The VLDB Journal (2012) 21:359–384
DOI 10.1007/s00778-011-0246-6

REGULAR PAPER

A survey of skyline processing in highly distributed environments

Katja Hose · Akrivi Vlachou

Received: 14 December 2010 / Revised: 20 May 2011 / Accepted: 14 July 2011 / Published online: 11 August 2011

Abstract During the last decades, data management and storage have become increasingly distributed. Advanced query operators, such as skyline queries, are necessary in order to help users to handle the huge amount of available data by identifying a set of interesting data objects. Skyline query processing in highly distributed environments poses inherent challenges and demands and requires non-traditional techniques due to the distribution of content and the lack of global knowledge. This paper surveys this interesting and still evolving research area, so that readers can easily obtain an overview of the state-of-the-art. We outline the objectives and the main principles that any distributed skyline approach has to fulfill, leading to useful guidelines for developing algorithms for distributed skyline processing. We review in detail existing approaches that are applicable for highly distributed environments, clarify the assumptions of each approach, and provide a comparative performance analysis. Moreover, we study the skyline variants each approach supports. Our analysis leads to a taxonomy of existing approaches. Finally, we present interesting research topics on distributed skyline computation that have not yet been explored.

Keywords Skyline processing · Distributed systems · P2P

K. Hose (✉)
Max-Planck Institute for Informatics,
Saarbrücken, Germany
e-mail: hose@mpi-inf.mpg.de

A. Vlachou
Department of Computer Science,
NTNU, Trondheim, Norway
e-mail: vlachou@idi.ntnu.no

1 Introduction

Already before the introduction of skyline queries into database research, the problem was known as the maximum vector problem or the Pareto optimum [24,32]. In recent years, skyline query processing has become an important issue in database research. The popularity of the skyline operator is mainly due to its applicability for decision making applications. In such applications, the database tuples are represented as a set of multidimensional data points, and the skyline set contains those points that are the best trade-offs between the different dimensions.

For example, consider a database that contains information about hotels. Each tuple of the database is represented as a point in a data space consisting of numerous dimensions. Assume a user is looking for hotels in Miami that are as cheap as possible and as close as possible to the beach. In this case, it is not obvious whether the user would prefer (i) a hotel that is very close to the beach but more expensive than others or (ii) a hotel that is very cheap but farther away from the beach. Furthermore, it is difficult to answer the question of how much cheaper a hotel should be, if it is a little bit farther away from the beach. The skyline query retrieves all hotels for which no other hotel exists that is cheaper and closer to the beach. A skyline query does not require a scoring function defining the relative importance of all criteria. Figure 1 shows an example, where each point represents a hotel with price per night and distance to the beach as coordinates; hotels a, i, m, and k are in the skyline set.

In addition to a hotel booking scenario, further skyline applications have been identified; one of them is applying skyline queries in the context of electronic marketing places, for instance for buying cars, where users usually have multiple criteria to rank the offers, e.g., minimum price, minimum age, and maximum high speed. Similarly, skyline

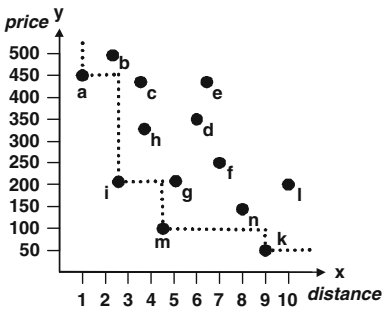


Fig. 1 Skyline example

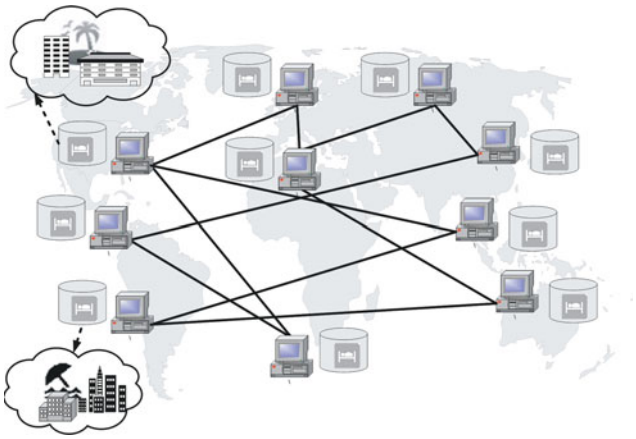


Fig. 2 Distributed hotel reservation system

queries can also be useful for users browsing through a real estate database for houses, where users may want to minimize the price and maximize the quality of neighboring schools.

Skyline processing was first studied in a single-database environment, i.e., in a centralized setup. As nowadays data are increasingly stored and processed in a distributed way, skyline processing over distributed data has attracted much attention recently. Consider a global-scale web-based hotel reservation system, consisting of a large set of independent servers geographically dispersed around the world (Fig. 2). Servers accept subscriptions from travel agencies, in order to advertise their hotels. Each server may provide offers for hotels all over the world, e.g., servers in Paris, Sydney, and Los Angeles might provide different offers for hotels in Miami. Such a system could potentially provide booking services over the universal hotel database, without requiring from each travel agency to register with each server. This need becomes even more important, due to the fact that the number of providers (and therefore data) increases at tremendous rates. The challenge is to enable users to pose interesting queries, such as skyline queries, over a network of servers and retrieve only those tuples that match the user-defined query.

In this paper, we give an overview of the existing approaches for skyline query processing in highly distributed environments, where each server stores a fraction of the available data. For example, the peer-to-peer (P2P) paradigm emerges as a powerful model for organizing and searching large data repositories distributed over independent sources. It should be noted that even though the majority of the papers have been proposed for P2P architectures, several principles are applicable in other distributed systems where the data are distributed over autonomous servers such as grid systems, large-scale data centers, or cloud computing infrastructures. For example, [50] exploits the applicability of distributed indexing techniques proposed for P2P systems for cloud infrastructures. In fact, some of the approaches [8,13,35,36,56] described in this survey were proposed for a more general distributed architecture than P2P and therefore they can easily be adapted to other distributed systems. Moreover, although this paper focuses on highly distributed environments, we also present briefly skyline query processing in other distributed systems (web information systems and wireless sensor networks) as well as different data types (stream data and uncertain data).

While Sect. 2 gives an introduction to the basics of skyline processing and skyline variants, Sect. 3 introduces the main concepts of P2P systems. Readers with background knowledge in skyline queries and P2P systems can simply skip these two sections. Section 4 presents the main concepts of distributed skyline processing, whereas Sect. 5 gives an overview of existing approaches for skyline processing in highly distributed environments. Section 6 provides a comparative performance analysis. In Sect. 7 we elaborate on the skyline variant each approach was proposed for and classify the existing approaches based on the skyline variant they support. Then, Sect. 8 proposes a taxonomy based on the main principles employed by each approach. In Sect. 9, we briefly present approaches for other distributed architectures and explain why their techniques differ from those that are applicable for skyline processing in highly distributed systems. Finally, Sect. 10 comments on open issues, and Sect. 11 concludes the paper.

2 Skyline query processing

Before going into details on distributed skyline processing, this section first provides some fundamental definitions and then presents an overview of the evolution of skyline queries.

2.1 Skyline queries and their variants

The skyline operator was first introduced in [3], where the authors extended SQL's SELECT statement by an optional *SKYLINE OF*, such that the user can specify the dimensions as well as the function (MIN, MAX, DIFF) used for the skyline query. For example (Fig. 1), a user who is looking

for a cheap hotel that is close to the beach would express the skyline query in SQL as: *SELECT * FROM Hotels SKYLINE OF distance MIN, price MIN*.

More formally, given a data space D defined by a set of d dimensions $\{d_1, \ldots, d_d\}$ and a dataset S on D with cardinality n, a point $p \in S$ can be represented as $p = \{p_1, \ldots, p_d\}$ where p_i is a value on dimension d_i. Without loss of generality, let us assume that the value p_i in any dimension d_i is greater or equal to zero ($p_i \geq 0$) and that for all dimensions, the minimum values are more preferable.

Definition 1 (*Skyline*) A point $p \in S$ is said to *dominate* another point $q \in S$, denoted as $p \prec q$, if (1) on every dimension $d_i \in D$, $p_i \leq q_i$ and (2) on at least one dimension $d_j \in D$, $p_j < q_j$. The *skyline* is a set of points $SKY(S) \subseteq S$ that are not dominated by any other point. The points in $SKY(S)$ are called skyline points.

The notion of skyline queries can be extended to subspaces, where a subspace skyline query only refers to a user-defined subset of attributes. In our running example, the hotel database may contain various other attributes, such as the number of rooms, the size of the room, and the star rating. Each nonempty subset U of D ($U \subseteq D$) is referred to as a *subspace* of D. The data space D is also referred to as the full space of dataset S. Then, the **subspace skyline** of U is a set of points $SKY_U(S) \subseteq S$ that are not dominated by any other point on subspace U. Consider for example the two-dimensional dataset S depicted in Fig. 1. The skyline points are $SKY(S) = \{a, i, m, k\}$, which are the best possible tradeoffs between price and distance from the beach. On the other hand, for the subspace $U = \{x\}$, the subspace skyline set is $SKY_U(S) = \{a\}$.

Skyline queries have also been studied for the case where constraints exist. Typically, each constraint is expressed as a range along a dimension, and the conjunction of all constraints forms a hyper-rectangle in the d-dimensional attribute space. A **constrained skyline query** returns the skyline set of the subset of the points S' that satisfy the given constraints. For example, for a user, a hotel may be interesting only if the price of the room is in the range of $100–$200. Given this constraint, the skyline set is retrieved from a subset of S that contains all points that satisfy the constraint. In the example of Fig. 1, the constrained skyline points are $\{m, i\}$ with respect to the above-mentioned constraint on the price.

Finally, literature also proposes dynamic skyline queries.

Definition 2 (*Dynamic skyline*) Given a data space D and a dataset S and m dimension functions $f_1, f_2, \ldots, f_m$ such that each function $f_i (1 \leq i \leq m)$ takes as parameters a fraction of the coordinates of the data points, the dynamic skyline query returns the skyline set of S according to the new data space with dimensions defined by $f_1, f_2, \ldots, f_m$.

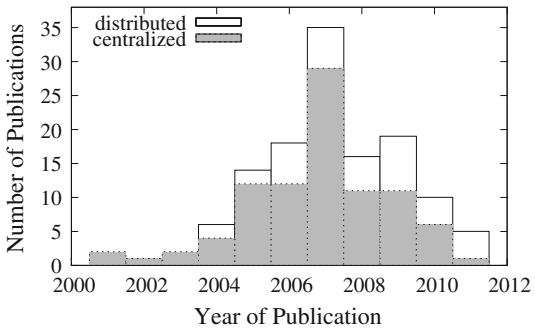


Fig. 3 Publication of skyline papers

For example, consider that each hotel in the database is described by its x and y coordinates, and its price. A user may be interested in minimizing the distance to her/his current location (q_1, q_2) in terms of Euclidean distance and the price of the hotel. Thus, each hotel is described in a two-dimensional space defined by the functions f_1 and f_2 : $f_1(p) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$ and $f_2(p) = p_3$.

2.2 Evolution of skyline queries over time

Since the introduction of skyline queries [3] in 2001, more than a hundred papers have been published. Figure 3 depicts the number of papers per year that are related to skyline computation in centralized and distributed environments published in well-known database conferences or journals. These papers have not only studied efficient skyline computation in centralized or distributed systems but also proposed variations of the traditional skyline operator and studied different premises. Table 1 summarizes the evolution of the skyline queries in database research, while in the following, we describe shortly the related work.

Skyline computation. Börzsönyi et al. [3] first introduced the skyline operator and presented two basic main memory algorithms: BNL (block nested loop) and D&C (Divide & Conquer). The BNL algorithm uses a block nested loop to compare each tuple of the database with every other tuple. A tuple is reported as a result only if it is not dominated by any other tuple. The D&C algorithm recursively divides the set of input tuples into smaller sets (regions), computes the individual skyline for each region separately, and merges them into the final skyline. SFS (Sort-First-Algorithm) [10] and LESS [17] improve the performance of BNL by first sorting tuples according to a monotone function. The main principle of sort-based approaches is that if the tuples are ordered based on a monotone scoring function, then no tuple can be dominated by subsequent tuples.

Skyline query processing with the use of index structures was first proposed by Börzsönyi et al. [3] but elaborated on in later works [23,29,39]. The key idea is to use an index

Table 1 Evolution of skyline queries in database research

	2001	2002	2003	2004	2005	2006	2007	2008
<2001 maximum vector problem	2001 skyline operator [3]	2002 skyline processing with R-tree [23]	2003 skyline variants [29]	2004 skylines in web information systems [2]	2005 subspace skyline queries [31,54]	2006 skylines in P2P systems [18,51]	2007 reverse skyline [14]	2008 parallel skyline computation [45]
	2001 skyline processing with B-tree [39]		2003 skyline processing with sorting [10]		2005 skylines over streams [26]	2006 k-dominant skylines [5]	2007 probabilistic skyline [30]	

to determine dominance between tuples and to prune tuples from further consideration at an early stage. Algorithms using an R-tree were also proposed, namely NN-search (nearest neighbor) [23] and BBS (branch and bound skyline) [29]. These algorithms first compute the nearest neighbor to the origin, which is guaranteed to be part of the skyline result set. Obviously, the region dominated by the nearest neighbor can safely be pruned from consideration. By looking repetitively for the next nearest neighbors in the non-dominated regions, the complete skyline is determined. It was shown that BBS [29] guarantees minimum I/O costs on a dataset indexed by an R-Tree.

