over several network nodes acting as data stream processors, which then share the processing load and perform processing close to data sources. Due to this fact, distributed DSPS are able to cope with higher rate of streaming data objects compared to centralized solutions, but are also much harder to design and build. Publish/subscribe systems are efficient for distributed processing mainly because of rather simple stateless query definitions.

Although k-NN/w processing has been a particularly active research area in the last years [4, 14, 20], existing works assume a centralized setting. This paper presents the first step towards the challenging problem of distributed k-NN/w processing by combining the distributed nature of publish/subscribe systems with efficient data stream processing. Our system is composed of a network of processing nodes, where each node performs k-NN/w processing over a subset of queries and data objects. Queries and data objects are assigned to responsible nodes using the underlaying CAN network. Each k-NN/w processing node implements one of our original centralized algorithms for efficient top-k/w processing over data streams which is available in [26]. The main contributions are summarized as follows:

1. We present a formal model for processing k-NN/w queries over certain data streams. The k-NN/w processing model regards the processing engine as a black-box, and discusses distribution-related issues such as query and object propagation.
2. We propose a distributed peer-to-peer k-NN/w processing system that is built on top of the CAN structured overlay [27], and design protocols for query and data processing, and node joins, departures, and failures.

3. We present results of an extensive experimental evaluation clearly showing the proposed solution is scalable for increasing number of queries and peers, while the messaging overhead remains low.

The paper is organized as follows. In Section 2 we give a survey of related work in the field of DSPSs and distributed publish/subscribe systems. Section 3 presents a formal k-NN/w data stream processing model. In Section 4 we briefly present our centralized k-NN/w processing algorithms as they are the building blocks of the distributed engine. Section 5 presents D-ZaLaPS,[1] a distributed DSPS supporting k-NN/w queries. In Section 6 we analyze results of an extensive experimental evaluation and conclude the paper with Section 7.

2 Related work

The processing of continuous sliding-window top-k queries (top-k/w processing) over data streams has attracted considerable attention in recent years due to its potential application in many different areas such as environmental monitoring using wireless sensor networks, information filtering, computer and telephone network monitoring, financial and stock trade monitoring, etc. All current works on top-k/w processing [4, 9, 11, 13, 14, 18–20, 25, 26, 33] assume centralized processing at a single network node and thus differ significantly from the distributed k-NN/w processing approach we present in this paper. These works can be classified in two categories: deterministic approaches [4, 9, 11, 18–20, 26] which produce correct results to defined queries, and probabilistic approaches [13, 14, 26] which generate errors and thus produce approximate results, but are in general more efficient and require less memory than the deterministic approaches. Furthermore, as a top-k/w query continuously identifies k best-ranked data objects in the query window with respect to an arbitrary scoring function, we can additionally classify these works according to whether distance [4, 14, 20], aggregation [9, 18, 33] or relevance [11, 19] scoring function is assumed. Following this categorization, k-NN/w queries are top-k/w queries with distance scoring functions. Hereafter we list the most relevant papers dealing with centralized top-k/w and k-NN/w processing.

In our previous works [25, 26], we present a probabilistic and deterministic algorithm for efficient centralized processing of generic top-k/w queries. A generic top-k/w query may define any of the three possible categories of scoring functions. It is important to mention that our probabilistic algorithm assumes the random-order data stream model, in which any permutation of streaming objects is equally likely to appear in a data stream. Recently, this model has attracted a lot of attention as it often describes real-world applications much better than the worst-case model [6, 7, 10, 13]. Although in this paper we extend and apply our previously developed centralized algorithms, this paper significantly differs from our previous work since it focuses on distributed processing of k-NN/w queries.

[1]Zagreb–Lausanne distributed top-k/w publish/subscribe system supporting distance scoring functions.

The first paper addressing the problem of deterministic top-k/w processing is [18]. It presents two algorithms named *skyband monitoring algorithm* (SMA) and *top-k monitoring algorithm* (TMA) for processing of top-k/w queries with aggregation scoring functions. In contrast to our approach, the authors assume that all queries are defined with the same sliding window size, while all data objects in the current window are stored in memory of the centralized processor. In their subsequent paper [20], the same group of authors present CPM and SNN which are similar to TMA and SMA, and are developed for the problem of k-NN/w monitoring. The algorithms index queries in the regular grid such that the score of k-th ranked object is used as an indexing threshold. We find the algorithms costly for a distributed environment since they perform periodical k-NN computations from scratch which could generate too much network traffic.

Another work relevant to ours is [4], which proposes an algorithm for continuous k-NN monitoring over data streams with limited memory. This paper is the first that introduces a recent buffer to avoid inserting less relevant data objects to the query data structure. The query filter, called the approximate skyline, is based on the presented algorithm and uses the score of an object with rank k in the filter as a threshold for indexing in a regular grid. This approach is most relevant to our deterministic implementation and could be used as a building block of our distributed system. However, it relies on a different type of a query filter which is less efficient than the probabilistic filter.

The paper [9] focuses on processing of ad-hoc top-k/w queries over data streams. An ad-hoc query is interested in data objects that have appeared both before and after the point in time of its activation. In order to perform efficient processing, the paper introduces a geometrical representation of data objects that supports processing using arrangements [12]. In our opinion, it would be very costly to implement a distributed version of this approach since it maps each query to a vertical ray shooting query in the dual plane which makes the indexing of queries quite complicated.

The paper [14] presents a technique for the processing of e-approximate k-NN queries over data streams. It partitions the attribute space using a regular grid such that the maximum distance between any pair of points in a cell is at most e, and keeps at most $K \geq k$ points per cell. The indexing of points is done using B-tree and space-filling curves. Similarly to TMA and SMA, it would also be very costly to implement a distributed version of this approach since it periodically performs k-NN search.

The distributed k-NN/w processing solution that we present in this paper is built on top of the CAN structured overlay [27]. Due to the specifics of k-NN/w queries, which are interested in data objects that are closest to a query point, the CAN overlay network is much better suited for this type of queries than the competing overlay networks such as Chord [28] and P-Grid [1]. The most similar distributed systems to the one that we present in this paper are [29, 30, 34] since they present structured peer-to-peer publish/subscribe systems which are also built on top of the CAN overlay network. However, when compared to the stateful top-k/w queries in which we are interested in this paper, these approaches only support much simpler stateless continuous queries defined as static subspaces of the given attribute space.

Distributed processing of top-k queries in structured peer-to-peer networks [5, 16, 17, 23, 31, 32] is also related to our work, but while we are interested in continuous queries, these works focus on efficient processing of one-time queries.

3 Sliding-window k-NN processing model

Figure 1 sketches our sliding-window k-nearest neighbor (*k-NN/w*) processing system. It is composed of a k-NN/w processor, which can be either centralized or distributed, and a set of clients. Clients produce data objects which cannot change after entering the processor, and activate and cancel their k-NN/w queries. The processor accepts k-NN/w queries, and reports matching k-NN/w data objects to the clients. We assume queries reference future data objects, i.e. objects entering the system after query activation, and cannot reference past objects published before query activation. Note that in this section we assume the processor is composed of a single processing node to simplify definition of its functionality, while the distributed implementation consisting of a network of k-NN/w processing nodes is presented in Section 5. The processor memory stores both active queries and a set of data objects from the input stream needed for k-NN/w processing. Each incoming data object is seen only when entering the processing system unless it is explicitly stored in memory. As the processing efficiency largely depends on the number of data objects maintained in memory, the goal is to store the minimal set of stream objects necessary for k-NN/w processing. Moreover, if we assume memory size is relatively small compared to the size of data within the stream window, only a restrictive subset of data objects can be maintained in memory.

In this section we first present a model for continuous k-NN/w processing which forms the basis for k-NN/w system implementation. Without loss of generality, the model is built assuming time-based sliding windows associated with continuous queries, and may be extended in a straightforward manner to support count-based sliding windows. Second, we provide a problem definition which identifies data stream objects maintained in memory which are necessary for continuous k-NN/w processing.

3.1 Model definition

We define a triple $\boldsymbol{B} = (\boldsymbol{C}, \boldsymbol{O}, \boldsymbol{Q})$, where $\boldsymbol{C}$ is a finite set of clients, $\boldsymbol{O}$ is a finite set of data objects, and $\boldsymbol{Q}$ is a finite set of queries in a sliding-window k-NN query processing system. $\boldsymbol{B}$ gives the *structural view* of a k-NN/w system and determines

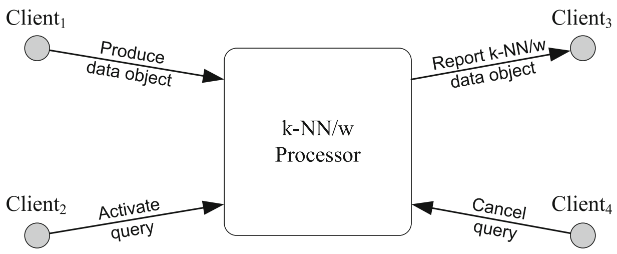


Figure 1 k-NN/w processing system.

the boundaries of the system state space, because it defines the type and number of entities that can exist in a system. A client $c \in \boldsymbol{C}$ may produce data objects from $\boldsymbol{O}$, and activate or cancel queries from $\boldsymbol{Q}$.

We define data objects and queries in the k-NN/w system as points in a d-dimensional Euclidean space.

