

Efficient Skyline Computation in Structured Peer-to-Peer Systems

Bin Cui, *Senior Member, IEEE*, Lijiang Chen, *Student Member, IEEE*, Linhao Xu, Hua Lu, *Member, IEEE*, Guojie Song, Quanqing Xu

Abstract—Peer-to-peer (P2P) computing systems have been thought of as a powerful paradigm for data sharing, and peer-based data management has attracted increasing interest from the database and information retrieval communities recently. In this paper, we investigate the multi-dimensional skyline computation problem on a structured peer-to-peer network. In order to achieve low communication cost and quick response time, we utilize the $i\text{MinMax}(\theta)$ method to transform high-dimensional data to 1-dimensional value and distribute the data in a structured peer-to-peer network called BATON. Thereafter, we propose a progressive algorithm with adaptive filter technique for efficient skyline computation in this environment. We further discuss some optimization techniques for the algorithm, and summarize the key principles of our algorithm into a query routing protocol with detailed analysis. Finally, we conduct an extensive experimental evaluation to demonstrate the efficiency of our approach.

Index Terms—Distributed networks, Database Management, Query processing.

1 INTRODUCTION

IN recent years, peer-to-peer (P2P) computing has become very popular. Due to its desirable features like low deployment cost, but high scalability, flexibility and computing capability, P2P has been widely used in various applications such as resource sharing, scientific computation and distributed data management [5], [8], [16], [27], [29]. Among these, marrying P2P computing with data management technology and exploiting P2P protocol to process various queries are of special interests.

So far, conventional queries including nearest neighbor (NN) queries and range queries [19], [22] have been successfully adapted to the P2P networks. In these precedents, queries against a large number of distributed sites have been facilitated by the flexible and powerful functionalities of the P2P networks. The success of such queries undoubtedly encourages further attempts to adapt other kinds of queries to the P2P networks. Skyline query is one of hot spots in this field.

Given a set of d -dimensional points, a skyline query returns the subset of points that are not *dominated* by any other point. A point p_1 dominates another point p_2 , if p_1 is not worse than p_2 in any dimension and is better than p_2 in at least one dimension. In different contexts, “better” can have different meanings like “larger” values or “shorter” distances.

- Bin Cui, Lijiang Chen, Guojie Song, Quanqing Xu are with School of Electronics Engineering and Computer Science, Peking University, Beijing, China. Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, China.
E-mail: {bin.cui, clj, gjsong, xqq}@pku.edu.cn
- Linhao Xu is with IBM China Research Lab, China.
E-mail: xulinhao@cn.ibm.com
- Hua Lu is with Aalborg University, Denmark.
E-mail: luhua@cs.aau.dk

This research was supported by the National Natural Science foundation of China under Grant No.60603045, National Grand Fundamental Research 973 program of China under Grant No.2004CB318204.

An example of skyline query is shown in Figure 1, where each hotel has two attributes: the price and the distance to the beach. The skyline hotels are those with low prices and short distances to the beach, i.e., the bold dots drawn in the figure. Such hotels are generally preferred by the tourists. Because of the capacity of retrieving interesting points from a multi-dimensional set, skyline queries [4] play an important role in multi-criteria decision making and user preference applications.

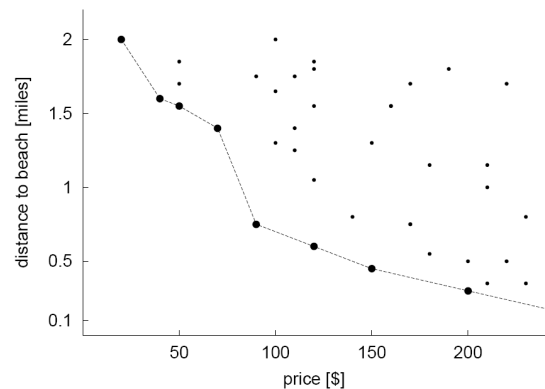


Fig. 1. Skyline of Hotels

Skyline queries have been well studied in the centralized systems [4], [6], [11], [18], [23], [25]. However, little work has concentrated on efficient skyline query processing in P2P networks. The problem is that it is hard to elegantly adopt a centralized index scheme to a distributed network for computing skyline queries. Though well-known DHT-based P2P networks [27], [29] offer advantages over unstructured ones on workload balance and hop-guaranteed object lookup, they destroy data locality and cannot support multi-dimensional search efficiently. Hence, it is difficult to answer a skyline query in the DHT-based P2P systems.