Skyline variants and different domains. Papadias et al. [29] first introduced different variants of the skyline operator, such as constrained, subspace, and dynamic skyline queries. Constrained skyline queries were also discussed in [13,51]. Subspace skyline queries were discussed primarily from the view of query semantics in [31]. Then, SKYCUBE [54] was defined as the union of all skyline points of all possible non-empty subspaces. Subspace skyline retrieval was also studied in [42], which proposed the SUBSKY algorithm. Aiming to restrict the skyline cardinality, Chan et al. [5] proposed the k-dominant skyline query. The authors relaxed the idea of dominance to k-dominance, in order to increase the probability of one point dominating another point. Reverse skyline queries have been studied in [14]. Given a query point q, the reverse skyline set contains all points p whose dynamic skyline set contains q. For data point p, the dynamic skyline query employed by the definition of the reverse skyline set uses d dimension functions defined as the absolute difference between the attribute values on each dimension. This corresponds to the skyline set of a transformed data space where p becomes the origin and all other data points are represented by their coordinate-wise distances to point p. Thus, the reverse skyline query retrieves the data objects that are at least in one dimension more similar (in terms of absolute difference of attribute values) to q than all other data objects.

Apart from the efficient computation of the skyline operator and its variants, skyline queries have been studied in different domains, such as probabilistic skyline queries over uncertain data [30], skyline queries on incomplete data [22] and partially ordered domains [4]. Furthermore, efficient skyline computation over streams has first been studied in [26]. Moreover, in [47], bandwidth-constrained skyline queries over mobile devices were studied.

Distributed environments. The main focus of this survey is skyline computation in highly distributed systems, such as P2P systems, where each server stores a fraction of the available data. In the following, we provide a thorough survey of approaches that are applicable for skyline queries in highly distributed systems and summarize their common principles and objectives leading to a generic distributed model. However, skyline queries have also been studied in other distributed environments, such as web information systems [2,28] or parallel shared-nothing architectures [43,45], and with respect to different data types, such as streamed [38] or uncertain data [30]. For completeness, we briefly present other distributed approaches in Sect. 9, we explain why their underlying assumptions are hardly applicable to large-scale distributed systems, and we highlight their differences to our generic model.

3 Peer-to-peer systems

A P2P system corresponds to a distributed computer architecture designed for sharing resources, by direct exchange, rather than requiring a central coordinating server. In P2P systems, the interconnected computers, called *peers*, are organized in a network in a distributed and self-organizing way and share resources, while respecting the autonomy of

peers. This means peers are independent with respect to decisions, such as which fragment of the local data to expose to neighbors, whether to update their local data, or when to leave or join the network. The main characteristic of a P2P system is the ability to adapt to peer failures (fault tolerance) and accommodate a large number of participating peers (scalability), while keeping the performance of the network at an acceptable level and maintaining the peer connectivity.

In general, all peers are equivalent in terms of tasks and functionality they perform. Each peer has a collection of files or data to share. Two peers that maintain an open connection between them are called *neighbors*. The number of neighbors of a node defines its *outdegree* or *degree*. Any peer can issue a query in order to retrieve interesting content; this peer is called *querying peer* or *query initiator*. Query messages are forwarded only between open connections, i.e., between neighboring peers. By issuing a query and sending it to its neighbors, each peer can transparently access in principle all the data in the system. This is achieved by having each peer that receives a query from one of its neighbors forward it again to some of its neighbors to which the query initiator does not have a direct connection. Hence, even data residing on peers located in a distance of several *hops* (number of times a query is forwarded in order to reach its destination) can be queried, without the necessity of deliberate query planning and optimization at the initiator's side. However, the higher the distance (or the number of hops) a query has to travel in order to reach peers with relevant data, the higher are query execution costs in terms of the number of messages and the amount of transferred data between peers.

3.1 Classification of P2P networks

Despite all these common features, P2P systems can be classified into different categories, based on the structure of the P2P system and the degree of centralization.

3.1.1 Structure of P2P systems

P2P systems can be classified into two categories, based on the way the content is located in the network: structured and unstructured P2P systems.

Structured P2P systems. Structured P2P systems are built in a controlled manner and impose a relation between peer content and network topology. A peer that joins the network connects to a well-defined set of peers specified by the structure of the overlay network, e.g., a hypercube (CAN [33]) or a ring (CHORD [37]). The data provided by each peer are redistributed according to a commonly known rule. The rule maps data on peers and also determines which peer is responsible for which fragment of the data. In several well-known structured P2P systems [33,37], the mapping is realized as a distributed hash table (DHT). The main advantage of this approach is that queries can be routed efficiently to peers, by applying the rule according to which the data have been distributed in the first place.

Unstructured P2P systems. In contrast to structured P2P systems, peers in unstructured P2P systems retain a higher degree of autonomy. So they keep data sovereignty, i.e., the data a peer provides remain with the peer and are in general not redistributed to other peers in the system. Furthermore, the network structure solely depends on a peer's choice of neighbors and peers select neighbors arbitrarily. Due to the absence of any relation between the network topology and the stored data objects, peers have only limited information about data objects stored at other peers. Thus, searching may cause peers to query all their neighbors for data objects that match the query—this approach is referred to as *flooding*. To avoid the expensive flooding of the network, peers in unstructured P2P systems often build and maintain routing indexes [11,18] that provide approximate knowledge about what data objects can be accessed by forwarding queries to a specific neighbor (*query routing*). It should be noted that due to the absence of global knowledge, each peer in unstructured P2P systems has to optimize query routing with respect to the locally available information.

3.1.2 Degree of centralization

In principle, the P2P paradigm refers to a completely decentralized system architecture. However, in practice, P2P systems with different degrees of centralization have been developed. Thus, P2P systems can be classified into two categories [1,34] based on the degree of centralization: purely decentralized architectures and hybrid P2P systems. Hybrid systems can further be divided into two main classes: centralized indexing systems and decentralized indexing systems.

In purely decentralized architectures, all computers in the network perform exactly the same tasks, and there is no central coordination of their activities. The main shortcoming of purely decentralized architectures is that the costs of query processing increase with the number of peers.

Hybrid centralized indexing systems use a central server to facilitate interaction between peers and to maintain a centralized index. Napster, for example, uses a centralized index, which is built in cooperation with all participating peers. The centralized index keeps information about the data stored at each peer, together with the peer identifier. Therefore, a single message is required to determine which peer stores relevant information. It should be noted that the actual data exchange between peers is established by direct communication between peers, without interaction with the central server. Despite the efficiency of query processing, centralized indexes have a major drawback, namely they constitute

a "single point of failure". Moreover, the centralized index may become a bottleneck for the system, especially for large networks and during periods with high query rates.

Hybrid decentralized indexing systems, also called super-peer networks [53], harness the merits of both purely decentralized systems and hybrid centralized systems by combining aspects from both approaches. A single point of failure is avoided by introducing more than one peer with special roles (super-peers). Each super-peer has several associated peers and facilitates interaction between peers. If only super-peers are considered, then they form a purely distributed P2P system.

4 Objectives and principles of distributed skyline processing

In this section, we first outline the objectives of a distributed skyline processing approach. Then, we present the main principles of distributed skyline processing, point out the major phases of any distributed skyline algorithm, and therefore provide useful guidelines for the design of efficient distributed skyline algorithms.

4.1 Objectives

The main objective of distributed skyline processing is minimizing query execution time. We refer to the time that passes between the moment the query is issued and the moment until all results have been reported to the user as *execution time*. There are several factors that affect the execution time:

- *Total processing time*: The total processing time is defined as the fraction of the execution time caused by peers evaluating the query locally. Therefore, minimizing the execution time requires minimizing the total processing time. The latter is partially achieved by minimizing the individual processing time at each peer. This is accomplished by utilizing efficient local indexing at each peer and adopting state-of-the-art indexing techniques from centralized settings. Moreover, it is important to exploit parallelism in order to minimize the total processing time. When the query is processed locally in parallel on different peers, the impact of the total processing time on execution time is reduced.
- *Number of queried peers*: The objective is to minimize the number of queried peers by avoiding contacting peers (in vain) that do not contribute to the skyline result set. A peer is said to contribute to the result if either the peer's local data belong to the result set or the peer is part of a path leading to relevant data. The number of queried peers relates indirectly to the number of messages exchanged to process the query. Consequently, it is important for the scalability of the system to have a reduced number of queried peers. Since most of the approaches aim to find the exact and complete result set (the skyline set of all data stored locally at any peer), peers cannot be avoided to be queried unless they do not contribute to the query.
- *Network traffic*: The aim is to minimize the number of data objects transferred, thus reducing the network transfer time, which in turn reduces the execution time. For this purpose, it is preferable to evaluate as many parts of the query as possible locally, rather than transferring the entire dataset to the querying peer.

The main goal of minimizing the execution time comes as a consequence when the aforementioned factors are satisfied. However, as will be shown at the end of this section, the factors are often contradictory and there exists a trade-off between them.

4.2 Main principles

In the following, we identify three main principles that are used in almost every existing approach for distributed skyline processing: (1) the additivity of the skyline operator, (2) pruning of local data objects via filtering, and (3) peer pruning based on local information.

The key property of the skyline operator that is used in distributed skyline processing is the *additivity of the skyline operator*. Given n datasets $S_i (1 \leq i \leq n)$ corresponding to n peers, the skyline points are the same if the skyline operator is evaluated on (i) the union of the n datasets or (ii) first on each set in separate and then once more on the union of the result sets.

Definition 3 (*Additivity of the skyline operator*) Given a dataset S and n datasets S_i such that $S = S_1 \cup \cdots \cup S_n$, the following equation holds: $SKY(S_1 \cup \cdots \cup S_n) = SKY(SKY(S_1) \cup \cdots \cup SKY(S_n))$

Let us consider the example illustrated in Fig. 4. We have two datasets S_1 and S_2 (local data) stored at peers P_1 and P_2. The aim of distributed skyline processing is to retrieve the skyline points of the dataset $S = S_1 \cup S_2$. The additivity of the skyline operator ensures that it is sufficient to take into account only the skyline points of S_1 (points a, h, m, and k) and S_2 (points b, i, and n), also referred to as the local skyline points of peers P_1 and P_2. No other point can be part of the skyline set because it is dominated by at least one point of the peer's local data and because the dominance relation is transitive. In order to determine the overall skyline set (points a, i, m, and k), the skyline points of the union of the local skyline sets are computed, so that dominated points are pruned.

Due to the additivity of the skyline operator, the skyline query can be processed in a distributed fashion, where each peer processes a skyline query based on the data that are

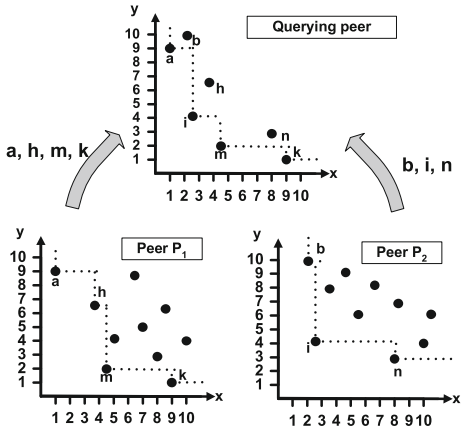


Fig. 4 Skyline additivity

stored locally. Thus, the computational cost of skyline processing on all data points is shared among multiple peers. Furthermore, the additivity also reduces the network traffic because only local skyline points $SKY(S_i)$ have to be transferred instead of the entire local dataset S_i, and because $|SKY(S_i)| \leq |S_i|$. In fact, a peer that receives local result sets from its neighbors may first process a skyline query on the received data and its local result set and create a partial result set. Then, it forwards the partial result set back to the peer the query has been received from. It should be noted that the partial result set is not a subset of the overall skyline set (as it may contain false positives), but it is the result set that corresponds to the data stored in a subset of the network.

The second principle used for efficient distributed skyline processing is *pruning of data points through filtering* (or *filter points*). The main idea is that peers also forward additional information (filtering information) along with the query, that concisely describes the already retrieved local result sets. Commonly, this information is a subset of the local skyline points, called filter points. Filter points are used by the neighboring peer to discard local skyline points that are dominated by them. If the filtering information is selected carefully so that there exists a high probability that local skyline points are discarded, then the transferred data can effectively be reduced through filtering. Also, local processing at a peer can be more efficient through filtering since in some cases a peer can immediately detect that all local points are dominated by the filter points.

Finally, the third principle used for efficient skyline computation in distributed environments is to exploit the dominance relation to exclude peers from further consideration, also called *peer pruning*. In most approaches, each peer uses the local skyline points in order to determine whether a neighboring peer can store data points that are not dominated. In case all the data a neighbor provides are dominated by the local result set, there is no need to query the neighbor, and it can be safely pruned from further consideration. The information about the data stored at each neighbor peer is called *routing information*, which is used for peer pruning. In unstructured P2P networks, a peer can detect whether a neighbor can be pruned by collecting and storing locally some summary information about the data accessible through each neighbor. Then, this routing information is tested for dominance based on the local skyline points. In structured P2P systems, the routing information is essentially the rule according to which the data have been distributed in the first place. The rule together with the network structure yields information that can be used in conjunction with the local result set to prune peers. Furthermore, the local skyline points can be combined with filtering information, so that there exists a higher chance of pruning neighboring peers.

4.3 Phases of distributed skyline processing

In general, query processing in distributed environments adheres to the following phases that are utilized by almost all existing approaches:

	Each peer that poses a query or receives a query from a neighbor:
1.	**[LOCAL PROCESSING]** computes the skyline set based on local data and (optionally) the filtering information received along with the query,
2.	**[QUERY ROUTING]** decides which neighbors can contribute to the skyline set, tries to eliminate neighbors by peer pruning, and forwards the query to the remaining neighbors,
3.	**[RESULT MERGING]** receives the local result sets from queried neighbors and merges all partial results by checking for dominated points. Then, the peer outputs the result to the user or forwards it to the peer the query has been received from.

Almost all distributed skyline processing algorithms contain these three phases. It should be noted that the query routing approach could be executed before the local processing phase. This leads to a non-blocking approach with higher parallelism since the query is propagated to the neighbors without waiting for the local processing to finish. Nevertheless, it is common for distributed skyline processing algorithms to first conduct the local processing, so that filtering and more efficient query routing (peer pruning) are possible. This is because the gain in execution time resulting from the reduction of transferred data and peer pruning is higher than the delay caused by local query processing.