Definition 1 (Point) A point p in a d-dimensional Euclidean space $\mathbb{R}^d$ is an ordered d-tuple $p = \{v_1, v_2, \ldots, v_d\}$, where $\forall i \in \{1, 2, \ldots, d\} : v_i \in \mathbb{R}$ is the value of d-th coordinate of p.

Definition 2 (Data object) Let p_o be a point in a d-dimensional Euclidean space $\mathbb{R}^d$, let $c_o \in \boldsymbol{C}$ be a client, and let t_o^A be a point in time. We define a triple $o = \{p_o, c_o, t_o^A\} \in \boldsymbol{O}$ as a data object produced by client c_o at t_o^A.

An object has an associated *time of appearance* t_o^A denoting a point in time when object o appears in the system. We assume the object content is certain, and its content and time of appearance do not change after entering the system. We further assume data objects are assigned unique t_o^A when entering the processing system such that all objects can be ordered by their time of appearance. This is indeed true for a single processing node which can assign unique timestamps to incoming objects, while this assumption does not hold in a distributed setting. However, the assumption is further relaxed in a distributed setting such that each processing node needs to assign unique timestamps only to its incoming data objects, while timestamps need not be unique over the entire processing network.

Hereafter we identify the characteristics of k-NN/w queries supported by the k-NN/w system. Each query is defined as a point in a d-dimensional Euclidean space associated with two parameters, the size of the time-based sliding window w, and parameter k—the number of k-NN objects from the query window. Furthermore, as queries are continuous, they have a predefined time of activation and cancellation.

Definition 3 (Sliding-window k-NN query) Let p_q be a point in a d-dimensional Euclidean space $\mathbb{R}^d$, let $k_q \in \mathbb{N}$ and $w_q \in \mathbb{R}^+$, let t_q^A and t_q^C be two points in time such that $t_q^A < t_q^C$, and let $c_q \in \boldsymbol{C}$ be a client. We define a sextuple $q = \{p_q, c_q, k_q, w_q, t_q^A, t_q^C\} \in \boldsymbol{Q}$ as a continuous k-NN query over time-based sliding window of size w_q defined by client c_q.

A query is associated with *time of appearance* t_o^A denoting a point in time when q enters the processing system, and *time of cancellation* t_o^C. Analogously to objects, we assume implicit timestamps for queries which are assigned by the processing nodes. Of course, the transmission of queries and data objects to processing nodes introduces certain latency compared to the time of creation at the source nodes. However, we can provide correctness guaranties regarding the resulting data set by tolerating bounded query/object propagation times, as it is commonly done in distributed implementations.

Depending on the time of object appearance, and the time of query activation and cancellation, an object is either valid or invalid for a query.

Definition 4 (Valid data objects for a query) Let $q \in \boldsymbol{Q}$ be a k-NN/w query. We define the set of valid data objects $V_q \subseteq \boldsymbol{O}$ for q as

$$V_q \stackrel{\text{def}}{=} \{o : (o \in \boldsymbol{O}) \wedge (t_q^A < t_o^A \leq t_q^C)\}. \tag{1}$$

In other words, a data object is valid for a query if the object appears during the period of time when the query is active.

The following definition specifies the set of data objects within the query window at t. Note that queries use different window sizes, and therefore data objects within the windows of different queries are also different at a point in time.

Definition 5 (Data objects in a query window) Let $q \in \boldsymbol{Q}$ be a k-NN/w query, let t be a point in time when q is active. We define the set of data objects in the query window $W_q(t) \subseteq \boldsymbol{O}$ of q at t as follows:

$$W_q(t) \stackrel{\text{def}}{=} \{o : (o \in V_q) \wedge [t_o^A < t \leq (t_o^A + w_q)]\}. \tag{2}$$

In other words, a data object is within the query window at a point in time t, if the object is valid for the query, and less than w_q time units have passed since its appearance. We are therefore supporting *queries referencing future data objects*. We denote a point in time when o is dropped from the query window by t_o^D such that $t_o^D = t_o^A + w_q$.

In addition to ordering objects by their time of appearance, objects may also be ordered based on their content. Thus we define a scoring function for calculating object-specific scores which enables object ranking with respect to a query.

Definition 6 (Scoring function) We define the scoring function $u : \boldsymbol{Q} \times \boldsymbol{O} \mapsto \mathbb{R}$ as follows

$$u(q, o) = d(p_q, p_o) = \sqrt{\sum_{i=1}^{d} (v_i - v_i)^2}, \tag{3}$$

where $q \in \boldsymbol{Q}$, $o \in \boldsymbol{O}$, and $p_q = \{v_1, v_2, \ldots, v_d\}$ and $p_o = \{v_1, v_2, \ldots, v_d\}$ are elements from a d-dimensional Euclidean space $\mathbb{R}^d$.

The scoring function is defined as the Euclidean distance between points representing an object and query. Since we are interested in nearest neighbor points, points with smaller scores are assigned with higher ranks with respect to the query. The following definition specifies k-NN objects from the query window at t.

Definition 7 (k-NN objects in a query window) Let $q \in \boldsymbol{Q}$ be a k-NN/w query, and let t be a point in time when q is active. We define the set of k-NN data objects in the query window $T_q(t) \subseteq W_q(t)$ for q at t as follows:

$$T_q(t) \stackrel{\text{def}}{=} \{o : (o \in W_q(t)) \wedge (|B_q(o, W_q(t))| < k_q)\}, \tag{4}$$

where $B_q(o, W_q(t)) \stackrel{\text{def}}{=} \{o' : (o' \in W_q(t)) \wedge (u(q, o') < u(q, o)\}$ is the set of data objects from $W_q(t)$ with better ranks for q than o at t.

In other words, a data object o is a k-NN object for a query q at t if and only if it is among k_q best ranked objects, i.e. if p_o is among k_q closest ponts to p_q, within the query window. The set of k-NN/w objects is continuously being updated because, although object scores do not change over time, their rank with respect to the query changes as new objects appear while others are dropped from the window. Next, we define under which conditions a data object must be reported to a client as a result object, and define a set that forms the correct answer to a k-NN/w query.

Definition 8 (Query result set) Let $q \in \boldsymbol{Q}$ be a k-NN/w query of a client $c_q \in \boldsymbol{C}$. We define the query result set $R_q \subseteq \boldsymbol{O}$ for q as follows:

$$R_q \overset{\text{def}}{=} \{o : (o \in \boldsymbol{O}) \wedge \exists t(o \in T_q(t))\}. \tag{5}$$

The result set associated with a k-NN/w query q is deterministically correct if and only if it contains all data objects from $\boldsymbol{O}$ that are k-NN objects in the query window at any t, such that $t_q^A < t \leq t_q^C$. Each data object $o \in R_q$ has to be reported to client c_q with an active query q at a point in time t_o^R when o becomes an element of R_q.

3.2 Problem statement

Let us now discuss when an object o becomes an element of R_q, i.e. the point in time t_o^R. According to Definition 7, an active object within a query window becomes a k-NN query object when there are less than k_q higher ranked objects within the query window. This can happen under the following circumstances:

1. at t_o^A when o enters the processor if and only if there are less than k_q higher ranked objects within the query window, and
2. at a later point in time $t_o^A < t \leq t_o^D$ when one of k-NN objects is dropped from the query window and o has the highest rank among all other objects from the window that are not within k-NN objects.

The implementation of the first scenario is quite straightforward: Each incoming data object is compared against the list of current k-NN query objects, and, in case its rank is higher than the rank of the last k-NN object, it is added into the query result set. This requires continuous maintenance of k-NN objects in memory, because, although these objects have already been inserted into the query result stream, they are continuously compared to arriving objects. Additionally, we need to detect empty slots in the k-NN list due to object droppings from the query window to implement the second scenario.

The second scenario requires a more elaborate algorithm as it has already been recognized in [4, 13, 20]: The processor needs to instantly fill the slot of a dropped k-NN object with an object that currently has the best rank among non k-NN objects within the query window. Therefore, the processor needs to store additional *candidate data objects* in memory that have the potential to become k-NN objects. A straightforward solution would store all data objects within the query window in memory. However, since the number of objects in the query window can in general be much larger than k, and as we assume that memory is limited, our goal is to minimize the number of candidate objects maintained in memory.

Let us discuss which data objects within the query window have the potential to become k-NN/w objects in future. A data object *dominates* another object for a given query if and only if the former object is more recent (i.e. younger) and has a higher rank for the query than the latter object. It has already been shown that a data object which is at a point in time dominated by k or more objects in the query window, cannot become a k-NN/w object in the future for this query [4, 18, 20]. Therefore, only those objects which are *non-dominated* (i.e. dominated by less than k objects) at a point in time have potential to become k-NN/w objects in the future. According to [24], the set of such data objects (at a point in time) is called *k-skyband*, while *k-skyline* is a set of points that are dominated by $k-1$ other points. The set of objects within the k-skyband at a point in time contains all k-NN objects, and objects that have the potential to become k-NN/w in future. Therefore, if a query k-skyband is maintained in memory over time, it can be used to deterministically answer the associated k-NN/w query.

The aforementioned problem of continuous k-skyband maintenance is related to the sliding-window k-NN processing model since it enables model implementation by maintaining the minimal and deterministic set of candidate objects that may over time be reported as k-NN/w objects. Efficient maintenance of a k-skyband over time is a non-trivial task, both in case of a centralized and distributed processor architecture. In the centralized architecture k-skyband maintenance is implemented at a single node, while in the distributed case each processing node maintains k-skybands for queries under their responsibility. Thus in the following section we discuss the algorithms for efficient k-skyband maintenance at a single k-NN/w processing node because it is the building block of our distributed k-NN/w processing system introduced in Section 5.