In the *query routing phase*, the query is forwarded to some of the neighboring peers in order to retrieve their partial results. Thus, the peer decides whether a neighbor can

Table 2 Categorization of the distributed approaches based on the P2P overlay

Approach	Overlay	
DSL [51]	Structured	DHT (CAN)
SSP/Skyframe [48,49]	Structured	Tree-based (BATON) and DHT (CAN)
iSky [9,12]	Structured	Tree-based (BATON)
SSW [25]	Structured	Semantic small world networks (SSW)
SFP [20]	Unstructured	Pure P2P
DDS [18,19]	Unstructured	Pure P2P
SKYPEER/SKYPEER + [44,46]	Unstructured	Hybrid (super-peer)
BITPEER [16]	Unstructured	Hybrid (super-peer)
PaDSkyline [8,13]	Unstructured	Fully-connected network topology
AGiDS [35]	Unstructured	Fully-connected network topology
FDS [56]	Unstructured	Fully-connected network topology
SkyPlan [36]	Unstructured	Fully-connected network topology

contribute to the skyline set (relevant peer) based on available routing information. After having identified neighbors relevant to the query, the peer can either forward the query to all these neighbors *in parallel* or query them in a *sequential* manner by waiting for the partial result set of a neighbor before contacting the next. In most cases, query routing to relevant peers is performed in parallel. However, in some scenarios, contacting relevant peers sequentially may be more efficient since each time a partial result set is received, the filtering information can be updated before querying the next relevant peer. Furthermore, based on the partial result set, some of the previously relevant peers can be pruned. Hence, when relevant peers are contacted in a sequential manner, the aim is to contact those peers first that most likely return data points having a high probability to dominate many other points and peers.

In the *merging phase*, the local and the partial result sets have to be collected and a skyline query has to be processed on the union of those sets to discard dominated points. We refer to this process as merging of result sets. During the merging process, only data points that are stored or accessed through different peers have to be tested for dominance. By using this property, result merging is actually more efficient in terms of computational costs than executing a skyline query on the union of the result sets. Furthermore, for the result merging phase, there are two options. Either each peer collects the local result sets of its queried neighbors and merges them by discarding dominated points, or it immediately forwards the received local result sets to the peer the query has been received from without merging. Obviously, merging of the local result set may lead to reducing the transferred data since some dominated points are discarded. On the other hand, merging may increase execution time since each peer has to wait until it collects the local result sets from all queried neighbor peers. It should be noted that in any case, the querying peer has to merge the local and partial result sets before returning the skyline points to the user.

5 Approaches for skyline processing in P2P environments

In this section, we describe all existing approaches dealing with the problem of processing skyline queries in highly distributed environments. We divided the approaches into two groups. The first group consists of approaches designed for structured P2P systems, whereas the second group consists of approaches suitable for unstructured P2P systems. Table 2 summarizes this categorization and provides more specific details on the underlying P2P networks. Recall that in our descriptions, unless mentioned otherwise, we assume that minimum values are preferable.

5.1 Structured P2P

We have already introduced the main characteristics of structured P2P networks in Sect. 3. During distributed skyline computation in structured P2P networks, the existing overlay network is exploited for query routing, while filter points are used for improving the overall query processing performance. In the following, we present the existing approaches in detail and elaborate on characteristics of each overlay network that are used for skyline computation in these networks. Therefore, this section covers all details necessary to understand how skyline computation exploiting overlay-specific details works.

5.1.1 DSL [51]

Wu et al. [51] propose DSL (Distributed Skyline) to compute constrained skyline queries in CAN networks [33]. DSL uses the CAN overlay to map data to regions and assign these regions to peers (content-based data partitioning). Given a constrained skyline query, peers that do not contribute data

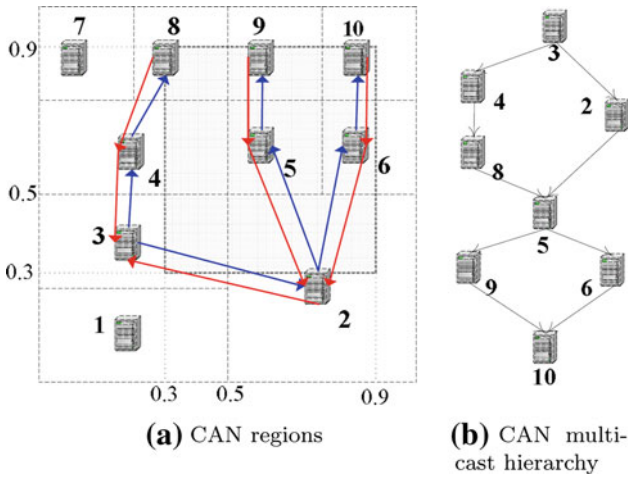


Fig. 5 DSL [51]: the queried range is ((0.3,0.3),(0.9,0.9)); the peer guaranteed to hold data that are part of the global skyline set is peer 3, which therefore serves as the root of the multicast hierarchy that connects peers responsible for neighboring regions in the CAN overlay

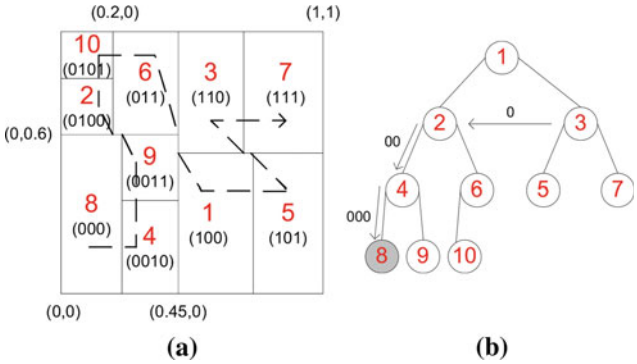


Fig. 6 Skyframe example [49]: **a** peers and their assigned regions, **b** routing in BATON overlays—peer 3 retrieves data assigned to peer 8

complying with the constraints can be immediately pruned, and an ad hoc multicast tree that connects all remaining peers is built. By organizing the peers in a multi-level hierarchy and using it to propagate queries and local result sets, each queried peer computes the skyline set on its local data, and intermediate peers merge result sets from their children. Figure 5 shows an example of a hierarchy built at runtime. The query is propagated along the edges of the hierarchy; peers perform local computation and propagate the results back on the same paths—on the way back, results are aggregated exploiting skyline additivity.

During query processing, DSL builds a multicast hierarchy in which the peer that is responsible for the region containing the lower left corner of the constraint is the root. The data points stored at this peer are guaranteed to belong to the global skyline set as these data points cannot be dominated by points stored at any other peer. Furthermore, the hierarchy is built in such a way that only peers whose data points cannot dominate each other are queried in parallel. In general, any peer in the CAN overlay can decide whether its local skyline points are in the global skyline set by only consulting a subset of other peers. Therefore, a partial order between the peers is defined which captures these computational dependencies between the local result sets. In DSL, the hierarchy is built dynamically and each queried peer decides which neighboring peers should be queried next by using dynamic region partitioning and encoding. Thus, a peer that receives a query along with the local result set first waits to receive the local skyline sets from all neighboring peers that precede it in the hierarchy. Then, it computes the skyline set based on its local data and the received data points. Thereafter, the local skyline points are forwarded to the peers responsible for neighboring regions, in such a way that only peers whose data points cannot dominate each other are queried in parallel. In addition, neighboring peers that are dominated by the local skyline points are not queried because they cannot contribute to the global skyline set. Finally, all local result sets are collected at a peer that cannot forward the query any further, and the global skyline set is reported back to the query initiator.

5.1.2 SSP and skyframe [48,49]

Wang et al. [48] present an approach (Skyline Space Partitioning, SSP in short) for distributed processing of skyline queries in BATON [21] networks. Peers in BATON are organized in a balanced binary tree structured overlay network, where each peer is responsible for a region in data space. Techniques for splitting and merging allow for load balancing among peers. By dynamically sampling load from random peers, load imbalance can be detected and data may be migrated to other peers in order to counteract the imbalance.

As BATON networks have originally been designed for one-dimensional data, Wang et al. map the multidimensional data space to one-dimensional keys using a Z-curve. Figure 6 shows an example of the mapping of the data regions to the peers in the BATON network. Regions in BATON are created by successively splitting existing regions into two parts with respect to a specific dimension, and each region is represented by a binary string that identifies the region and is consistent with the Z-order of the region. Each peer knows the split history (i.e., dimension, split value) that its region originates from. Based on this information, for a given region, the identifier is determined. In addition, the routing table of each peer contains links to other peers (parents, children, adjacent peers, and other peers on the same level), so that the query is efficiently routed to a particular peer.

Figure 6b illustrates the principle of routing in BATON overlays. Assume peer 3 needs to retrieve data contained in the region assigned to peer 8. The identifier of the responsible region starts with 0 because it is contained in the lower part of the first split (at 0.45 in the x dimension). Thus, peer

3 forwards the query to a known peer (peer 2) with a leading 0 in its assigned region. Using the same strategy, peer 2 forwards the query to peer 4, which again forwards the query to the peer holding the queried data, namely peer 8.

Skyline processing in BATON networks relies on identifying relevant regions and routing the query to peers responsible for those regions. More precisely, skyline computation starts at the peer p_{start}, which is the peer responsible for the region containing the origin of the data space. Peer p_{start} computes the local skyline points that are guaranteed to be in the global skyline set. Then, p_{start} selects the most dominating point p_{md} (i.e., the point dominating the largest region [20]), which is used to refine the search space and to prune dominated regions and therefore also the responsible peers from consideration. A peer can safely be pruned if the best point (i.e., the lower left corner) of its region is dominated by p_{md}. Then, the querying peer forwards the query to the peers that are not pruned and gathers their local skyline sets. Finally, the query initiator computes the global skyline set by discarding dominated local skyline points.

In [49], Wang et al. generalize SSP by proposing Skyframe. In more detail, Wang et al. [49] propose an alternative algorithm for skyline processing without the need to determine a peer p_{start} before query processing starts. Instead, the querying peer forwards the query to a set of peers called border peers. A peer that is responsible for a region with minimum value in at least one dimension is called border peer. In our example, peers 1, 2, 4, 5, 8, and 10 are the border peers. Once the initiator receives the local skyline results, it computes p_{md} and determines whether additional peers need to be queried. Then, the querying peer queries additional peers, if necessary, and gathers the local skyline results. When no further peers need to be queried, the query initiator computes the global skyline set. Wang et al. show in [49] that Skyframe is also applicable for CAN networks.

5.1.3 iSky [9,12]

Chen et al. [9] propose the iSky algorithm for skyline computation on structured peer-to-peer networks. Similar to Skyframe [48,49], iSky relies on the BATON [21] overlay but employs a different transformation, namely iMinMax, to assign data to BATON peers. Each multidimensional tuple is mapped to a one-dimensional value (iMinMax value) by first determining the maximum value for this tuple among all dimensions. Assuming that the range of each dimension is normalized into [0,1), the iMinMax value is defined as the sum of (i) this maximum value and (ii) the number of the dimension it originates from. Each peer is responsible for a specific non-overlapping range of iMinMax values, so that each data point is assigned to a specific peer. Figure 7 shows an example of a BATON network and also for each peer the range of iMinMax values which each peer is responsible for.

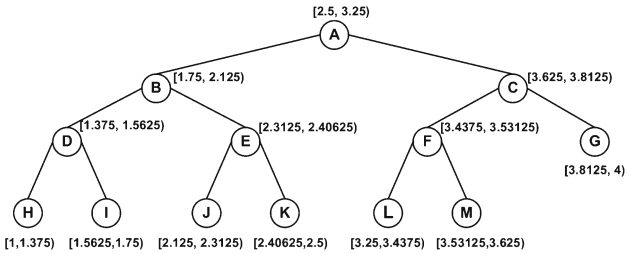


Fig. 7 iSky example [9]: BATON network and iMinMax ranges assigned to peers

It should be noted that iSky assumes for each dimension, larger values are preferable.

Given a skyline query, iSky first determines a set of initial skyline peers, which consists of these peers storing data points with maximum value in any dimension. These peers are chosen because points with maximum value in some dimension are guaranteed to be part of the global skyline set. Because of the iMinMax transformation, which has been used to distribute the data in the network, the initial skyline peers are responsible for a data range close to an integer, e.g., a peer responsible for the range [1.75,2.125). Thereafter, the initial skyline peers are queried, and their local skyline results are merged into an initial set of skyline points by discarding dominated local skyline points. Then, the querying peer determines a threshold and a filter point. In order to define the threshold, the minimum value of all dimensions for each initial skyline point is computed. Then, the maximum value of all minimum values of all initial skyline points is selected as a threshold by using its range of iMinMax values. Any data point can be pruned if its maximum attribute value in all dimensions is smaller or equal to the received threshold. In addition, the most dominating point (i.e., the point dominating the largest region [20]) is chosen as filter point. The threshold and the filter point are attached to the query, which is forwarded to any neighboring peer that stores interval values larger than the threshold. When a peer receives a query, then it first uses the threshold to check whether all its data are pruned. In this case, the peer just forwards the query to its neighboring peers that have interval values larger than the threshold. Otherwise, the peer processes the query on its local data and uses the filter point to discard dominated tuples. Finally, each peer refines the threshold and the filter point before forwarding the query and immediately sends the local result set to the query initiator, which then merges the local results and obtains the global skyline set after all queried peers have processed the query.

5.1.4 SSW [25]

Li et al. [25] use a space partitioning method that is based on an underlying semantic overlay network (Semantic Small

World—SSW). In such a network, the multidimensional data space is partitioned into non-overlapping regions, which are mentioned also as clusters. Assuming that each peer stores data that form one or more clusters, peers are assigned to the non-overlapping regions based on semantic labels. The semantic label of a peer corresponds to the region, which contains the centroid of its largest data cluster. For the data not contained in the region corresponding to the semantic label of a peer, foreign indexes are created at peers whose semantic label covers the data. A peer's foreign index holds information about data contained in its region that is stored at remote peers assigned to other regions. Each peer maintains links to other peers assigned to the same cluster and links to at least one peer in each neighboring cluster.

The computation of a skyline starts from the region that is guaranteed to contain skyline points, i.e., the region containing the origin. Then, the skyline query is evaluated over the data provided by peers in this cluster. The local skyline point that corresponds to the nearest neighbor of the origin is used as a filter point. In more detail, all regions that are entirely dominated by the filter point are not considered for further processing. After querying the remaining regions, reporting back all local result sets to the query initiator, and checking for mutual dominance, the global skyline set is returned to the user.

Apart from this algorithm, Li et al. [25] also propose an approximate algorithm, which does not require a semantic overlay network. Still, peers have semantic labels that are used to process the skyline query. As a peer is assumed to know the semantic labels of its neighbors, it forwards the query to the neighbor with the best semantic label, which is defined as the semantic label closer to the origin. Once a peer cannot find a neighbor with a better semantic label than its own semantic label, skyline computation ends and the result is returned to the user. Another variant of this strategy (multi-path) forwards the query for each dimension to the peer that provides the best data, if only this dimension is considered. The peer that initiates the query coordinates the computation and might optionally issue a stop command when a certain number of peers have been queried.

5.2 Unstructured P2P

In contrast to skyline query processing in structured P2P networks, where the overlay is used to locate peers that store relevant data, in unstructured P2P networks all peers have to be queried or routing indexes have to be constructed. It should be noted that some approaches assume the query originator has a direct connection to all other peers (fully connected network topology). In fact, these approaches assume a more general distributed architecture than P2P and therefore they can easily be adapted to other distributed systems.

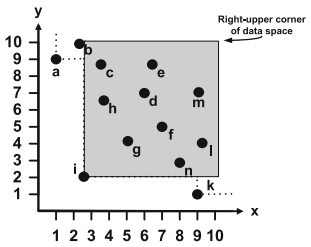


Fig. 8 SFP example [20]: dominating region of point *i*

5.2.1 *Single filtering point (SFP) [20]*

Huang et al. [20] assume a setting with mobile devices communicating via an ad hoc network (MANETs) and study skyline queries that involve spatial constraints. Even though Huang et al. focus on a mobile environment, there are several similarities between ad hoc networks and unstructured P2P networks. The technique (single filtering point—SFP) used to reduce the transferred data is directly applicable to P2P networks. Thus, we analyze this technique and omit any details related to spatial constraints and optimizations of query processing on mobile devices presented in [20].

The main feature of SFP is the usage of a point that belongs to the local skyline set as a filter to discard local skyline points of other peers. The selection of the filter point is based on the volume of the dominating region (Fig. 8). The dominating region is defined as the area in data space that is dominated by a skyline point. Assuming a uniform distribution, a larger dominating region means a higher probability to dominate other points. When a peer receives a query request, it processes the query first locally and then propagates the query to its neighboring peers by attaching a filter point to the query. The filter point is used to discard local skyline points before sending back the local result to the query initiator. Each peer updates the filter point if a local skyline point has a larger dominating region.

5.2.2 *DDS [18,19]*

Hose et al. [18,19] propose an approach for skyline processing in unstructured P2P networks that uses routing indexes to identify relevant peers. Routing indexes, or distributed data summaries (DDS) respectively, are summaries of the data accessible via a peer's neighbors. Thus, there is one summary for each neighbor, summarizing not only its local data but also the data of peers that are located several hops away but reachable via the neighbor. Each peer in the network is assumed to hold such summaries for its neighbors. Hose et al. consider two variants of data summaries, one based on multidimensional histograms and the other one based on the QTree, a combination of R-trees and multidimensional histograms.

Processing skyline queries based on DDS in this scenario works as follows: First, the query initiator computes the skyline set based on its local data. Then, it decides, based on its data summaries, on the relevance of its neighbors. The main idea is to prune all neighbors that provide only data that are dominated by local skyline points. A data summary based on histograms can be regarded as a set of regions (corresponding to buckets) represented by rectangles. Given a skyline query and a rectangular region, then the *best point* that might be contained with respect to domination is the rectangle's lower left corner. The best point dominates all data points possibly contained in the region. If the best point is dominated by a local skyline point, then the region can be pruned. If all regions that summarize data of a specific neighbor are pruned, then the query is not forwarded to this neighbor. Otherwise, the query is forwarded and the local skyline points are also forwarded to the neighboring peer in order to prune its neighbors. To minimize load at the initiator, local skyline points are routed on the same path the query was propagated on; a peer merges the local result sets received from its neighbors with its own local skyline set, checks for mutual dominance, and sends the obtained result to the peer that it received the query from.

DDS also supports approximate skyline queries mentioned as relaxed skylines. A relaxed skyline query aims at reducing the computational costs of skyline processing by representing regions of a peer's data by a single local skyline point. A region describing the data of a neighboring peer is represented by only one local skyline point if any point of the region has a distance to the representative point less than a given threshold, so a neighbor is pruned if all the regions describing its data are either dominated or represented by representatives, i.e., local skyline points. Thus, the query result set does not contain all skyline points, but a subset of skyline points and additionally representative data points that represent regions that are nearby and possibly contain further skyline points.

5.2.3 SKYPEER [44] and SKYPEER+ [46]

Vlachou et al. [44] proposed SKYPEER, a distributed framework for efficient computation of subspace skyline processing over a super-peer architecture. To this end, the notion of domination is extended by defining the extended skyline set, which contains all data points that are sufficient to answer a skyline query in any arbitrary subspace. In a preprocessing phase, each super-peer computes and stores the extended skyline set of its associated peers. Then, when a super-peer receives a subspace skyline query, SKYPEER propagates the query to all super-peers and gathers the local skyline sets. Moreover, SKYPEER utilizes an efficient thresholding scheme that facilitates pruning of dominated data across the peers. In order to support threshold-based query processing,

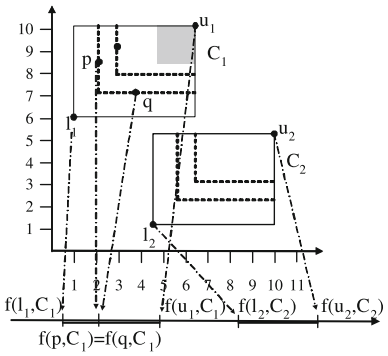


Fig. 9 SKYPEER+ example [46]:Indexing of extended skyline points with one-dimensional mapping

data are transformed into one-dimensional values. Then, during query processing, a threshold value is defined based on already computed subspace skyline points. The threshold is attached to the query before it is propagated in the network. Vlachou et al. explore different strategies for (i) threshold propagation and (ii) result merging over the P2P network aiming to reduce both computational time and volume of transmitted data. As far as threshold propagation is concerned, two strategies are examined, namely fixed threshold (the query initiator sets the threshold) and refined threshold (each super-peer updates the threshold based on its local result). Similarly, for result merging, two strategies are examined, namely merging the local result sets by the query initiator or progressive merging by intermediate super-peers.

SKYPEER was extended in [46] leading to SKYPEER+, which focuses on efficient routing of skyline queries over a super-peer network with the aim of reducing the number of contacted super-peers. Instead of flooding the network, a routing mechanism is established in order to contact only those super-peers that may contribute to the global skyline set. More precisely, in the preprocessing phase, each super-peer additionally applies a clustering algorithm on its locally stored extended skyline set. Then, the extended skyline set is stored based on the one-dimensional mapping, as depicted in Fig. 9. The clusters are represented by MBRs and each point is mapped to a one-dimensional value, while all points that belong to the same dashed line have the same one-dimensional value. Then, based on the threshold employed by SKYPEER+, point p prunes the shadowed area. The cluster descriptions are broadcast over the super-peer network. Each super-peer collects the cluster information of all super-peers and builds routing indexes based on them. The one-dimensional mapping is combined with the clustering information, and a novel indexing technique is proposed for building the routing indexes, which support efficiently the thresholding scheme of SKYPEER. During query processing, the routing indexes are used to propagate the query only to network paths with super-peers storing data points that

may contribute to the skyline set. In addition, the routing information is used to refine the threshold. Therefore, SKYPEER+ further improves the thresholding scheme and drastically reduces the amount of transferred data.

5.2.4 BITPEER [16]

Fotiadou et al. [16] proposed BITPEER for subspace skyline queries over a super-peer architecture. Similar to SKYPEER, each super-peer stores the extended skyline of its peers. Differently to SKYPEER that was proposed independently of the skyline algorithm at each (super-)peer, Fotiadou et al. [16] focus on distributed skyline computation based on BITMAP [39]. Therefore, BITPEER uses a bitmap representation that summarizes all extended skyline points. Given a subspace skyline query, the query is flooded in the super-peer network, and local results are sent back to the querying super-peer by using progressive merging at intermediate super-peers. The authors also discuss caching of subspace skyline points and continuous skyline queries. In order to enable the reusability of subspace skyline results, during query processing, the querying super-peer gathers the extended subspace skyline [44] instead of the subspace skyline. Therefore, BITPEER is able to use a cached skyline set also for subspace skyline queries that refer to a subspace of the query in the cache.

5.2.5 PaDSkyline [8,13]

Cui et al. [13] study skyline query processing in a distributed environment, where the querying peer can directly communicate with all peers. The proposed algorithm is called PaDSkyline (*Pa*rallel *D*istributed *Skyline* query processing), and the main principle is to determine which peers can process the query in parallel under the assumption that the data points of each peer lie only in a part of the data space.

The querying peer first gathers a set of minimum bounding regions (MBRs) from each peer that summarizes the data stored at each peer. Subsequently, the querying peer processes the collected MBRs and groups them into one or more incomparable groups such that any data point summarized by an MBR of one group cannot be dominated or dominate any data point captured by an MBR of another group. Figure 10a depicts a set of MBRs gathered by the querying peer. These MBRs form two incomparable groups, namely MBRs m_1 and m_2 form the first group, while the second group consists of the remaining MBRs. Incomparable groups can be queried in parallel without requiring merging of local results. For each incomparable group, a specific plan is constructed which aims at maximizing the gain achieved by the filter points and defines a beneficial order to query the peers. An example of an execution order is depicted in Fig. 10b. In general, dominated MBRs are discarded, while MBRs (for example m_6) that are partially dominated (for example by m_5) are executed after the partially dominating MBR. For each group, the plan is sent to the peer (head group) responsible for the head MBR of the plan. The peer that receives a plan processes the query locally. Once a peer has processed the query, it removes itself from the query plan and forwards the query to the next peer indicated by the plan. In order to reduce network traffic, each peer attaches a set of K filter points to the query for discarding local skyline points of the peers that belong to the same group. The goal is to select as filter points the local skyline points that are more likely to dominate many other points. Two different strategies are studied. The first strategy is to select the K points with the largest volume of their dominating region [20]. The proposed alternative is to pick the K points with the maximal distance between them. The aim of this strategy is to minimize the overlap between the dominating regions of different points. After the peer has processed the query locally, the results are sent back directly to the head group, and after discarding all dominated points, the results are sent back to the querying peer.

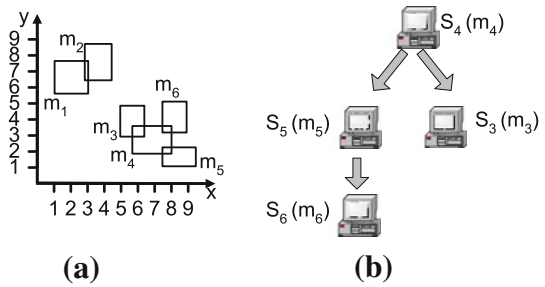


Fig. 10 PaDSkyline example [13]: **a** incomparable groups of MBRs **b** order of executing the query on different peers

5.2.6 AGiDS [35]

Rocha-Junior et al. [35] propose *a gri*d-based approach for *d*istributed *s*kyline processing (AGiDS), which shares assumptions similar to [13]. Differently, AGiDS assumes that each peer maintains a grid-based data summary structure for describing its data distribution.

AGiDS assumes that all peers share common cell boundaries for the grid structure that leads to non-overlapping cells, which increases the probability of domination between cells and enables efficient merging of local skyline set. The set of cells of a peer that contain at least one data point and that are not dominated by other cells is called region-skyline set of the peer. Only these cells of the grid contain data that belong to the local skyline set. At query time, the query initiator first contacts all peers and gathers the region-skyline sets of all peers. Then, the query initiator merges the collected cells into a new region-skyline set by discarding dominated cells.

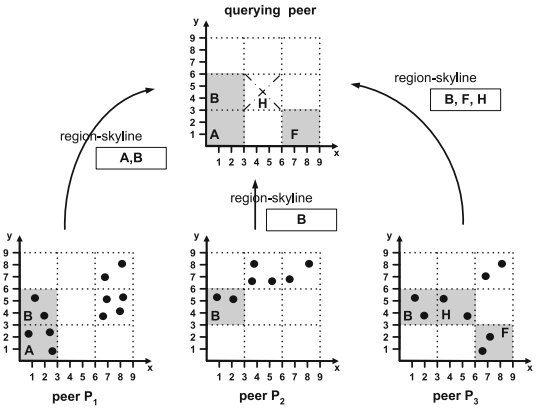


Fig. 11 AGiDS example [35]: finding the non-dominating regions of different peers

An example of this process is depicted in Fig. 11. Finally, queries are forwarded only to peers that correspond to at least one cell in the region-skyline set. The query initiator requests only a subset of local skyline points, namely those that belong to the cells of the region-skyline set. After having gathered all relevant points, the querying peer computes the global skyline set by testing only the necessary regions for dominance.

5.2.7 FDS [56]

Zhu et al. [56] propose a *f*eedback-based *d*istributed *s*kyline (FDS) algorithm, which assumes no particular overlay network. FDS aims at minimizing the network bandwidth consumption, measured in the number of tuples transmitted over the network. FDS requires a scoring function that is used by all peers, and each query is processed in multiple round trips. In each round trip, all peers send to the querying peer the k local skyline points with the lowest score based on the scoring function. Then, the querying peer computes the maximum score of all transferred local skyline points and requests from all peers the remaining local skyline points that have scores smaller than the maximum score. Finally, the querying peer merges the local result sets and selects a subset of the current skyline points as a feedback that is sent to all peers. Peers receiving the feedback remove from their local data points all points that are dominated by the points of the feedback.