4 Centralized sliding-window k-NN processing

In this section we briefly present two original centralized sliding-window top-k processing algorithms that form the building blocks of our distributed system: *Relaxed Candidate Pruning Algorithm* (RA) and *Probabilistic Candidate Pruning Algorithm* (PA). RA is deterministic and is based on the maintenance of a relaxed k-skyband with candidate k-NN/w objects. PA is probabilistic and uses a probabilistic criterion to decide whether a newly appearing object has good chances to become a k-NN object in future or not. A detailed presentation of RA and PA along with a complexity analysis and extensive comparative evaluation is available in [26]. The evaluation has been performed in a centralized settings using both synthetic and real datasets, and has shown that RA and PA significantly outperform the competing approaches for centralized sliding-window top-k processing, both in terms of memory consumption and processing efficiency. Both algorithms are generic, i.e. they support various sliding-window top-k queries using arbitrary scoring functions, however, in this paper they have been tailored to the specific problem of k-NN/w processing. Please note that RA can be applied to all types of data stream sources, while PA performs best in case when data objects are generated according to the random-order data stream model. RA generates correct results at the expense of increased memory consumption and processing performance compared to PA, while PA may generate false positive or false negative matching objects with a controllable probability of error.

Furthermore, we briefly outline *Relaxed Candidate Pruning Algorithm with Probabilistic Filter* (RAPF), a more efficient version of RA which uses a probabilistic filter for delaying the insertion of recent objects with low ranks into the relaxed k-skyband. To simplify discussion, we present RA, PA, and RAPF with respect to a single k-NN/w query $q = \{p_q, k_q, w_q, t_q^A, t_q^C\}$ and the associated k-skyband maintenance. Finally, we discuss the implementation which supports multiple queries, and explain the indexing of k-NN queries in a regular grid.

4.1 Relaxed candidate pruning algorithm (RA)

As previously stated, the problem of k-NN/w processing can be reduced to the maintenance of a k-skyband containing non-dominated objects. According to our experiments [26], continuous maintenance of a strict k-skyband produces large processing overhead due to immediate pruning of dominated objects after each object arrival. Therefore, RA applies lazy pruning and allows referencing of additional dominated data objects in memory in a *relaxed k-skyband*. RA periodically prunes dominated objects from the relaxed k-skyband and improves the processing performance, while memory consumption is slightly increased compared to the strict k-skyband maintenance.

We represent the relaxed k-skyband in memory using two self-balancing binary trees, top tree and candidate tree, as shown in Figure 2. Each object is associated with two attributes: its score with respect to a query (written inside the square which represents an object in Figure 2), and its time of appearance. The *top tree* stores *k*-NN data objects from the current window, while the *candidate tree* stores other non-dominated objects for the query. Additionally we use a time tree to sort objects by their time of appearance to efficiently locate objects which are dropped from a query window.

The following parameters are associated with a query: (1) *pruning coefficient* γ, and (2) *candidate tree limit*. The pruning coefficient represents the percentage of the acceptable candidate tree size overhead compared to the strict k-skyband size, and is used to calculate the candidate tree limit which determines the maximum candidate tree size. When the candidate tree limit is reached, it triggers the process of relaxed k-skyband pruning.

The processing of an incoming data object is done in three steps, as shown in Figure 2. We first compare the incoming object score with the score of the top tree tail. In the second step, the object is inserted either to the top tree if its score is lower than the top tree tail, or otherwise to the candidate tree. In the third step, the process

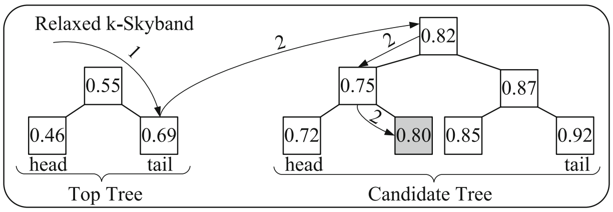


Figure 2 *RA*: Three steps to add object *o* into to the relaxed k-skyband: *1* compare *o*'s score with the top tree tail, *2* insert *o*, and *3* prune dominated objects if necessary.

of pruning dominated objects is triggered if the number of objects in the candidate tree is larger than the predefined limit. This process will reduce the relaxed k-skyband to the strict k-skyband containing the minimal set of non-dominated objects in the candidate tree.

Note that all top tree objects have already been reported as k-NN/w objects upon insertion into the top tree, while candidate tree objects are waiting for their opportunity to become k-NN/w objects when top tree objects are being dropped from the query window. In particular, the candidate tree head will be moved to the top tree to fill in the empty slot, and will subsequently be reported as a k-NN/w object to the client. When a top tree object is dropped, it will be removed from the time tree and top tree while its empty slot in the top tree will be filled by the candidate tree head. The removal process from the top tree is straightforward because all other k-skyband objects are younger and cannot be dominated by the removed object.

4.2 Probabilistic candidate pruning algorithm (PA)

In practice, most of the objects from a query k-skyband will never become k-NN/w objects because k is typically much smaller than the number of non-dominated window objects, and will never be reported as answers to a client. Our original algorithm PA utilizes this fact and prunes k-skyband objects that have small probability to become top-k objects in future. It therefore maintains a significantly smaller set of candidate data objects compared to RA, and is more efficient than deterministic algorithms since memory consumption is the most important efficiency measure for all streaming algorithms [13].

PA uses a probabilistic criterion originally published in [25] to decide upon object arrival whether the object has good chances to become a top-k object with respect to a query in future or not. If the computed probability of becoming a top-k object is above a predefined threshold, the object is maintained in the probabilistic k-skyband; otherwise, it is discarded. PA is based on the assumption that the sequence of data objects entering the processor follows the *random-order data stream model* for which any permutation of streaming data objects is equally likely to appear in a stream. This does not imply that object scores are random for a query, but that data objects are drawn independently from a non-time-varying distribution which characterizes a number of independent data sources, e.g. RSS feeds or large sensor networks. Random-order data stream model was originally introduced in [21], which is one of the first papers in the field of data stream processing with limited memory, and has been used to describe and analyze a number of real-world application scenarios [6, 7, 10, 13]. In comparison to deterministic algorithms, the main drawback of PA is that, as a probabilistic algorithm, it produces approximate results.

Analogously to RA, PA maintains the probabilistic k-skyband in memory using two self-balancing binary trees sorted by ascending ranks, as depicted in Figure 3, and an additional time tree. The probabilistic k-skyband structure is similar to the relaxed k-skyband, and each object is associated with its score and time of appearance. The *top tree* stores k-NN data objects, while the *candidate tree* stores other referenced objects for a query, while the maximum number of such objects is smaller than the probabilistic limit. Two attributes are associated with each query: (1) *probabilistic limit* on the number of candidates, and (2) *threshold*. The probabilistic limit denotes the maximum candidate tree size and is calculated using the probabilistic criterion

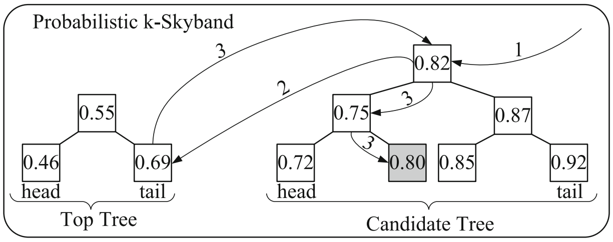


Figure 3 *PA*: Four steps to add object *o* to the probabilistic k-skyband: *1* compare *o*'s score with the threshold, *2* compare *o*'s score with the top tree tail, *3* insert *o*, and *4* update the threshold.

and a predefined acceptable probability of error. The threshold value is set to the score of the candidate tree tail object.

The processing of an incoming data object is done in four steps, as shown in Figure 3. We first compare the object score with the threshold, and abort the procedure when it is higher that the threshold because the object has poor chances to become a k-NN/w object, or otherwise continue with the second step. In the second step we compare the object score with the score of the top tree tail. In the third step, the object is inserted either to the top tree if its score is lower than the top tree tail, or to the candidate tree. The candidate tree tail is removed from the probabilistic k-skyband if the number of objects in the candidate tree is larger than the probabilistic limit, and subsequently the threshold is updated. If the object is added to the top tree such that its size becomes larger than *k*, we move the top tree tail to the candidate tree.

4.3 Relaxed candidate pruning algorithm with probabilistic filter (RAPF)

RA inserts all incoming data objects into the relaxed k-skyband because all arriving objects are non-dominated as they are the youngest. Many such objects may have low ranks with respect to a query, and will soon become dominated by new higher ranked and more recent objects. The insertion of such objects into the k-skyband wastes a lot of resources, and thus we design an enhanced version of RA which filters out recent objects with low ranks and delays their insertion into the relaxed k-skyband. The algorithm is named Relaxed candidate pruning Algorithm with Probabilistic Filter (RAPF) because it reuses the probabilistic criterion applied by PA to implement the probabilistic filter.

To enable filtering of recent objects, we employ *recent buffer*, a special FIFO buffer of b most recent incoming objects represented in memory as a singly-linked list of objects (b is typically much smaller than the number of query window objects). We associate an auxiliary probabilistic k-skyband to a query—*probabilistic filter* (PF)—filled with data objects from the buffer. Note that a filter is quite similar to a query: the major difference is in the set of objects that are of interest to a filter, i.e. the b most recent objects from the stream compared to all objects within the window of size w_q.