In the feedback phase, FDS selects filter points for each peer; these are skyline points that are guaranteed to dominate at least ℓ local data points. To this end, for each local skyline point, the distance of the ℓ-nearest neighbor is computed and attached to it before sending it to the querying peer. The distance is combined with the score of the scoring function in such a way that FDS can decide whether a skyline point satisfies the condition. An example of the feedback algorithm is

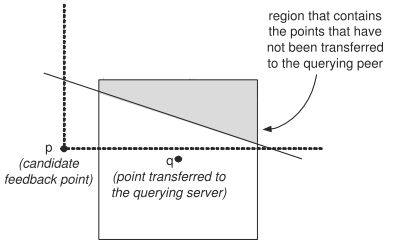


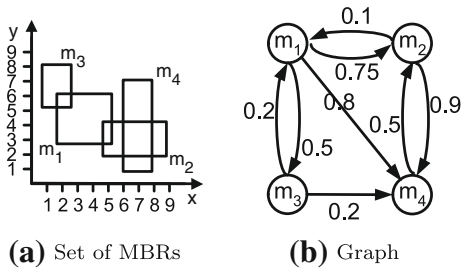
Fig. 12 FDS example [56]: example of the feedback algorithm

depicted in Fig. 12. The depicted rectangle is defined by the distance of the ℓ-nearest neighbor based on L_∞. The score of the scoring function used for sorting the data points defines the region that encloses the points that have not been transferred to the querying peer. Then, if the dominating region of a skyline point covers this region, it will dominate at least ℓ points. FDS is efficient in terms of bandwidth consumption. However, several round trips are required to compute the skyline set. Thus, it may incur high response time.

5.2.8 SkyPlan [36]

SkyPlan was proposed in [36] for improving the performance of PaDSkyline [13]. Similar to PaDSkyline, during query processing, each peer reports a set of minimum bounding rectangles (MBRs) to the querying peer as a summarization of its data. SkyPlan addresses the problem of generating execution plans, which define the order the query is executed. The query plan has a direct impact on the performance of skyline query processing. By querying the peers consecutively, some peers may not have to be contacted at all, if all points of a peer are dominated by a point stored locally at another peer. Furthermore, the amount of transferred data can be drastically reduced. However, if the filter points fail to prune any point of a peer, then no gain can be obtained from querying the peers consecutively. In this case, the parallelism should be preserved, in order to minimize the latency and therefore also the response time.

In SkyPlan, the query originator creates a weighted directed graph (SD-graph) that captures the dominance relationships between the collected MBRs. Each vertex of the graph represents a non-dominated MBR, while an edge between two vertices means that one MBR dominates partially the other MBR. The weights on the graph edges are defined by the pruning power, which is used to quantify the potential gain through filtering. For example, consider the MBRs and the graph depicted in Fig. 13. MBR m_1 partially dominates m_2 because the lower left corner of m_1 dominates the upper right corner of m_2. Thus, a directed edge from m_1 to m_2 is added to the graph. Before executing the query, SkyPlan transforms the SD-graph into an execution plan (one or more directed trees) that maximizes the total



(a) Set of MBRs **(b)** Graph

Fig. 13 SkyPlan example [36]: example of the MBRs and the constructed *graph*

pruning power while preserving the parallelism when no significant gain can be obtained from processing the queries on different peers consecutively. It has been shown that SkyPlan supports also multi-objective executions plans, in case that additional objectives need to be fulfilled simultaneously, such as additionally restricting the number of hops that the query is forwarded.

Finally, the distributed skyline query is processed based on the execution plan. The querying peer sends the query to the root of every directed tree in the execution plan. Each queried peer processes the skyline query locally, refines the execution plan, and selects a set of filter points. The refinement of the execution plan produces a new execution plan that does not contain the MBRs that are dominated by the local skyline points. The filter points are selected based on the dominating region [20]. Since the volume of the dominating region does not necessarily relate to the area within the MBR that is dominated by a filter point, SkyPlan takes into account the MBRs of the execution plan while selecting the filter points. Eventually, each peer gathers the local result sets of the peers that had received the query through and merges them by discarding dominated points. Local processing on peers terminates by returning the merged skyline points to the previous peer based on the execution plan.

6 Performance analysis

Most approaches described in Sect. 5 have been designed for efficient processing of skyline queries over different types of distributed systems with different assumptions. Thus, they are not directly comparable. Table 2 provides a categorization of approaches in two main groups depending on the overlay network: structured and unstructured P2P systems.

Qualitative comparison. Whereas structured P2P networks can only be applied when having full control over all participating peers, unstructured P2P systems allow peers to retain a higher degree of autonomy. As a natural consequence of assigning data to peers according to a globally known rule, structured P2P networks achieve more efficient query routing by applying the same rule at query time. On the other hand, in unstructured P2P networks, each peer routes the query based on the locally stored routing information. In worst case, the query is flooded in the entire network. However, structured P2P systems also require a higher construction cost. The reason is that each peer's data point needs to be indexed by an appropriate peer, which leads to a construction cost of $O(|S| \cdot N_P \cdot \log N_\mathrm{p})$ in terms of network traffic, where $|S|$ is the cardinality of the dataset S, and N_p the number of peers.

Based on these considerations, it is not necessarily meaningful to perform a direct comparison of both groups of approaches under the same conditions. Consequently, in the following, we will discuss each group in separate. Our objective was to provide a comparative performance analysis that highlights the differences and similarities of the approaches, their advantages and disadvantages, and their effect on the performance of the algorithms.

Metrics. One important factor that influences the performance of a distributed skyline algorithm in a P2P network is the length of the longest path the query is propagated on. A longer path implies a higher response time, since the query originator has to wait until the last peer receives the query, processes it, and finally forwards its results through the network. In addition, it also affects the amount of transferred data. The length of the path is measured in the number of hops, and the length of the longest path is referred to as maximum number of hops or cost. In our analysis, we mainly focus on this metric due to its important effect on other metrics. However, we also provide useful information on other related metrics, such as response time, amount of transferred data, and number of contacted peers. In the following, we denote the number of peers and super-peers as N_p and N_sp.

6.1 Structured P2P systems

The performance of approaches that rely on structured P2P networks mainly depend on (1) the underlying overlay network, (2) the rule for locating data, and (3) the individual features of the algorithm, such as filtering, pruning of dominated regions, and result propagation. The overlay network is the most important factor as it influences the efficiency of locating relevant regions as well as the ability of pruning irrelevant regions of the multidimensional space. Two main overlay networks are employed in the related work: CAN and BATON (Table 3).

The CAN overlay represents a grid-based partitioning of the d-dimensional Cartesian coordinate space among N_p peers. Therefore, approaches that rely on space partitioning, such as DSL and SSP/Skyframe, can be directly applied on CAN. On the other hand, BATON is a tree-based structured network that handles one-dimensional values. Hence, a one-dimensional mapping is required.

Table 3 Performance analysis for structured P2P approaches

Approach	Number of hops (CAN)	Number of hops (BATON)
DSL [51]	$O\left(d \cdot N_p^{\frac{1}{d}}\right) + O(N_p)$	–
SSP/Skyframe [48,49]	$O\left(2 \cdot d \cdot N_p^{\frac{1}{d}}\right)$	$O\left(\left(1 + 2 \cdot d \cdot \left(1 - \frac{1}{\sqrt[d]{N_p}}\right)\right) \cdot \log N_p\right)$
iSky [9,12]	–	$O\left(d \cdot \log N_p\right) + O\left(\frac{N_p}{d}\right)$

Comparison. DSL and SSP initiate query processing by locating the peer that is responsible for the region near the origin of the data space. Assuming a CAN overlay, this requires $O(d \cdot N_p^{\frac{1}{d}})$ hops. Thereafter, DSL dynamically constructs a multicast hierarchy, which in worst case will have a maximum length of N_p. Since all peers in the multicast hierarchy represent adjacent partitions, the cost of traversal is $O(N_p)$. On the other hand, SSP contacts all non-dominated regions in parallel, which leads to a cost of $O(2 \cdot d \cdot N_p^{\frac{1}{d}})$ hops [49]. Skyframe first queries all border regions in parallel and then the remaining non-dominated regions. Thus, it incurs the same cost as SSP. In terms of transferred data, DSL transfers all local skyline points through the multicast hierarchy (i.e., multiple hops), whereas SSP/Skyframe routes only the filter point and gather the local results by direct transfer (i.e., single hop). Thus, DSL incurs higher bandwidth consumption than SSP/Skyframe.

The experimental evaluation in [48,49] shows that Skyframe performs better than DSL in all considered aspects, i.e., in the number of nodes involved in the search process, the number of skyline search messages, the number of search hops, the amount of bandwidth consumption, and the query load distribution, for varying network size, data dimensionality, and cardinality. DSL outperforms Skyframe only with respect to load balancing with a relatively small number of nodes (up to about 120).

SSP/Skyframe employs the Z-curve method for one-dimensional mapping to BATON. Even though the algorithm is the same as in the case of SSP/Skyframe on the CAN network, the cost of locating the regions changes. As stated in [49], the cost of a skyline query is $O((1 + 2 \cdot d \cdot (1 - \frac{1}{\sqrt[d]{N_p}})) \cdot \log N_p)$ hops for the uniform distribution. On the other hand, iSky employs data transformation to one-dimensional values (using the maximum value in all dimensions) rather than space partitioning. As stated in [12], iSky exploits the BATON protocol better than SSP/Skyframe since it distributes data portions across peers more deliberately. With respect to query processing, iSky detects the peers storing boundary partitions with cost $O(d \cdot \log N_p)$. The skyline query is first processed by these peers, and some of the remaining peers are pruned based on a threshold value. To further reduce the amount of transferred data, a filter point is used, similar to SSP/Skyframe. Then, the query is propagated to peers that have not been pruned based on the threshold until all peers have been queried or pruned. Thus, in contrast to SSP/Skyframe, only some peers are processed in parallel, and the query is propagated to neighboring peers. In each step, both the threshold and the filter point are refined. The adaptive filtering technique of iSky enhances its pruning capability, which probably leads to fewer transferred data than SSP/Skyframe. Still, in the worst case iSky needs to contact all peers. Since the query is forwarded to neighboring peers by starting at d peers, we estimate this cost as $O(\frac{N_p}{d})$.

The experiments provided in [9,12] compare SSP to iSky with respect to dimensionality, network size, and cardinality. The results show that iSky outperforms SSP in consideration of communication costs and the number of involved nodes. Due to the fact that only iSky is progressive, iSky returns first results much earlier than SSP. In addition, the experiments show that iSky also outperforms SSP with respect to the total execution time. It should be noted that in [48,49], it has been shown that Skyframe/SSP outperforms DSL when applied on BATON.

Summary. We conclude that SSP/Skyframe seems to be more appropriate if the underlying network is CAN, while iSky outperforms Skyframe in the case where the underlying network is BATON. It should be noted that we omitted SSW from the discussion. The reason is that it is completely different from CAN and BATON because the data in SSW are not assigned to peers based on a globally known rule. Instead, each peer is associated with a semantic label that is used for routing.

6.2 Unstructured P2P systems

Almost all algorithms proposed for unstructured P2P systems ultimately aim at minimizing the response time. The only exception is FDS that focuses on minimizing the amount of transferred data. To achieve this goal, FDS requires several round trips to compute the skyline set, leading to high response times. Therefore, FDS is excluded from the following discussion as it is only applicable when the bandwidth is restricted or costly, as in the case of mobile environments.

In an unstructured P2P network, the maximum number of contacted peers and super-peers is N_p and N_{sp} respectively because in the worst case scenario, all peers locally store at least one of the global skyline points. This depends on the

Table 4 Performance analysis for unstructured P2P approaches

Approach	Number of hops (Construction)	Number of hops (Query)	Routing paths
SFP [20]		$O(N_p)$	Network topology
DDS [18,19]	$O(N_p)$	$O(N_p)$	Network topology
SKYPEER/SKYPEER + [44,46]	$O(1)/O(1) + O(N_{sp})$	$O(N_{sp})$	Network topology
BITPEER [16]	$O(1)$	$O(N_{sp})$	Network topology
PaDSkyline [8,13]		$O(1) + O(N_p)$	Domination relationships
AGiDS [35]		$O(1) + O(1)$	Direct
SkyPlan [36]		$O(1) + O(N_p)$	Domination relationships

distribution of data points with respect to the peers, which cannot be improved if the peers' autonomy is preserved.

Comparison. PaDSkyline, SkyPlan, and AGiDS assume a fully connected network topology and processing consists of two round trips. In the first round trip, the querying peer gathers a summary of the data stored at each peer. In the second round trip, the query is propagated only to those peers that store relevant data. Thus, in the first round trip, N_p messages are required, but the maximum number of hops is equal to 1. For the second round trip, AGiDS directly communicates with the relevant peers, while PaDSkyline and SkyPlan create a query plan that in the worst case leads to N_p hops. Thus, the number of maximum hops for AGiDS is $O(1) + O(1)$, while for PaDSkyline and SkyPlan it is $O(1) + O(N_p)$.

All approaches based on super-peers require a preprocessing phase, in which the extended skyline points of the peers are transferred to the associated super-peer with cost $O(1)$. In addition, SKYPEER+ constructs routing indexes, which requires propagating the MBRs in the super-peer network, leading to an additional cost of $O(N_{sp})$. During query processing, the maximum number of hops for all approaches is $O(N_{sp})$ since in the worst case all super-peers have to be contacted sequentially.

SFP and DDS assume a pure P2P network. SFP uses flooding for query propagation. Thus, the maximum number of hops is $O(N_p)$. DDS constructs routing indexes that lead to a construction cost of $O(N_p)$. During query processing, the maximum number of hops for DDS is $O(N_p)$ because in the worst case all peers have to be contacted sequentially.