The filter is probabilistic because the number of objects maintained in the filter k-skyband is calculated using the probabilistic criterion defined for PA. It enables filtering of objects with low ranks by avoiding their insertion into the relaxed k-skyband associated with a query at the time of object arrival: An object o is rather temporary maintained in the probabilistic filter, and the filter makes the second attempt to insert o into the query k-skyband, just before it is dropped from the buffer.

At a later point, *o* will probably be dominated by younger objects with higher ranks, and will be discarded, unless the quality of incoming data objects with respect to the query changes such that newly arriving objects mainly have lower ranks than *o*. In the second insertion attempt we insert *o* in the relaxed k-skyband only if less than *k* higher ranked objects are left in the filter, and it was not inserted in the relaxed k-skyband during the first attempt. Otherwise, *o* is discarded since it is dominated for the query by *k* or more objects in the buffer.

Note that RAPF is a deterministic algorithm, although it uses the probabilistic filter, because of the two insertion attempts into the query k-skyband for each incoming data object. If an object is not among *k-NN buffer objects* when it enters the system, its insertion into the query k-skyband is delayed. The buffer size is chosen such that the object cannot become a *k-NN query object while in the buffer* because of other better ranked objects within the window. When the object is dropped from the buffer, it may become a k-NN object in the window and therefore we perform the second insertion attempt into the query k-skyband. Only non-dominated buffer objects are inserted into the query k-skyband in the second insertion attempt, and therefore the probabilistic filter is needed to maintain such non-dominated buffer objects. The proof of the aforementioned property, as well as the condition for choosing an adequate buffer size is available in [26]. Moreover, our experiments show that RAPF significantly outperforms RA in terms of processing efficiency although it stores all buffer objects in memory,[2] and requires the maintenance of an additional filter k-skyband per each query. However, the filter k-skyband is probabilistic and contains a small number of objects compared to the number of objects in the relaxed query skyband. Additionally, it enables query indexing which enables a highly-efficient algorithm implementation for multiple query processing, as explained in the following subsection.

4.4 Supporting multiple k-NN/w queries

When multiple k-NN/w queries are simultaneously active in the system, a few additional data structures and minor algorithm modifications are needed when compared to single k-NN/w query processing. The following data structures are introduced:

1. *query tree*, a self-balancing binary tree for storing all active queries and query filters in the system;
2. *object tree*, a self-balancing binary tree for storing all data objects that are referenced by at least one query k-skyband or filter;
3. *common time tree* shared among all queries and filters in the system since it is more efficient than having separate time trees for different queries; and
4. *single recent buffer* shared among all filters in the system.

If we closely analyze PA, we conclude that a query does not need to be informed about a newly arrived object if its score is higher than the k-NN/w query threshold. On the contrary, RA works by processing all incoming data objects and does not define such a threshold. However, RAPF defines a threshold as the score of the filter element with rank *k*, because in both insertion attempts only objects with ranks

[2]Note that the buffer is shared by all queries processed on a single processing node.

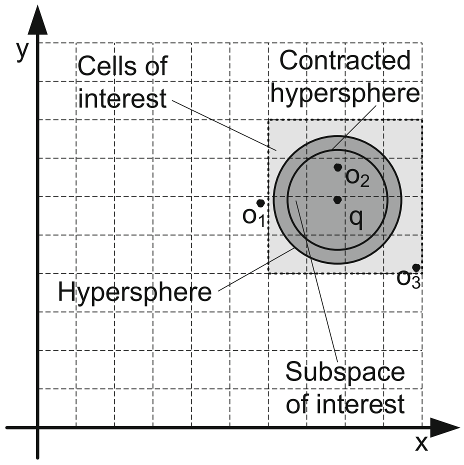


Figure 4 Indexing of k-NN queries in a regular grid.

higher than this object are inserted to the corresponding relaxed query k-skyband. Note that all thresholds values change in time with object arrivals and droppings.

Query indexing reduces the number of data objects a query needs to process by skipping those objects that are certainly not potential k-NN/w objects for the query, and identifies the attribute subspace of interest for the query using the identified threshold. We use a regular grid for indexing k-NN/w queries: It divides an attribute space in cells of equal size, while a threshold defines the subspace of interest for a query. Therefore, each threshold represents the radius of the subspace of interest around a query point. Please note that the space of interest is covered by the cells of interest from the regular grid that encompass a larger portion of the attribute space than the subspace of interest itself. An example query with a hypersphere delimiting its subspace of interest, and the corresponding cells of interest is depicted in Figure 4. Since o_2 and o_3 fall within the cells of interest of query q, the query will be informed about the appearance of o_2 and o_3, but not about object o_1 which is outside the cells of interest. Note that a query subspace of interest is varying in time due to continuous threshold changes and can either expand or contract.

Query indexing changes PA and RAPF implementation in the following way: Upon each object arrival, we first find the grid cell in which the new object resides, and then find a subset of queries whose cells of interest encompass this cell to inform them about the appearance of the new object. Therefore only a subset of active queries processes an incoming object instead of all active queries which is necessary in case when a processing node applies RA.

5 Distributed sliding-window k-NN processing

The major challenges arising in real-time processing of data streams are related to real-time computation, storage, and transmission of data objects to the processing node [22]. The centralized k-NN/w processing algorithms presented in Section 4 (RAPF and PA) can deal with the first two issues since they are designed for efficient processing with low memory consumption. However, centralized processing cannot deal with the third issue especially in environments with highly-distributed data sources because object transmission introduces significant latency and generates huge network traffic. Moreover, a centralized solution is not scalable for an increasing number of clients and queries because a single k-NN/w processor becomes the major system bottleneck.

The scalability problem can be solved by employing a network of k-NN/w processing nodes, and by distributing the queries and thus processing load among such nodes. Each node is built as a k-NN/w processor implementing either RAPF or PA. It processes incoming data objects to deliver k-NN objects to querying clients under their responsibility, and acts as a representative for both querying and data producing clients since true data produces/consumers typically connect to the closest nodes in terms of network distance. Such distributed architecture can balance the load among k-NN/w processing nodes expressed as the number of queries under each node responsibility. The next question arising from such network organization is related to distribution of incoming data objects over processing nodes. A simple solution that floods data objects across all processing nodes is unscalable in terms of generated traffic similarly to the centralized solution. It is therefore necessary to distribute the queries and incoming data objects across the nodes such that an incoming object needs to be processed by a minimal set of k-NN/w processing nodes (preferably even by a single node).

Peer-to-peer (P2P) networks offer the means for distributing queries and data objects in a decentralized fashion while balancing the load among the peers, i.e. nodes. We employ CAN [27] as the underlying structured overlay interconnecting the k-NN/w processing nodes. CAN arranges a multi-dimensional Euclidean space between the nodes in a self-organizing way. Since in our model data objects and queries are represented as points from the Euclidean space while the distance between two points defines our scoring function, CAN was a natural choice for the underlying structured overlay.

This section presents D-ZaLaPS, a distributed k-NN/w processing system which successfully copes with the previously mentioned stream processing challenges. It is a hybrid between a P2P publish/subscribe system and data stream processing engine built on top of CAN that relies on indexing of k-NN queries in a regular grid as explained in Section 4.4. In the following we explain how k-NN/w nodes are mapped onto CAN, and later on describe the protocols for query activation/cancellation and data object publication in our distributed system architecture. Furthermore, we introduce the protocols for peer joins, departures, and failures.

5.1 CAN overlay for k-NN/w processing

Content Addressable Network (CAN) is a structured peer-to-peer overlay network which dynamically partitions a d-dimensional Euclidean space on a d-torus into zones. It assigns non-overlapping zones to the peers, and ensures that peers in neighboring zones are connected, while each part of the Euclidean space has a responsible peer. Every peer maintains a routing table storing the IP address and information regarding assigned zone for each of its neighbors. This purely local information is sufficient to route a message between any two arbitrary points in the space: The points are assigned to responsible peers in their zones and each peer on the path from the source to destination peer forwards a message to a selected neighbor that is the closest to the destination point in the Euclidean space. CAN also supports dynamic joining/leaving of peers, and enables load balancing between the peers such that the Euclidean space is divided according to the current peer load, and not evenly. For details regarding CAN please refer to [15, 27].

Let us discuss how D-ZaLaPS divides the *d*-dimensional CAN space among the processing nodes, and assigns responsibility for query activation/cancellation and data processing. It partitions the attribute space to cells as in the grid to identify cells of interest for a query, and assigns responsibility over CAN zones to peers, where each zone comprises a set of cells. However, please note that the size and the distribution of the cells depends on the distribution of data in the CAN space. For example, in the case of uniform data, all cells should be equal in size and uniformly distributed in the CAN space. On the other hand, in the case of clustered data, cells should be smaller and placed more densely around the center of a cluster. Both queries and data objects fall into cells from the Euclidean space based on the points which represent them, and each peer is responsible for processing those queries and data objects assigned to cells within its zone of responsibility.

For example, Figure 5 shows a two-dimensional CAN space consisting of 32 zones assigned to 32 peers. Zone boundaries are depicted by solid lines while peers are represented by black dots. The CAN space is also partitioned into cells of equal size and cell boundaries are depicted by dotted lines. It is visible that some zones comprise 2, while other comprise four cells. A new peer joining the network may split a two-cell zone with an existing peer and therefore also one-cell zones are possible in our example. Note that zones cannot be smaller than cells.

The additional partitioning of zones into cells is needed to reduce the message overhead when updating query thresholds, as explained later in this section. Our approach can be further generalized to support zones comprising cells of different sizes: Each peer would have to partition its zone into cells, and inform its neighbors about the partitioning of its zone, similarly to zone partitioning which CAN uses. In addition to having zone-related information of neighboring peers, each peer would also maintain information regarding cell partitioning of each zone. For distributing cell partitioning information between neighboring peers we use existing messages

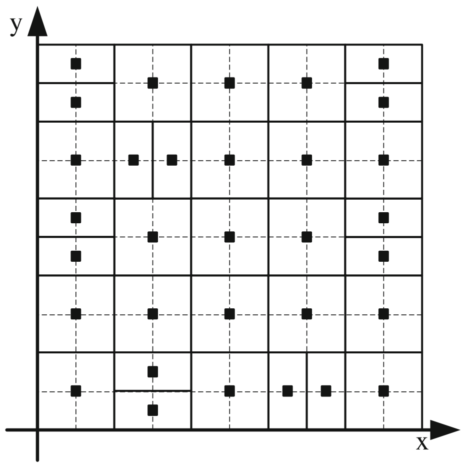


Figure 5 The partitioning of CAN space into zones and cells.

defines by the CAN protocol, *neighbor inform* messages, which are periodically exchanged between neighboring peers. Balanced zone partitioning solves the problem of cell overload, while CAN deals with load balancing at the peer level. However, to simplify the presentation, let us assume that cells are of equal size, and that they correspond to cells within the regular grid.

5.2 Protocols for query and data processing

Let us now explain the algorithms for query and data routing. Each peer is the *rendezvous peer* for all queries and data objects belonging to its cells of responsibility. In this way, each newly appearing object and activated query is routed to the responsible rendezvous peer. A rendezvous peer processes only queries and incoming data objects under its responsibility. As producing and consuming clients are connected to peers, a peer can act as a *source peer* for a data object and an *owner peer* associated with an activated query. Note that an owner peer is probably not the rendezvous peer for its own queries, although such situation is possible in practice.

A query under the responsibility of a peer may cover the cells of interest outside the zone of peer responsibility because cells of interest for a query change over time, and each rendezvous peer is usually responsible for a set of queries. Therefore, a peer needs to interact with its neighboring peers to receive data objects of interest for its queries, although the peer is not responsible for processing such queries. To reduce the number of exchanged messages between neighboring peers, each peer *merges* the queries it is responsible for in a special *merger* which covers all subspaces of interest of the merged queries. The subspace of interest of such a merger is always mapped to cells, while peers responsible for the cells are informed with a *merger update message* that the rendezvous peer is interested in data objects belonging to their cells. Therefore, such peers forward data objects falling into cells under their responsibility to interested rendezvous peers for further processing.

Query activation When a new query is activated, the owner peer uses the CAN protocol to route a *query activation message* to the rendezvous peer responsible for the query. Subsequently, the rendezvous peer merges the new query with other queries under its responsibility. If the created merger expands to additional cells due to new query activation, the rendezvous peer will inform all peers responsible for cells within the merger that it is interested in data objects belonging to their cells by sending a *merger update message*.

Figure 6 depicts an example query activation in a CAN network. Upon query activation by its client, the owner peer sends a query activation message to the responsible rendezvous peer, as shown in Figure 6a. In Figure 6b we see that the query has expanded the merger of the rendezvous peer, which than sends merger update messages to the neighboring peers covering the new merger subspace of interest.

Query cancellation Similarly to query activation, when a query is canceled, the owner peer sends a *query cancellation message* to the rendezvous peer responsible for the query. The rendezvous peer contracts its merger if necessary, and subsequently sends a *merger update message* to all peers responsible for cells that are no longer of interest.

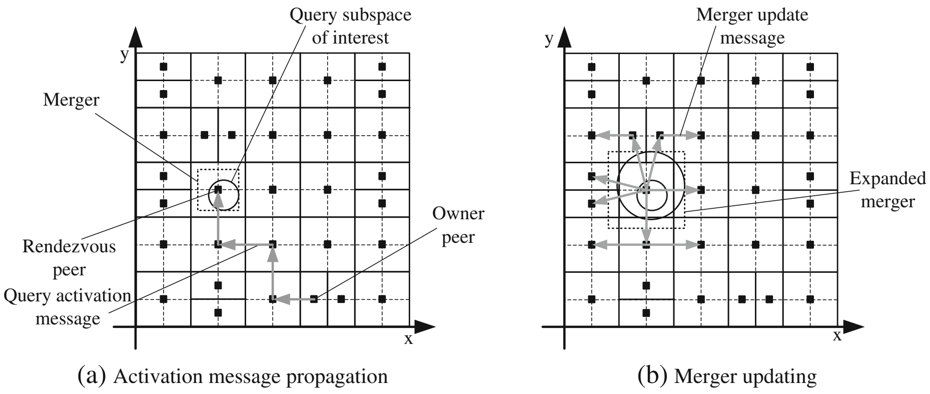
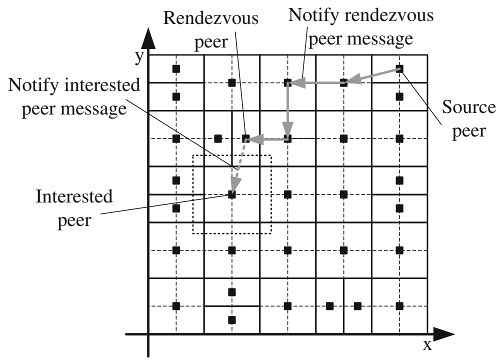


Figure 6 Query activation in D-ZaLaPS.

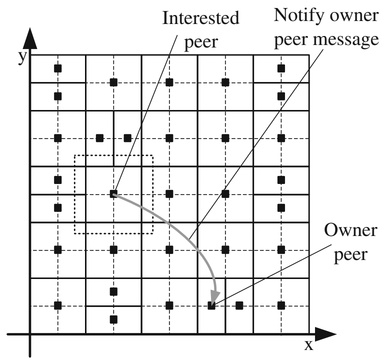
Note that query activation and cancellation expands and contracts a merger of a rendezvous peer causing the generation of merger update messages. We investigate the number of such messages in the experimental evaluation in Section 6, and show that the number of merger update messages would be significantly larger if the attribute space has not been partitioned into cells.

Data object appearance When a new data object appears, the source peer sends it in a *notify rendezvous peer message* to the rendezvous peer responsible for the object. The rendezvous peer performs k-NN/w processing of this object for active queries under its responsibility using either RAPF or PA. From Section 4.4 we know that a query is interested in a data object if object score with respect to the query is lower than the query threshold. When this happens, the rendezvous peer directly forwards the new object to the query owner peer in a *notify owner peer message*. Additionally, it also forwards the object in a *notify interested peer message* to those peers that have previously expressed interest in the cell to which the object belongs to. When a peer receives a notify interested peer message, it performs the regular k-NN/w processing and may report the object as a k-NN/w object to a set of query-owner peers, or store it as a candidate object for some of its queries. The third option is to discard the object as it has low probability to become a k-NN/w object if the node applies PA, or store it temporarily in its recent buffer if the node applies RAPF for k-NN/w processing. Note that a recent buffer has to be maintained at every rendezvous peer, and is shared by all queries under the peer's responsibility. Analogously to the centralized implementation when we try to insert a data object into a k-skyband twice, the rendezvous peer also has to notify interested peers about a data object twice: the first message is sent immediately after object arrival, whereas the second is sent when the object is dropped from the recent buffer.

Figure 7 shows an example scenario for data object activation. The object is first forwarded in a *notify rendezvous peer message* to the responsible rendezvous peer which forwards the object to an interested peer. The interested peer processes the received data object and concludes that this object is a k-NN/w object for the query whose activation is shown in Figure 6. Thus the peer stores it in the query k-skyband,


Rendezvous peer
Notify rendezvous peer message
Notify interested peer message
Source peer
Interested peer
y
x

(a) Propagation of messages: notify rendezvous peer and interested peer



(b) Forwarding a k-NN/w object to the query owner peer

Figure 7 Data object appearance in D-ZaLaPS.

and forwards the object to the query owner peer in a *notify owner peer message*, as shown in Figure 7b.

5.3 Protocols for peer joins, departures and failures

The CAN overlay network handles peer joins, departures, and failures by using special self-organizing protocols. D-ZaLaPS reuses existing CAN protocols for this purpose with minor modifications.

Peer join Upon joining, a new peer randomly selects a point in the Euclidean space and sends a *join message* to the rendezvous peer responsible for this point. The message is routed through the CAN network starting at any CAN peer. Upon receiving the join message, the rendezvous peer splits its zone of responsibility in two equally sized sub-zones while taking into account cell granularity, selects one of two sub-zones for itself, and assigns the other sub-zone to the new peer. The rendezvous peer also hands over the responsibility for queries belonging to the assigned zone by forwarding them to the newly joined peer. Additionally, both the rendezvous and newly joined peer inform their neighbors about the new organization of the CAN subspace.