Beyond the longest path, the amount of transferred data influences the response time. In the worst case, all approaches need to transfer all local skyline points (SKY_i). The optimal case is to transfer only those local skyline points that belong to the global skyline, i.e., SKY points, but this is not feasible in a distributed environment. In general, it holds that $\sum_{i=1...N_p} SKY_i \approx N_p * SKY >> SKY$ because the cardinality of the skyline set mainly depends on the data distribution and dimensionality, and much less on the data cardinality. All approaches try to reduce the amount of transferred data by filtering or query plans. The amount of pruned data is difficult to estimate because it depends on the data distribution, the data dimensionality, the given network topology, and the way that data are distributed to the peers. AGiDS, SkyPlan, and PaDSkyline discard dominated regions, therefore, also dominated local skyline points. AGiDS summarizes the data using non-overlapping regions that probably lead to more regions being discarded. In contrast, PaDSkyline and SkyPlan use MBR as a summarization, which leads to overlapping regions that may result in contacting all peers sequentially. In this case, the longest path is N_p peers, but this enables more effective filtering and thus less transferred data. PaDSkyline creates routing paths based on domination relationships and uses filtering to discard dominated local skyline points. On the other hand, SKYPEER, SKYPEER+, BITPEER, DDS, and SFP use filtering (or thresholding) to discard dominated points, but the routing paths belong to the given network topology. Furthermore, it has been shown [46] that SKYPEER+ leads to fewer transferred data than SKYPEER because the threshold is refined based on the routing information.

Summary. Table 4 summarizes our findings. When a fully connected network topology can be supported, it is preferable as it is expected to work better in practice. The reason is that the routing paths are based on the domination relation between data, thereby pruning a significant number of local data. The experiments provided in [35] show that AGiDS outperforms PaDSkyline in all tested setups. On the other hand, AGiDS requires global knowledge of a grid structure with identical boundaries for all peers, which may not be feasible in some applications. Furthermore, in [36] it has been shown that SkyPlan outperforms PaDSkyline due to a more efficient execution plan. In case multiple hops are required for contacting each peer, the cost of these approaches increases since at most N_p hops are required for gathering the summaries or querying the peers at each step leading to a cost of $O(N_p)+O(N_p)$ (for AGiDS) or $O(N_p)+O(2 \cdot N_p)$ (for PaDSkyline, SkyPlan). Therefore, in this case, AGiDS, SkyPlan, and PaDSkyline are not suitable.

Since $N_{sp} < N_p$, approaches based on super-peers are more efficient than pure P2P approaches, if the opportunity

of having dedicated servers exists. It has been shown experimentally [46] that SKYPEER+ outperforms SKYPEER. Finally, if no dedicated servers exist and peer failures are common, the construction of the routing information at the super-peers will be too costly. In such a case, DDS or SFP is more appropriate. DDS has a smaller cost and is expected to lead to a better performance than SFP.

However, some of the approaches require a preprocessing phase for constructing routing information. These approaches are not applicable when data are highly dynamic since the overhead of updating the routing information will outweigh the performance gains. In this case, it is more efficient to apply SFP, AGiDS, SkyPlan, or PaDSkyline, which gather summary information only during query time.

7 Skyline variants in P2P systems

Several of the aforementioned distributed approaches were proposed for efficient processing of skyline query variants, such as subspace, constrained, or dynamic skyline queries. Independently from the variant the approach was originally proposed for, they all support skyline queries efficiently because the skyline query in its original sense is a special case of any skyline variant. Whether an approach additionally supports a skyline variant (other than the one it was proposed for) mainly depends on its routing mechanism and filtering method.

In this section, we elaborate on the skyline variant each approach was proposed for and discuss whether an approach can easily be adapted to support other skyline variants efficiently. Therefore, we identify the underlying factors that may alter the correctness or the efficiency of an approach when a skyline query variant is processed.

Before delving into details, let us point out two approaches, SFP [20] and FDS [56], that support all skyline variants considered in this survey. SFP [20] forwards the query to all peers. Hence, SFP can be adapted in a straightforward manner to support any skyline query variant. The only necessary adaptation is that the filter point has to be selected properly based on the given query variant. This is straightforward for any skyline query variant. FDS [56] also forwards the query to all peers and supports all skyline query variants, but in order to be efficient in terms of bandwidth consumption, the data points should be sorted appropriately depending on the given query variant. The monotone function that is used for sorting should be applicable in the given subspace or dynamic space.

7.1 Subspace skyline queries

Subspace skyline queries are supported by most of the distributed skyline approaches. SKYPEER [44,46] and BITPEER [16] were originally proposed for supporting subspace skyline queries efficiently. For the remaining approaches, the question that arises is whether they can perform efficient query routing to relevant peers when only a subset of the dimensions is considered.

Approaches that use MBRs for query routing, such as PaDSkyline [13], SkyPlan [36], and DDS [18,19], can easily be adapted to support subspace skyline queries since the MBRs can be projected into the given subspace before being processed. Then, peers can be pruned based on the projected MBRs, and the correctness of the approaches is still guaranteed. AGiDS [35] can also be extended for processing subspace skyline queries, even though AGiDS adopts a grid-based data summary for query routing, instead of MBRs. Subspace skyline queries can be supported by using the projection of the data partitions in the requested subspace before query routing. If the grid-based data summary was created by dividing each coordinate into s_i intervals, then the projected cells form a grid-based data structure, which enables query routing and peer pruning without influencing the correctness of the algorithm.

DSL [51], Skyframe [48,49], and SSW [25] assume a similar space partitioning and map each partition to a particular peer through the structured overlay. These approaches cannot support subspace skyline queries efficiently, and the reason is threefold. First, these approaches assume that the local skyline points of the peer that indexes the origin of the data space can immediately be returned to the user without merging. This does not hold when subspace skyline queries are processed since more than one data partition overlap in the projected space. Furthermore, the efficiency of these approaches relies on the fact that adjacent data partitions are indexed by neighboring or nearby peers, which in turn enables efficient query routing. In the case of subspace skyline queries, data partitions that are indexed by non-neighboring peers may be projected into the same region in the subspace. Finally, another property that is required by these approaches is that the data partitions are non-overlapping, which allows these approaches to route the query efficiently and to avoid contacting the same peer twice. If the data partitions are projected into a subspace, it is possible that the projected data partitions are overlapping. It is not straightforward how peers that store adjacent data partitions in subspaces can be detected efficiently and how these approaches would handle overlapping data partitions. None of the proposed approaches has handled these issues effectively. iSky [9] assigns data points to peers based on the dimension each data point has the maximum value of its coordinates. Although this improves the efficiency of skyline query computation, it makes iSky inapplicable for subspace skyline queries. The mapping of data points to peers does not allow to process only a subset of dimensions as some data points would be falsely discarded. The alternative would

Table 5 Different skyline query variants supported by distributed approaches (×: proposed for, ◇: also supports)

Approach	Skyline	Subspace	Constrained	Dynamic
DSL [51]	◇		×	
SSP/Skyframe [48,49]	×		◇	
iSky [9,12]	×			
SSW [25]	×		◇	
SFP [20]	×	◇	◇	◇
DDS [18,19]	×	◇	×	◇
SKYPEER/SKYPEER+ [44,46]	◇	×		
BITPEER [16]	◇	×		
PaDSkyline [8,13]	◇	◇	×	◇
AGiDS [35]	×	◇	◇	
FDS [56]	×	◇	◇	◇
SkyPlan [36]	×	◇	◇	◇

be to consider all dimensions and process a subspace skyline query in a similar way as a skyline query. Although this alternative would lead to the correct result, it is inefficient since the dimensionality of the data space may be much higher than the subspace dimensionality and therefore processing a subspace skyline query should also be more efficient than exploiting the entire data space (Table 5).

7.2 Constrained skyline queries

DSL [51], DDS [18,19], and PaDSkyline [13] were originally proposed not only for skyline queries but also for constrained skyline queries. Therefore, these approaches support constrained skyline queries efficiently. The main challenge that distributed approaches must handle in order to efficiently support constrained skyline queries is that the routing mechanism and filtering method must be applicable in case only a region of the entire data space is considered.

Similar to DSL [51], Skyframe [48,49] and SSW [25] support constrained skyline queries as they all assume a similar space partitioning that is mapped to peers through different structured overlays. The constrained skyline query processing starts from the peer that indexes the lower left corner of the given constraint (instead of the peer that indexes the data origin), and peers that do not overlap with the query are pruned. The main challenge that Skyframe and SSW must address is to detect the starting peer efficiently, which is feasible with no significant additional cost. AGiDS [35] can easily be adapted for constrained skyline queries, even though it was proposed and evaluated only for skyline queries. Constrained skyline queries can be supported by taking into account only the cells of the grid that overlap with the query. In this case, the proposed approach has to be modified slightly since some cells may partially overlap with constraints. Therefore, it is not guaranteed that they contain at least one skyline point. SkyPlan [36], similar to PaDSkyline, can efficiently support constrained skyline queries by discarding MBRs that do not overlap with the given constraints.

On the other hand, iSky [9] is inefficient for constrained skyline queries. Even though query processing could start from the peers that in each dimension store the highest value that is smaller than the upper bound of the constraint, the efficiency of iSky relies on the fact that the peers that index the maximum values are known a priori. This is not feasible when each query is associated with a different constraint, and it is not straightforward how to detect these peers efficiently during query processing. SKYPEER [44,46] and BITPEER [16] cannot be extended for constrained skyline queries easily because the thresholding scheme used for filtering relies on a one-dimensional mapping that is computed based on the data space, making the adaptation of the threshold for a given constraint non-trivial.

7.3 Dynamic skyline queries

Dynamic skyline queries comprise the most challenging variant of skyline queries for distributed computation. The main challenge is that the query is not executed in the original data space but in a dynamic space that is query dependent, i.e., it differs for different queries. None of the distributed skyline approaches has originally been proposed for efficient processing of dynamic skyline queries.

Approaches relying on an MBR-based routing mechanism, such as DDS [18,19], SkyPlan [36], and PaDSkyline [13], can support dynamic skyline queries under the condition that the functions of the dynamic skyline query allow to map every MBR to a transformed MBR that encloses all data points in the transformed data space. Then, the dominance relationship between MBRs in the dynamic space for query routing and peer pruning leads to the correct result set. DSL [51], Skyframe [48,49], and SSW [25] do not support dynamic skyline queries because peers do not necessarily index disjoint data partitions in the dynamic space, and neigh-

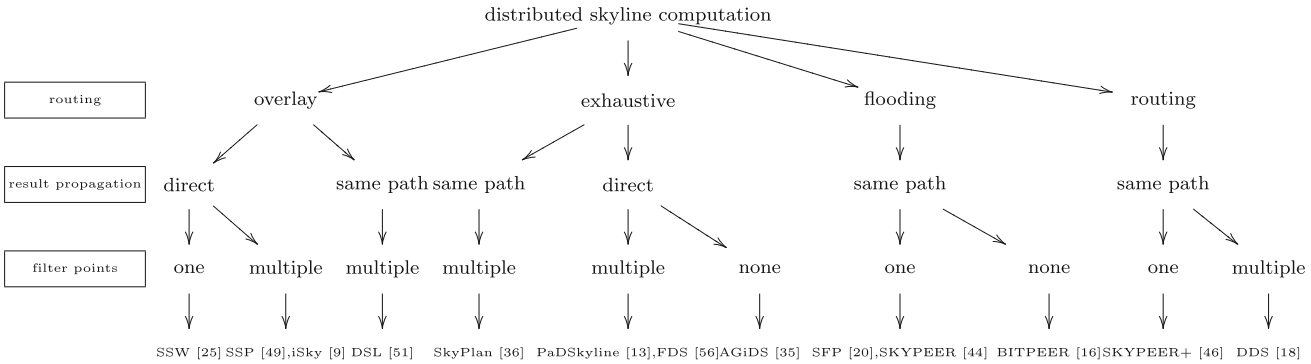


Fig. 14 Taxonomy of skyline query processing approaches

boring data partitions are not indexed by neighboring peers. Dynamic skyline queries cannot be supported by AGiDS as there is no guarantee that the cells of the grid remain non-overlapping in the dynamic space. iSky [9] cannot support dynamic skyline queries because the properties of the one-dimensional mapping that are used for query processing do not hold in the dynamic space. Similarly, SKYPEER [44,46] and BITPEER [16] cannot easily be extended for dynamic skyline queries because the thresholding scheme used for filtering is not extensible for the dynamic space.

8 Taxonomy

In this section, we provide a categorization of the distributed skyline approaches in a taxonomy that summarizes and highlights differences and similarities. The categorization is based on the different techniques employed in the phases discussed in Sect. 4 and on the main principles used by each approach. Figure 14 shows the resulting taxonomy, which categorizes the approaches discussed in this paper. Furthermore, in Table 6, we also state the objectives of each distributed skyline approach.

An important phase that influences the performance of a distributed skyline approach is query routing. During query routing, a peer tries to eliminate as many neighbors as possible and forwards the query only to the remaining neighbors. Most approaches for distributed skyline processing cause queries to travel along paths in the network. In general, these paths are not determined in advance but influenced by the local skyline points retrieved at each peer, which are used to decide on the subset of neighboring peers that can contribute to the skyline set. In case of structured P2P systems, a peer exploits the information about data distribution in the underlying overlay to route queries efficiently to relevant neighboring peers [9,12,25,48,49,51]. On the other hand, in unstructured P2P networks due to the absence of any information about the data distribution, a straightforward alternative is to forward the query to all available peers (or super-peers) using flooding [16,44]. Then, each peer that receives the query forwards it to all of its neighbors. A more efficient alternative used in unstructured P2P networks is to build routing indexes that store sufficient information to decide on the relevance of neighbors [18,19,46]. Routing indexes store summaries of the data that are available through each neighboring peer. Based on these summaries, it is possible to detect irrelevant neighbors and prune them from further consideration without querying them. Finally, in approaches [8,13,35,56] assuming a fully connected network topology that enables direct communication between peers, the querying peer contacts all peers to gather some information about their data. Then, based on this information, the querying peer decides which peers have to be queried in a subsequent round in order to compute the skyline set.