Peer departure In case of peer departure, the departing peer explicitly hands over its zone of responsibility and the corresponding cells to one of its neighbors. The departing peer initiates the hand over process by sending a *departure message* to the selected neighbor. Similarly to the peer join procedure, the selected neighbor informs all of its current neighbors about the departure. Both peer joins and peer departures affect only a small portion of an existing CAN network which is a desirable property in case of churn.

Peer failure Under normal operating conditions, a peer periodically sends neighbor inform messages to inform its neighbors about the partitioning of its zone of responsibility into cells. The prolonged absence of neighbor inform messages indicates the failure of the corresponding neighbor. When a peer detects such failure, it initiates the CAN mechanism for takeover of the abandoned zone of responsibility by one of the failed peer neighbors. Please note that the abandoned queries (and data objects memorized in the recent buffer) would be lost in the case of peer failure without an additional replication technique. Standard CAN replication techniques can be used for replicating queries and data objects at neighboring nodes.

6 Experimental evaluation

The experimental evaluation represents a feasibility study of the proposed distributed solution, and examines the number of exchanged messages in the peer network due to introduced protocols for query and data processing. In particular, we examine control messages (*notify interested peer* and *merger update messages*) because they are exchanged among neighboring peers, while *query activation/cancellation* and *notify rendezvous peer* messages use the standard CAN routing. Next, we compare a distributed solution to centralized in terms of exchanged messages and processing load per peer. Furthermore, we examine how the varying window size and data/CAN dimensionality affect our distributed system. Finally, we provide a scalability study for an increasing number of queries and peers in the system.

The experimental evaluation is performed using two synthetic and one real data set. In particular, we generate uniform and clustered Gaussian data within the interval [0, 1]. The clustered Gaussian data has two randomly chosen cluster centers and variance equal to 0.1 for each dimension. It has similar properties to the distribution of our real data set which is an excerpt of the LUCE deployment data, environmental data collected from a large-scale wireless sensor network deployed within the project SensorScope.[3] The LUCE deployment data is preprocessed to extract four-dimensional data objects (solar panel current, global current, primary buffer voltage and secondary buffer voltage) and normalized to the values within the interval [0, 1]. You can relate our data set to the motivating scenario (2) introduced in Section 1.

Our CAN network is composed of 256 peers that are randomly selected to generate both queries and data objects, and therefore become query owners and object producers. Each experimental run is performed with the total of 400 queries and 10^6 data objects, while the number of objects within the query window of all queries is set to 40,000. The default simulation parameters used in the evaluation are specified in Table 1.

We evaluate the system in a static network scenario which can be considered as a typical baseline operational scenario for a Sensor Web application. After creating a CAN network using the default CAN protocol, we first produce the set of k-NN/w queries, either by taking a random sample from the LUCE data, or by generating queries using one of the listed distributions. Each query is assigned to a randomly selected owner peer that activates the query. Second, we simulate the generation of data objects, either by randomly choosing data objects from the LUCE data, or by generating data objects using the same distribution as for the previously generated

[3] http://sensorscope.epfl.ch/

Table 1 Simulation parameters.

Parameter	Symbol	Value
Number of data objects	N	10^6
Number of queries	m	400
Intensity of object appearance (objects/window)	λ	40,000
Recent buffer size	b	2,000
Data dimensionality	d	4
Grid resolution	ρ	12
RA: pruning coefficient	γ	0.2
PA: probability of error	σ	10^{-3}
Peers	C	256

queries. A source peer for each simulated data object is chosen randomly from the CAN network. Finally, after N incoming objects are processed, we analyze the obtained query result set. It is important to mention that before each simulation run, we first publish 10^5 data objects to set initial query thresholds for all the queries so that system performance is analyzed under regular working conditions.[4]

Note that we use a regular grid with cells of equal size in all experiments, regardless of the data and query distribution. Such setup is optimal for the uniform data set because the load on the peers is balanced, but it also produces the largest number of control messages compared to the clustered and real data set. Conversely, in case of experiments with clustered and real data sets, peers located at the cluster center are highly loaded, although the number of exchanged messages in the entire network is smaller. The standard CAN mechanism for load balancing can be used to alleviate the problem, however note that a peer zone cannot be smaller than a grid cell. In case the grid is designed to follow the data distribution, results for the clustered and real data set would be the same as for the uniform dataset.

6.1 Number of exchanged messages

We investigate the number of exchanged messages in terms of the average number of generated messages within the peer network *per each published data object* to measure the overhead compared to a single message required to deliver the object to a centralized processor. Note that there are 400 active queries in the system and therefore these numbers include all exchanged messages needed for the parallel processing of all queries.

Notify owner peer messages Figure 8 shows the average number of exchanged *notify owner peer* messages per appeared data object in the default simulation scenario. The number of such messages equals the number of reported k-NN/w objects to querying peers, and obviously it grows when increasing k. For example, we report on average around 1 data object per all 400 queries when $k = 81$ for each published

[4] A k-NN/w query is interested in the entire attribute space upon activation until its threshold is set to an initial value. This process can generate a lot of threshold update messages. However, in a real-life system query thresholds can initially be estimated based on data stream characteristics and history. For this reason, we evaluate the system in a typical situation with already set thresholds.

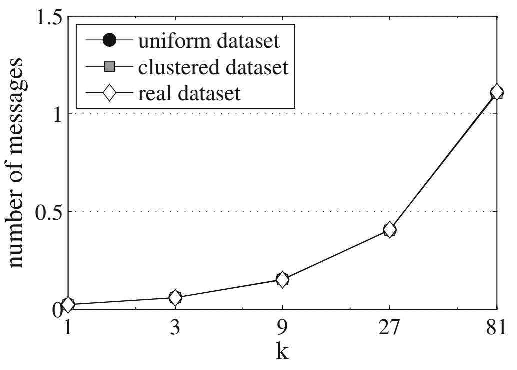


Figure 8 Average number of exchanged *notify owner peer* messages per appeared data object.

object. The number of reported objects is independent of the data set, and this experiment shows our implementation is in line the the proposed model since it performs effective filtering of the input data stream regardless of the dataset. Since PA has very low error rate, the number of generated messages for PA is similar to RAPF and therefore the differences are not visible in Figure 8.

Notify interested peer and merger update messages Figure 9 shows the number of exchanged *notify interested peer* messages per appeared data object for different data sets. The number of such messages depends on the query threshold since a query with a larger threshold has a larger subspace of interest which spans the zones of more peers. For all data sets, when $k > 3$, PA-based implementation generates significantly less messages than RAPF-based implementation. This is expected since PA usually sets lower query thresholds than RAPF, and additionally, there are two object insertion attempts for RAPF per each published data object. The number of messages is the highest for the uniform dataset as query thresholds are the largest. However, all peers generate and consume on average the same number of messages, while in case of clustered and real data set the message distribution within the peer network is unbalanced.

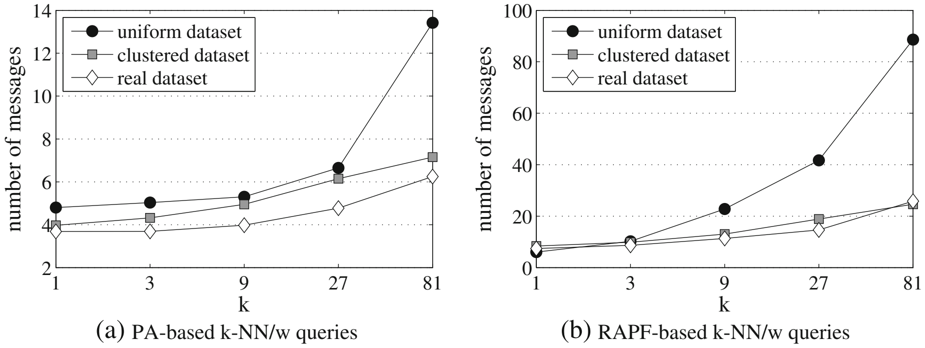


Figure 9 Average number of exchanged *notify interested peer* messages per appeared data object.

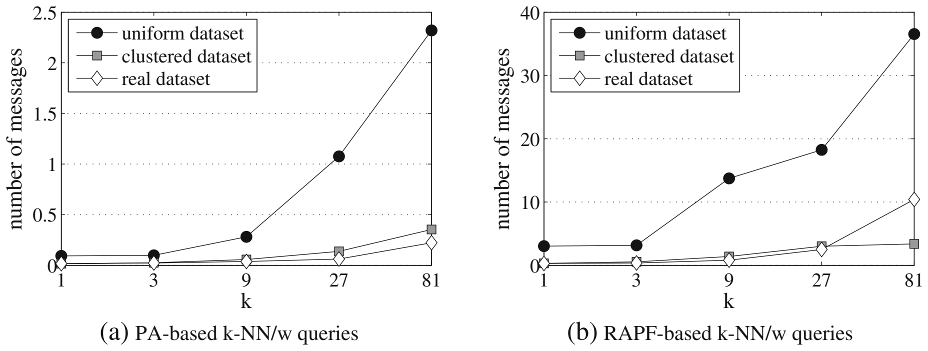


Figure 10 Average number of exchanged *merger update messages* per appeared data object.

Figure 10 shows the number of exchanged *merger update messages* per appeared data object for different data sets. As the number of such messages depends on the frequency of query threshold changes, PA-based implementation generates significantly less merger update messages than RAPF-based implementation because PA thresholds change less frequently. As for the notify interested peer messages, the number of merger update messages is the highest for the uniform dataset. If we closely analyze the number of generated messages per processed object, it turns out that notify interested peer messages represent the larges overhead, however the number of such messages is quite low for PA-based implementation.

Total number of generated messages Figure 11 shows the average number of all generated messages per appeared data object during the default simulation scenario for different data sets. The number of messages obviously grows for increasing k as more stream objects are reported as k-NN/w objects. PA-based implementation generates up to six times less messages than RAPF-based implementation. In particular, it grows up to 21 messages which is quite low compared to 256 messages that would be needed in case all published data objects are flooded to all peers.

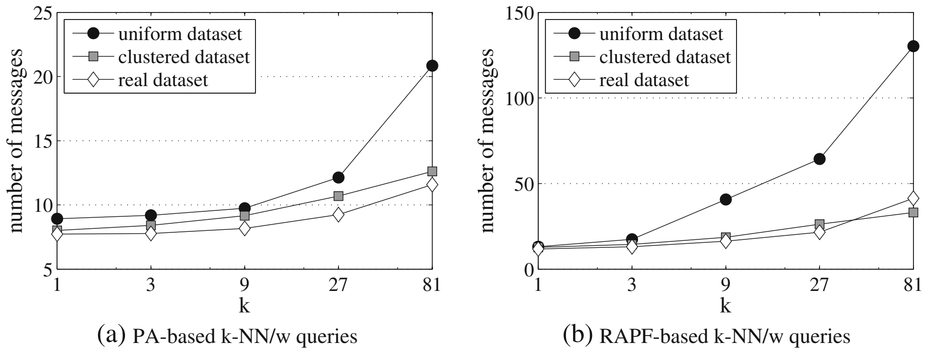


Figure 11 Average number of all exchanged messages per appeared data object.

6.2 Comparison with the centralized k-NN/w processing engine

We compare the average number of all exchanged messages in a distributed system to the number of messages needed to transport all objects and queries to a single processing node. Since *notify interested peer* and *merger update* messages do not appear in the centralized setup, the total number of exchanged messages in the centralized case is the sum of the *query activation*, *notify rendezvous peer* and *notify owner peer* messages.

Figure 12 shows the percent increase of the number of exchanged messages in the distributed system compared to centralized implementation. We observe that distributed D-ZaLaPS using PA generates around 5–10 times more messages than its analogous centralized version. However, since the complete processing happens at a single node in the centralized case, the average peer load in the distributed case is $C = 256$ times lower than the load of the centralized peer for the uniform data set because the processing load is uniformly distributed among available peers. Additionally, although the total number of exchanged messages is lower in the centralized case, the average network traffic per peer is $\frac{C}{10} \approx 25$ to $\frac{C}{5} \approx 50$ times lower in the distributed case. Similarly, distributed D-ZaLaPS with RAPF-based implementation generates around 10–60 times more messages than a centralized version which results in $\frac{C}{60} \approx 4$ to $\frac{C}{10} \approx 25$ times lower average traffic load per peer. One may argue the increase of the total number of messages is significant for the RAPF-based implementation, especially for large k, and therefore PA-based solution would be offered as more economical with significantly lower processing and message exchange load over processing nodes.

Furthermore, we can also see that for PA-based queries, the shown percent increase decreases with increasing parameter k in the case of the clustered and real dataset. This happens because in the distributed case, the number of notify owner peer messages increases linearly with parameter k, as shown in Figure 8, while the number of other types of messages increases sub-linearly with parameter k. However, in the centralized case, the number of all messages grows linearly since the number of *query activation* and *notify owner peer* messages is constant for different values of parameter k. For this reason, the rate at which the total number of messages increases with parameter k is higher for the centralized case than for the distributed case. The

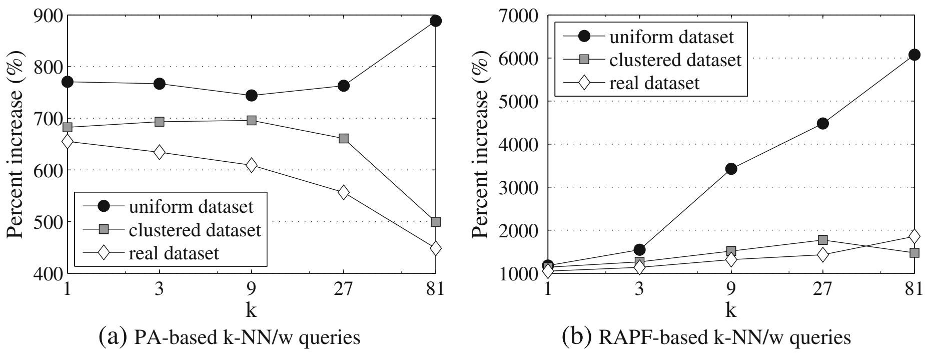
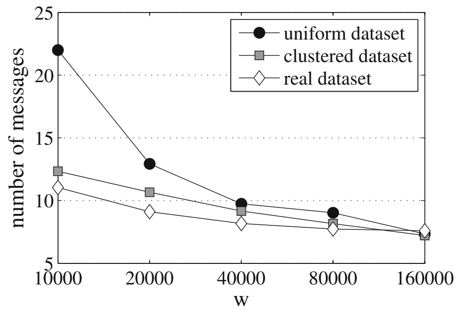
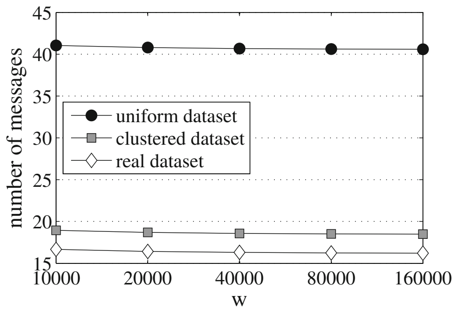


Figure 12 Percent increase in the number of exchanged messages when compared to the centralized k-NN/w processing system.



(a) PA-based k-NN/w queries



(b) RAPF-based k-NN/w queries

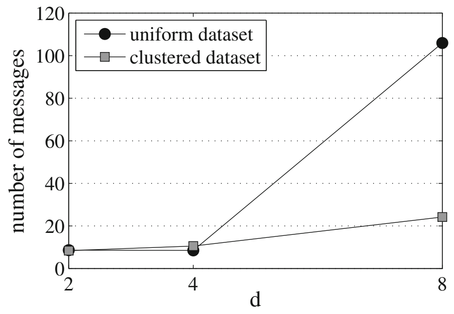
Figure 13 Average number of all exchanged messages per appeared data object for varying query window size.

sudden drop of the percent increase for $k = 81$ in the case of clustered dataset and RAPF-based queries can be explained analogously.

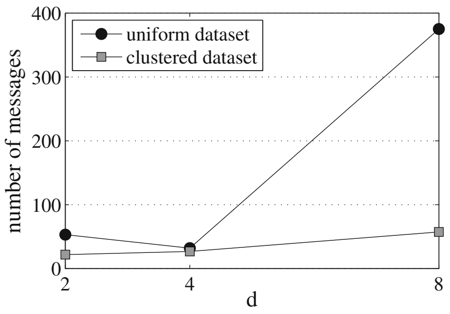
6.3 Varying query window size and data dimensionality

Figure 13 shows the average number of all exchanged messages per appeared data object for varying query window sizes. In the case of PA-based k-NN/w implementation, the number of messages decreases with increasing window size because less objects are dropped from a larger query window which then results in both lower and less dynamic query thresholds. The direct consequence is a lower number of exchanged *notify interested peer* and *merger update* messages. For RAPF-based implementation, the number of exchanged messages does not depend on query window sizes since query thresholds are influenced by the recent buffer size, and not by the query window size.

Figure 14 shows the average number of all exchanged messages per appeared data object while varying data dimensionality. We show only the results for the uniform and clustered dataset since the dimensionality of the real dataset is lower



(a) PA-based k-NN/w queries



(b) RAPF-based k-NN/w queries

Figure 14 Average number of all exchanged messages per appeared data object for varying data dimensionality.

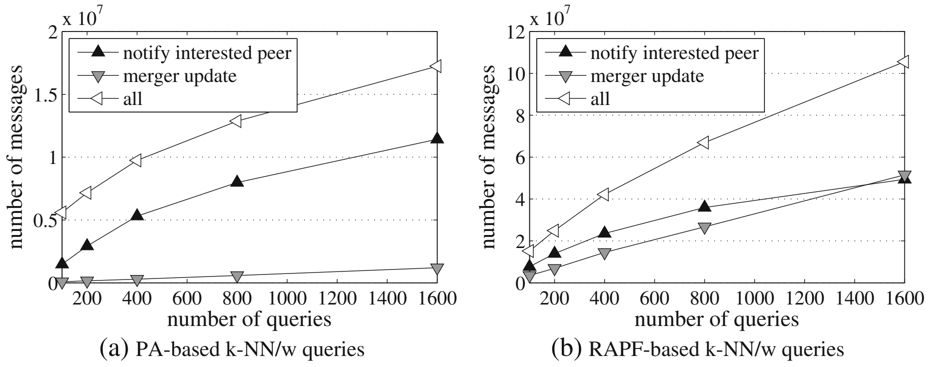


Figure 15 Total number of exchanged messages for different number of queries.

than 8. In this experiment we keep the number of grid cells $g = 2^{16}$ constant for different data dimensionality. For this reason, the grid resolution ρ is varying with dimensionality, i.e. it was 256, 16 and 4 in the case of 2, 4 and 8-dimensional data, respectively. For the uniform dataset, the number of messages for both the PA-based and RAPF-based implementation increases exponentially with the increasing dimensionality. The explanation for such behavior is in the number of peer neighbors (including edge peers) which equals $3^d - 1$. However, in the case of a clustered dataset, we can see that the number of messages for both the PA-based and RAPF-based implementation increases linearly because clustered data is generated with fixed variance for each dimension, which, when increasing the data dimensionality, results in the decreasing proportion of the hyperspace in which data is generated. The combined influence of these two factors produces a linear grow on the number of messages in this case.