Result propagation is another important factor that influences the performance of the merging phase. There are two ways of propagating local result sets through the P2P network to the query initiator. Some approaches use direct communication [8,9,12,13,25,35,48,49,56], i.e., each peer sends its local result set directly to the query initiator. The advantage is that local skyline points have to be sent only once. However, the computational load at the initiator is relatively high since all incoming points need to be checked for mutual dominance to obtain the global skyline set. We notice that most of the approaches [8,13,35,56] that assume a fully connected network topology use direct result propagation because it is feasible for every peer to contact the querying peer directly. Furthermore, approaches [9,12,25,48,49] that rely on a structured overlay also use direct result propagation. As an alternative to direct result propagation, local result sets can be propagated back using the same path that the query has been forwarded along [16,18–20,36,44,46,51]. Each peer receives the results from the neighbors it has sent the query to, discards dominated local skyline points, and sends the merged result sets to the peer it has received the query from. In this case, the processing load of result merging is shared among all peers. Every existing approach [16,18,19,44,46] that assumes an unstructured (pure or hybrid) P2P network

Table 6 Overview of the features and objectives of the different distributed approaches

	Filter points	Routing	Result propagation	Optimization goal
DSL [51]	All local skyline points	Overlay	Same path	Response time (network communication cost and load balancing)
SSP/Skyframe [48,49]	Most dominating point	Overlay	Direct	Network communication cost/response time
iSky [9,12]	Most dominating point and threshold	Overlay	Direct	Response time and network communication cost
SSW [25]	Nearest neighbor	Overlay	Direct	Scalability (network communication cost and contacted peers)
SFP [20]	Most dominating point	Exhaustive	Direct	Network communication cost
DDS [18,19]	All local skyline points	Routing index	Same path	scalability (network communication cost and contacted peers)
SKYPEER [44]	Threshold	Flooding	Same path	Response time (computational time and transferred data)
SKYPEER+ [46]	Threshold	Routing index	Same path	Response time (computational time, transferred data and contacted super-peers)
BITPEER [16]	No filter points	Flooding	Same path	Response time
PaDSkyline [8,13]	Multiple filter points	Exhaustive	Direct	Response time (parallelism and network communication cost)
AGiDS [35]	No filter points	Exhaustive	Direct	Response time
FDS [56]	Multiple filter points	Exhaustive	Direct	Network communication cost
SkyPlan [36]	Multiple filter points	Exhaustive	Same path	Response time (parallelism and network communication cost)

uses the same path for result propagation that was used for forwarding the query to the relevant peers.

Almost all approaches use the principle of filtering for efficient query processing. The filter points are attached to the query that is forwarded to neighboring peers, so that dominated local data points can be discarded immediately. Furthermore, filter points are also used to eliminate those neighboring peers that store only dominated data points. Usually, some information about the already computed local skyline set is forwarded along with the query. This information may consist of one, multiple, or all local skyline points. In the most common case, only a single point is used as a filter point [9,12,20,25,48,49]. This is because the filter points that are attached to the query also increase the amount of transferred data, especially if the filter points fail to prune any local skyline points or neighboring peers. Also, we notice that the approaches [9,12,20,48,49] that use a single filter point pick the most dominating point [20] as filter point. On the other hand, some approaches select multiple filter points [8,13,36,56] or even use the entire local skyline set for filtering [18,19,51]. Furthermore, other approaches [9,12,44,46] facilitate filtering by means of a one-dimensional threshold value. Even though filter points usually improve the performance of distributed skyline computation, in some cases [16,35], no filter points are used.

Table 6 states the objectives of each distributed skyline approach. Most of the existing approaches [8,9,12,13,16,35,36,44,46,48,49,51] aim to minimize the overall response time. In order to achieve low response time, most approaches aim to minimize different factors that influence the response time, which are also stated in the last column of Table 6. On the other hand, some approaches focus only on the network communication cost [20,56] or on the scalability [18,19,25] of the proposed method.

9 Other distributed environments

In the following, we review in detail approaches tailored for distributed environments other than highly distributed systems (such as P2P systems). The fundamental differences of these approaches are due to (some of) the following reasons: (1) each server does not necessarily store a fraction of the available data, (2) the goals of skyline processing are different, and (3) several round trips are required to retrieve the result set.

9.1 Web information systems

Skyline processing over distributed web information systems was studied in [2,28]. In such systems, each source stores the

object identifier and a different attribute from the remaining attributes of the data objects. With respect to our hotel example, the price can be provided by a travel agency website, whereas the distance to the beach by an online server providing geographical information. Approaches for web information systems have different objectives than those discussed before since the attributes of each data object are provided by different sources, while in a highly distributed system, we assume that the data objects are distributed in a way that guarantees that all attribute values of a data object are stored at the same server. This leads to a fundamentally different setup since the number of sources in web information systems, which are equal to the number of attributes of the skyline query, is very small in comparison with the total number of servers in a highly distributed system. Moreover, skyline processing in web information systems aims to minimize the response time, without the restriction of having a fixed number of round trips.

During skyline query processing, all sources need to be contacted. In more detail, each time a source is accessed, one object is transferred to the querying server. There are two basic kinds of accesses that are provided: retrieving the next object with the best value from one source with respect to a single attribute called a *sorted access* or retrieving the score value with respect to one source for a certain given object called a *random access*.

Three algorithms have been proposed for web information systems, namely the basic distributed skyline (BDS) algorithm [2], the improved distributed skyline (IDS) algorithm [2], and the progressive distributed skyline (PDS) algorithm [28]. All algorithms consist of two phases. In the first phase, a subset of objects is retrieved which includes at least all skyline objects. More specifically, data are retrieved by sorted access from the different sources until all attribute values of at least one data object (referred to as the terminating object) have been retrieved from all sources. The second phase discards all the non-skyline objects in the subset by pruning dominated points. Missing attribute values that are required for the domination tests are retrieved through random access during the second phase. The main observation [2] that is used to reduce the number of domination tests is that an object o can be dominated only by other objects that have been retrieved from the same sources as o.

An important differentiating feature of the three algorithms is the order in which the sources are accessed during the first phase. This order influences the number of sorted accesses and therefore the efficiency of the algorithm. Applying BDS, each data source is accessed in a round-robin fashion, while IDS uses a heuristic to detect the most promising source, i.e., the source that leads to a terminating object with fewer accesses. A score is assigned to each retrieved object to estimate the remaining number of sorted accesses required to retrieve all missing attributes. The score is calculated as the difference between the attribute values and the last values retrieved through sorted access from each source. The object with the best score is considered to be the most probable terminating object. In contrast to BDS that performs only sorted access in the first phase, IDS requires that the missing attribute values of the most probable terminating object are retrieved through random access in order to compute the score. In each iteration, IDS accesses a source from which the most probable terminating object has not yet been retrieved. PDS improves IDS by using a linear regression method to estimate the ranks of the object and determine a better order to access the sources. Furthermore, PDS is progressive and returns each skyline point to the user as soon as it is guaranteed that no other point can dominate it.

9.2 Parallel shared-nothing architecture

In a parallel shared-nothing architecture, there exists one central server, called coordinator, which is responsible for a set of servers. As the skyline computation is CPU-intensive, the coordinator distributes the processing task to all available servers. This is achieved by first partitioning the input data and then assigning each partition to one server. Then, each server computes the skyline over its local data and returns its local skyline result set to the coordinator, which merges the result sets and computes the global skyline result.

In this setting, the goal was to minimize response time by sharing the workload as evenly as possible among all participating servers. To achieve this objective, three fundamental issues need to be considered. First, each server should be allocated approximately the same number of data points. Second, the skyline algorithm should have similar performance on the data points in every partition. Finally, the local skyline points returned to the coordinator for the merging phase should be minimized in order to avoid overburdening the coordinator and wasting resources at the final step. Obviously, given a set of servers, the overall skyline query performance depends on the efficiency of the local skyline computation and the performance of the merging phase. Thus, the efficiency of the parallel skyline computation for a shared-nothing architecture mainly depends on the space partitioning method used for distributing the dataset among the servers.

Typical partitioning schemes include grid partitioning and random partitioning to servers. In [45], the advantages of an angle-based partitioning scheme are demonstrated for efficiently parallelizing skyline computation. The proposed technique first maps the Cartesian coordinate space into a hyperspherical space and then partitions the data space based on the angular coordinates into N partitions, as many as the available servers. This partitioning scheme alleviates most of the problems of traditional grid partitioning and hence manages to reduce the response time and to share the computational workload more fairly.

9.3 Distributed data streams

Skyline queries have also been defined over data streams under the assumption of sliding windows, where each data object is associated with a timestamp indicating its time of arrival. Furthermore, the width of the sliding window defines the lifespan of any object. Given a timestamp, the skyline set contains the data objects that are valid at this timestamp and not dominated by any other valid data object at that timestamp. Centralized algorithms [26,40] for skyline queries over streams focus on efficiently detecting data objects that become skyline points after a skyline point expires.

Relying on a distributed data stream model, Sun et al. [38] propose an algorithm (BOCS) for skyline queries over data streams. In such a system, there exists a set of servers, each of them producing some of the data objects of the stream. In addition, there exists a central server that communicates with the remote servers and is responsible for evaluating the queries. This setup is similar to highly distributed systems and more particularly to the fully connected network topology. The main difference is that in the case of streams, the challenge is to efficiently monitor the skyline over time, rather than computing the skyline at a given timestamp. In BOCS, each server monitors the local skyline set by using a centralized algorithm. Then, at each timestamp, only data objects that are added to the skyline set are sent to the central server. Based on the additivity of the skyline operator, points that are not contained in the local skyline set are not sent to the central server, thus reducing the communication overhead. Finally, the central server applies a centralized skyline algorithm over the received streamed data and efficiently computes the skyline set at each timestamp.

9.4 Wireless sensor networks

Supporting skyline query processing in wireless sensor networks (WSNs) has been studied in [7,52]. A wireless sensor network consists of n stationary sensors randomly deployed in a region of interest, and each sensor measures d attribute values. In addition, there exists a base station that communicates with all sensors through a single hop or multiple hops. Each sensor can communicate with the sensors located within its transmission range. A routing tree rooted at the base station that spans all sensors is employed for query processing in sensor networks. Initially, the query is pushed down to each sensor along the paths of the tree. Then, the results are collected from children to parent sensors and eventually to the base station through multiple hops.

In such an environment, the baseline skyline algorithm is in principle similar to those applied in the context of highly distributed data. Each sensor computes the skyline set of its local points and the points received from its children. Then, this skyline set is transmitted to its parent sensor. Finally, the base station calculates the skyline of the received points resulting in the skyline set of all data available in the WSN. Nevertheless, there are two main differences compared to highly distributed systems. First, the sensors only have limited storage and processing capabilities in comparison with powerful computers, such as those encountered in P2P systems. Second, the optimization goal is different since the main goal in WSN was to minimize the energy consumption of the battery-powered sensors. In turn, this means minimizing wireless communication, which is the dominant factor of energy consumption in WSN. Besides minimizing the total energy consumption, the maximum energy consumption among the sensors should also be minimized since the sensors near the base station are expected to exhaust their batteries first. Consequently, the remaining sensors would become disconnected from the base station as they would neither be able to receive queries nor send data to the base station.

In [52], Xin et al. propose an energy-efficient algorithm, called Sliding Window Skyline Monitoring Algorithm (SWSMA), that employs two types of filters within each sensor to restrict the amount of data transferred to reduce energy consumption. The tuple filter approach uses a single tuple as a filter, which is selected to be the tuple that dominates the most other tuples. This tuple is detected based on the probability density function that is approximated by a polynomial function. The alternative approach is the grid filter approach, where a grid is used to partition the data space. Given a skyline point, the dominated cells are set to zero, meaning that the tuples in those cells do not belong to the skyline. Then, the grid is transmitted to the sensors and used as a filter to discard dominated points.

In order to improve SWSMA, Chen et al. [7] propose an algorithm that proceeds in k iterations. In each iteration, the skyline set of a partition of the data space is computed. The distance of the data points from the origin of the data space is used to define the partitions. In each iteration, only the local skyline points whose distance belongs to a given range are transmitted to the parent sensor. More precisely, the local skyline points are merged by discarding dominated points with the skyline points of the children sensors and the filter points received from the parent sensor. After each iteration, the base station merges all local skyline points and computes a set of filter points that are transmitted to all the sensors. At most d filter points are selected based on a modified dominance area that takes into account which regions of the data space have already been examined.

9.5 Uncertain data

Query processing over uncertain data has received considerable attention from the database community recently. Skyline

queries over uncertain data have been studied in centralized [30] but also in distributed [15] domains. The distributed system employed in [15] is similar to the fully connected network topology. Nevertheless, the probabilistic skyline query differs substantially from the skyline query and different challenges arise for efficient processing of distributed probabilistic skyline queries.

The probability that a data object belongs to the probabilistic skyline set is equal to the probability that this data object exists, while at the same time, all data objects that dominate it do not exist. Essentially, all data points belong to the probabilistic skyline set with some probability. Therefore, a probabilistic skyline query is associated with a given threshold, and all data objects with a probability at least equal to the threshold belong to the result set.

The main principles for processing a distributed skyline query over uncertain data are similar to those discussed in Sect. 4. Each server computes its local probabilistic skyline set and sends it to the coordinator. These sets are merged by the coordinator, and for each local skyline point, the probability is refined based on all collected data points. Local skyline points with probabilities (of belonging to the skyline set) smaller than the threshold are discarded. This is different than traditional skyline queries, where dominated points are discarded. Moreover, after merging, the probabilistic skyline set is not guaranteed to be the correct result set since the property of additivity does not hold for the probabilistic skyline query. Therefore, an additional challenge is to compute the exact probabilities of the candidate skyline points based on all data points stored at any server. To this end, an additional round trip is required, where the candidate skyline points are sent to all servers in order to compute their exact probabilities. The correctness of this approach relies on two facts: (1) the local probability of a point is an upper bound of its actual probability and (2) the probabilities can be computed accumulatively.