6.4 Scalability

We evaluate the scalability of the D-ZaLaPS system when the number of queries and peers increases, respectively. In both experiments we use the default simulation setup producing 10^6 data objects with the uniform dataset.

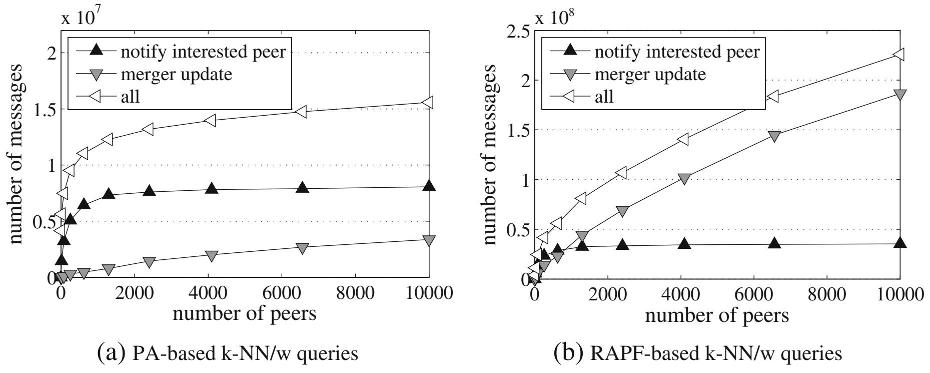


Figure 16 Total number of exchanged messages for different number of peers.

In Figure 15 we can see that the system is scalable since the number of messages grows sub-linearly with the increasing number of queries. However, please note that the number of *merger update* messages grows linearly with the increasing number of queries, while the number of messages is almost an order of magnitude higher for RAPF compared to PA in our simulation setup.

In Figure 16 we see that the system is scalable since the number of messages grows logarithmically for PA-based queries with an increasing number of peers, and sub-linearly for RAPF-based queries. As in the previous experiment, the number of merger update messages scales worse than the number of notify interested peer messages.

7 Conclusions and future work

In this paper we have presented an approach to distributed processing of continuous k-NN queries over sliding windows which combines the distributed nature of publish/subscribe systems with processing efficiency of data stream processing systems. We have shown that our model and corresponding system implementation enable effective filtering of data objects published by distributed data sources. In contrast to existing data stream processing systems for efficient processing of k-NN queries over sliding windows that are centralized, in this paper we offer the first distributed solution built over the CAN overlay. Our experiments using both real and synthetic data sets demonstrate the feasibility of the proposed solution because the message overhead of D-ZaLaPS is acceptable and beneficial in terms of peer processing load. Moreover, the PA-based implementation has significantly lower messaging overhead compared to RAPF-based implementation, and should be applied when network resources are scarce. The proposed system is scalable because the generated network traffic expressed in terms of the number of generated messages grows sub-linearly with increasing number of queries, and logarithmically with increasing number of peers. Therefore, it represents a viable system for effective and efficient filtering of structured data published on the Web at high rates, such as the Sensor Web. It could also be applied to process semi-structured data which can be represented in attribute spaces with low dimensionality, and our future work will be directed to design methods for efficient indexing of queries in such attribute spaces and evaluate them experimentally.

Acknowledgements This work was carried out within the research project "Content Delivery and Mobility of Users and Services in New Generation Networks", supported by the Ministry of Science, Education and Sports of the Republic of Croatia.

References

1. Aberer, K.: P-grid: a self-organizing access structure for P2P information systems. LNCS **2172**, 179–194 (2001)
2. Balazinska, M., Deshpande, A., Franklin, M.J., Gibbons, P.B., Gray, J., Hansen, M., Liebhold, M., Nath, S., Szalay, A., Tao, V.: Data management in the worldwide Sensor Web. IEEE Pervasive Computing **6**(2), 30–40 (2007)

3. Bell, T.A.H., Moffat, A.: The design of a high performance information filtering system. In: SIGIR, pp. 12–20 (1996)
4. Böhm, C., Ooi, B.C., Plant, C., Yan, Y.: Efficiently processing continuous k-nn queries on data streams. In: ICDE, pp. 156–165 (2007)
5. Cao, P., Wang, Z.: Efficient top-k query calculation in distributed networks. In: PODC (2004)
6. Chakrabarti, A., Cormode, G., McGregor, A.: Robust lower bounds for communication and stream computation. In: STOC, pp. 641–650 (2008)
7. Chakrabarti, A., Jayram, T.S., Pǎtraşcu, M.: Tight lower bounds for selection in randomly ordered streams. In: SODA, pp. 720–729 (2008)
8. Conover, H., Berthiau, G., Botts, M., Goodman, H.M., Li, X., Lu, Y., Maskey, M., Regner, K., Zavodsky, B.: Using Sensor Web protocols for environmental data acquisition and management. Ecol. Informa. **5**(1), 32–41 (2010)
9. Das, G., Gunopulos, D., Koudas, N., Sarkas, N.: Ad-hoc top-k query answering for data streams. In: VLDB, pp. 183–194 (2007)
10. Guha, S., McGregor, A.: Approximate quantiles and the order of the stream. In: PODS, pp. 273–279 (2006)
11. Haghani, P., Michel, S., Aberer, K.: The gist of everything new: personalized top-k processing over web 2.0 streams. In: CIKM, pp. 489–498 (2010)
12. Halperin, D.: Arrangements. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, chapter 24, pp. 529–562. CRC Press, Boca Raton (2004)
13. Jin, C., Yi, K., Chen, L., Yu, J.X., Lin, X.: Sliding-window top-k queries on uncertain streams. VLDB J. **19**(3), 411–435 (2010)
14. Koudas, N., Ooi, B.C., Tan, K.L., Zhang, R.: Approximate nn queries on streams with guaranteed error/performance bounds. In: VLDB, pp. 804–815 (2004)
15. Lua, E.K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. IEEE Commun. Surv. Tutor. **7**, 72–93 (2005)
16. Marian, A., Bruno, N., Gravano, L.: Evaluating top-k queries over web-accessible databases. ACM Trans. Database Syst. **29**, 319–362 (2004)
17. Michel, S., Triantafillou, P., Weikum, G.: Klee: a framework for distributed top-k query algorithms. In: VLDB, pp. 637–648 (2005)
18. Mouratidis, K., Bakiras, S., Papadias, D.: Continuous monitoring of top-k queries over sliding windows. In: SIGMOD, pp. 635–646 (2006)
19. Mouratidis, K., Pang, H.: An incremental threshold method for continuous text search queries. In: ICDE, pp. 1187–1190 (2009)
20. Mouratidis, K., Papadias, D.: Continuous nearest neighbor queries over sliding windows. IEEE Trans. Knowl. Data Eng. **19**(6), 789–803 (2007)
21. Munro, J.I., Paterson, M.S.: Selection and sorting with limited storage. In: SFCS, pp. 253–258 (1978)
22. Muthukrishnan, S.: Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci. **1**(2), 117–236 (2005)
23. Neumann, T., Bender, M., Michel, S., Schenkel, R., Triantafillou, P., Weikum, G.: Optimizing distributed top-k queries. LNCS **5175**, 337–349 (2008)
24. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Trans. Database Syst. **30**(1), 41–82 (2005)
25. Pripužić, K., Podnar Žarko, I., Aberer, K.: Top-k/w publish/subscribe: finding k most relevant publications in sliding time window w. In: DEBS, pp. 127–138 (2008)
26. Pripužić, K.: Top-k publish/subscribe matching model based on sliding window. Ph.D. thesis, University of Zagreb (2010). Section 3: Efficient top-k/w processing over data streams
27. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: SIGCOMM, pp. 161–172 (2001)
28. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications. In: SIGCOMM, pp. 149–160 (2001)
29. Terpstra, W.W., Behnel, S., Fiege, L., Zeidler, A., Buchmann, A.P.: A peer-to-peer approach to content-based publish/subscribe. In: DEBS, pp. 1–8 (2003)
30. Tryfonopoulos, C., Idreos, S., Koubarakis, M.: Publish/subscribe functionality in IR environments using structured overlay networks. In: SIGIR, pp. 322–329 (2005)
31. Yu, H., Li, H.G., Wu, P., Agrawal, D., Abbadi, A.E.: Efficient processing of distributed top-*k* queries. LNCS **3588**, 65–74 (2005)

32. Zhang, J., Suel, T.: Efficient query evaluation on large textual collections in a peer-to-peer environment. In: P2P, pp. 225–233 (2005)
33. Zhang, Y.: Computing order statistics over data streams. Ph.D. thesis, University of New South Wales (2008)
34. Zimmer, C., Tryfonopoulos, C., Berberich, K., Koubarakis, M., Weikum, G.: Approximate information filtering in peer-to-peer networks. In: WISE, pp. 6–19 (2008)