In [15], the DSUD algorithm aims to reduce the number of transmitted data objects by using multiple communication round trips. Initially, only the most promising point (in terms of local probability) is transferred from each server to the coordinator. After merging the candidate skyline points, the coordinator sends back as a feedback only the most promising candidate skyline point. The servers not only compute the accurate probability of the candidate skyline point but also refine the probabilities of the local data points based on the feedback. This is similar to filtering used in distributed skyline computation. But instead of filtering out points that are dominated, the aim here is to compute more accurate bounds for the probabilities, leading to pruning a higher number of data objects.

10 Open research directions

Even though several approaches have been proposed for efficient skyline computation in distributed environments, there are still interesting and challenging issues about distributed skyline computations that have not been studied so far in the related literature. First, as we concluded in Sect. 7, none of the existing approaches has studied dynamic skyline queries. Processing dynamic skyline queries is more challenging than traditional skyline computation as the skyline computation is based on a set of user-specified functions. Consider the travel agency application scenario, then it is reasonable that each hotel is associated with its geographical coordinates, while the user tries to minimize the distance to a location of interest, such as the conference venue. In this case, a dynamic skyline query has to be processed based on the conference location. Most of the existing distributed approaches cannot support such queries efficiently. Furthermore, when reviewing the related work, we noticed that only [56] tackles the interesting topic of continuous skyline maintenance. In a highly distributed setting, it is reasonable to try to keep the already computed skyline set up-to-date in the presence of data insertion or deletions, rather than processing the entire skyline query from scratch.

A current trend in data management research is management of data with uncertainty. Probabilistic skyline queries over uncertain data have been studied mainly in centralized settings [30] so far. In distributed environments, the uncertainty of the data occurs more naturally as it is caused by several different reasons related to the distributed setting. First, the data itself can be uncertain, similar to the case of the centralized setting. For example, in sensor networks, the uncertainty can be the result of noisy readings from sensors. Second, in a P2P network, peers themselves may not always be trusted by other peers, and some of them might act as cheaters deliberately. Each participating peer may be associated with a value of trustworthiness that indicates the validity of its data, thus rendering its query results uncertain. This value of trustworthiness may be dynamic and vary depending on the querying peer and its opinion about neighboring peers. In this case, the data points are associated with uncertainty based on which peer stores them or through which peer they are accessible and probabilistic skyline query processing is required. In our opinion, the uncertainty in data is inherent in distributed applications due to the lack of central control that verifies the quality of the data. Thus, distributed query processing on uncertain data is very important and challenging for deploying real-world widely distributed applications.

Finally, it has been shown that the cardinality of the skyline set can be high [55], especially in the case of high-dimensional or anti-correlated datasets. In a distributed environment, the high cardinality of the skyline set does not only incur high processing cost but also leads to high bandwidth

consumption due to the non-negligible amount of transferred data since in the best case scenario, at least the skyline points have to be transferred to the querying peer. Moreover, the total number of local skyline points is substantially higher than the number of global skyline points. In addition, in several distributed environments, such as mobile networks or cloud computing, the users may be charged based on the amount of transferred data. Therefore, cardinality estimation of the skyline set is very important in distributed environments. Even though several approaches have been proposed for estimating the cardinality of the skyline set in centralized settings, the applicability of these methods for distributed environments has not yet been studied. If the cardinality of the skyline set is estimated to be high, then it is more cost-efficient to compute an approximation of the skyline set by selecting only a subset of the local skyline sets and propagating them to the querying peer [47]. Thus, it becomes very important to study different selection techniques for local skyline points in order to retrieve an approximation of the skyline set with quality guarantees. Moreover, to address the problem of the high cardinality of the skyline set, recent trends in centralized settings include finding representative skyline points [27,41] or ranking the skyline points [6]. Finding representative skyline points in a distributed manner that fulfill properties similar to those proposed in [27,41] is still an open and challenging problem.

11 Conclusion

Less than a decade ago, the database research community began to pay rising attention to the problem of processing skyline queries. The popularity of the skyline operator is mainly due to its ability to identify a set of interesting objects in a large database. During the last decades, the vast number of independent data sources and the high rate of data generation have made a centralized assembly of data at one location infeasible. As a consequence, data are increasingly stored in a distributed way, therefore distributed query processing has become an important and challenging problem.

This paper provides a survey of existing approaches for skyline computation in highly distributed environments. We outlined the objectives and the main principles of distributed skyline processing. Most of the existing approaches have been proposed for P2P systems. Therefore, we categorized the existing approaches based on the underlying P2P system and clarified the assumptions of each approach. Furthermore, a comparative performance analysis was provided. We also elaborated on the skyline variants each approach was proposed for and classified the existing approaches based on the skyline variants they can support. We also proposed a taxonomy based on the main principles employed by each approach. Finally, we presented open issues related to distributed skyline computation that have not yet been explored.

References

1. Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. ACM Comput. Surv. **36**(4), 335–371 (2004)
2. Balke, W., Güntzer, U., Zheng, J.X.: Efficient distributed skylining for web information systems. In: Proceedings of International Conference on Extending Database Technology (EDBT), pp. 256–273 (2004)
3. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 421–432 (2001)
4. Chan, C., Eng, P., Tan, K.: Stratified computation of skylines with partially-ordered domains. In: Proceedings of International Conference on Management of Data (SIGMOD), pp. 203–214 (2005)
5. Chan, C., Jagadish, H., Tan, K., Tung, A., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: Proceedings of International Conference on Management of Data (SIGMOD), pp. 503–514 (2006a)
6. Chan, C.Y., Jagadish, H.V., Tan, K.L., Tung, A.K.H., Zhang, Z.: On high dimensional skylines. In: Proceedings of International Conference on Extending Database Technology (EDBT), pp. 478–495 (2006b)
7. Chen, B., Liang, W.: Progressive skyline query processing in wireless sensor networks. In: International Conference on Mobile Ad-hoc and Sensor Networks(MSN), pp. 17–24 (2009)
8. Chen, L., Cui, B., Lu, H.: Constrained skyline query processing against distributed data sites. IEEE Trans. Knowl. Data Eng. (TKDE) **23**(2), 204–217 (2011)
9. Chen, L., Cui, B., Lu, H., Xu, L., Xu, Q.: iSky: efficient and progressive skyline computing in a structured P2P network. In: Proceedings of the International Conference on Distributed Computing Systems (ICDCS), pp. 160–167 (2008)
10. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 717–816 (2003)
11. Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: Proceedings of the International Conference on Distributed Computing Systems (ICDCS), pp. 23–30 (2002)
12. Cui, B., Chen, L., Xu, L., Lu, H., Song, G., Xu, Q.: Efficient skyline computation in structured peer-to-peer systems. IEEE Trans. Knowl. Data Eng. (TKDE) **21**(7), 1059–1072 (2009)
13. Cui, B., Lu, H., Xu, Q., Chen, L., Dai, Y., Zhou, Y.: Parallel distributed processing of constrained skyline queries by filtering. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 546–555 (2008)
14. Dellis, E., Seeger, B.: Efficient computation of reverse skyline queries. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 291–302 (2007)
15. Ding, X., Jin, H.: Efficient and progressive algorithms for distributed skyline queries over uncertain data. IEEE Trans. Knowl. Data Eng. (TKDE) **99** (PrePrints) (2011). (To appear)
16. Fotiadou, K., Pitoura, E.: BITPEER: continuous subspace skyline computation with distributed bitmap indexes. In: Proceedings of International Workshop on Data Management in Peer-to-Peer Systems (DaMaP), pp. 35–42 (2008)
17. Godfrey, P., Shipley, R., Gryz, J.: Maximal vector computation in large data sets. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 229–240 (2005)
18. Hose, K., Lemke, C., Sattler, K.: Processing relaxed skylines in PDMS using distributed data summaries. In: Proceedings of International Conference on Information and Knowledge Management (CIKM), pp. 425–434 (2006)
19. Hose, K., Lemke, C., Sattler, K., Zinn, D.: A relaxed but not necessarily constrained way from the top to the sky. In: Proceedings

of International Conference on Cooperative Information Systems (CoopIS), pp. 339–407 (2007)
20. Huang, Z., Jensen, C.S., Lu, H., Ooi, B.C.: Skyline queries against mobile lightweight devices in manets. In: Proceedings of International Conference on Data Engineering (ICDE), p. 66 (2006)
21. Jagadish, H., Ooi, B., Vu, Q.: BATON: a balanced tree structure for peer-to-peer networks. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 661–672 (2005)
22. Khalefa, M., Mokbel, M., Levandoski, J.: Skyline query processing for incomplete data. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 556–565 (2008)
23. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 275–286 (2002)
24. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. J. ACM **22**(4), 469–476 (1975)
25. Li, H., Tan, Q., Lee, W.: Efficient progressive processing of skyline queries in peer-to-peer systems. In: Proceedings of the International Conference on Scalable Information Systems (Infoscale), p. 26 (2006)
26. Lin, X., Yuan, Y., Wang, W., Lu, H.: Stabbing the sky: efficient skyline computation over sliding windows. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 502–513 (2005)
27. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: the k most representative skyline operator. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 86–95 (2007)
28. Lo, E., Yip, K.Y., Lin, K.I., Cheung, D.W.: Progressive skylining over web-accessible databases. Data Knowl. Eng. (DKE) **57**(2), 122–147 (2006)
29. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: Proceedings of International Conference on Management of Data (SIGMOD), pp. 467–478 (2003)
30. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 15–26 (2007)
31. Pei, J., Jin, W., Ester, M., Tao, Y.: Catching the best views of skyline: a semantic approach based on decisive subspaces. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 253–264 (2005)
32. Preparata, F.P., Shamos, M.I.: Computational Geometry—An Introduction. Springer, Berlin (1985)
33. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), pp. 161–172 (2001)
34. Risson, J., Moors, T.: Survey of research towards robust peer-to-peer networks: search methods. Comput. Netw. **50**(17), 3485–3521 (2006)
35. Rocha-Junior, J.B., Vlachou, A., Doulkeridis, C., Nørvåg, K.: AGiDS: a grid-based strategy for distributed skyline query processing. In: Proceedings of International Conference on Data Management in Grid and Peer-to-Peer Systems (Globe), pp. 12–23 (2009)
36. Rocha-Junior, J.B., Vlachou, A., Doulkeridis, C., Nørvåg, K.: Efficient execution plans for distributed skyline query processing. In: Proceedings of International Conference on Extending Database Technology (EDBT), pp. 271–282 (2011)
37. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications. In: Proceedings of Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM), pp. 149–160 (2001)
38. Sun, S., Huang, Z., Zhong, H., Dai, D., Liu, H., Li, J.: Efficient monitoring of skyline queries over distributed data streams. Knowl. Inf. Syst. **25**, 575–606 (2010)
39. Tan, K.L., Eng, P.K., Ooi, B.C.: Efficient progressive skyline computation. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 301–310 (2001)
40. Tao, Y., Papadias, D.: Maintaining sliding window skylines on data streams. IEEE Trans. Knowl. Data Eng. (TKDE) **18**(3), 377–391 (2006)
41. Tao, Y., Ding, L., Lin, X., Pei, J.: Distance-based representative skyline. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 892–903 (2009)
42. Tao, Y., Xiao, X., Pei, J.: Subsky: efficient computation of skylines in subspaces. In: Proceedings of International Conference on Data Engineering (ICDE), p. 65 (2006)
43. Valkanas, G., Papadopoulos, A.: Efficient and adaptive distributed skyline computation. In: International Conference on Scientific and Statistical Database Management (SSDBM), pp. 24–41 (2010)
44. Vlachou, A., Doulkeridis, C., Kotidis, Y., Vazirgiannis, M.: SKYPEER: efficient subspace skyline computation over distributed data. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 416–425 (2007)
45. Vlachou, A., Doulkeridis, C., Kotidis, Y.: Angle-based space partitioning for efficient parallel skyline computation. In: Proceedings of International Conference on Management of Data (SIGMOD), pp. 227–238 (2008)
46. Vlachou, A., Doulkeridis, C., Kotidis, Y., Vazirgiannis, M.: Efficient routing of subspace skyline queries over highly distributed data. IEEE Trans. Knowl. Data Eng. (TKDE) **22**(12), 1694–1708 (2010)
47. Vlachou, A., Nørvåg, K.: Bandwidth-constrained distributed skyline computation. In: Proceedings of the International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE), pp. 17–24 (2009)
48. Wang, S., Ooi, B., Tung, A., Xu, L.: Efficient skyline query processing on peer-to-peer networks. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 1126–1135 (2007)
49. Wang, S., Vu, Q.H., Ooi, B.C., Tung, A.K., Xu, L.: Skyframe: a framework for skyline query processing in peer-to-peer systems. VLDB J. **18**(1), 345–362 (2009)
50. Wang, J., Wu, S., Gao, H., Li, J., Ooi, B.C.: Indexing multi-dimensional data in a cloud system. In: Proceedings of International Conference on Management of Data (SIGMOD), pp. 591–602 (2010)
51. Wu, P., Zhang, C., Feng, Y., Zhao, B., Agrawal, D., Abbadi, A.: Parallelizing skyline queries for scalable distribution. In: Proceedings of International Conference on Extending Database Technology (EDBT), pp. 112–130 (2006)
52. Xin, J., Wang, G., Chen, L., Zhang, X., Wang, Z.: Continuously maintaining sliding window skylines in a sensor network. In: Advances in Databases: Concepts, Systems and Applications (DASFAA), pp. 509–521 (2007)
53. Yang, B., Garcia-Molina, H.: Designing a super-peer network. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 49–60 (2003)
54. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient computation of the skyline cube. In: Proceedings of International Conference on Very Large Data Bases (VLDB), pp. 241–252 (2005)
55. Zhang, Z., Yang, Y., Cai, R., Papadias, D., Tung, A.: Kernel-based skyline cardinality estimation. In: Proceedings of International Conference on Management of Data (SIGMOD), pp. 509–522 (2009)
56. Zhu, L., Tao, Y., Zhou, S.: Distributed skyline retrieval with low bandwidth consumption. IEEE Trans. Knowl. Data Eng. (TKDE) **21**(3), 384–400 (2009)