Wu et al. [35] first attempted a progressive processing of skyline queries on a CAN [27] based P2P network. The proposal controls query propagation based on the partial order of CAN's zones. Unfortunately, it focuses on the constrained skyline queries [25], [26], [35], and consequently, its load balancing approach has been designed to solve the workload imbalance caused by the skewed query ranges. Wang et al. [32] proposed the Skyline Space Partitioning (SSP) approach to compute skylines on a structured P2P network called BATON [16], which is a P2P network overlay based on a balanced binary tree structure. The advantage of the SSP is that it relaxes both network organization and skyline query propagation, in comparison with [35]. However, the SSP has two disadvantages. First, at the early stage of each skyline query, the SSP misses a considerable number of peer nodes that have potential skyline points, which renders the result reporting less progressive. Second, during the query processing, the filtering points used by the SSP are not adaptive to the intermediate query results, which degenerates the filtering capacity.

Motivated by these observations, in this paper, we propose an efficient and progressive skyline computation approach in the structured P2P network BATON [16]. We use the BATON as our underlying P2P platform that naturally supports one-dimensional index schemes. Thus we can adopt a centralized index scheme to a distributed P2P network, so as to efficiently perform skyline queries and make the maintenance of the workload balance easier. The algorithm proposed in this paper exploits a data transformation mechanism, the BATON overlay and a filtering based candidate reduction strategy. Specifically, each d -dimensional data point is transformed into a 1-dimensional value and then stored in BATON, which is organized as a balanced binary tree where each tree node corresponds to a real peer. A centralized B-tree based skyline algorithm [30] then can be easily adapted to such a setting. Aside from the progressiveness gained from the data transformation and distribution, the communication cost between peers is cut down by using an adaptive filtering technique. When a peer is processing a query, a portion of its local skyline points are selected as filtering points, which are used to safely prune peers without expected data points and to remove unqualified intermediate answers on later involved peers.

We make the following major contributions in this paper.

- First, based on some good properties of BATON-oriented data preparation, our proposed iSky, together with its deliberately designed algorithms and adaptive filter technique, can efficiently and progressively process skyline queries.
- Second, in order to improve our algorithm to a more scalable solution, we upgrade our ISPeers locating algorithm to a distributed and layered approach, which can reduce the maintenance cost.
- Third, we formalize our query routing protocol and intensively analyze the integrity, correctness and progressiveness of the protocol.
- Forth, we provide extensive experimental results on both real and synthetic datasets to evaluate the performance of the iSky. The results show that the iSky is efficient,

robust and progressive.

This paper extends an earlier version [7] in several substantial ways. First, we provide a more detailed description of related background information. Second, we improve our *ISPeers location* algorithm to a distributed solution, which makes it more reliable and scalable. Third, some optimizations for the *peerSkyline* algorithm are discussed to achieve better performance. Fourth, we extract the key principles from our algorithms and summarize them into the query routing protocol. After that, we give a detailed analysis on integrity, correctness and progressiveness of our proposed query protocol. Finally, we conduct more extensive experiments to evaluate the performance of our proposed approach.

The rest of this paper is organized as follow. Section 2 reviews the related work. Section 3 presents the data preparation of the iSky. Section 4 details the iSky algorithms and describes some improvements as well as the analysis on both processing cost and algorithm correctness. Section 5 reports the experimental results, and finally we conclude the paper in Section 6.

2 RELATED WORK

The related work comes from three areas: conventional skyline computation, structured P2P systems and skyline query processing in P2P systems.

2.1 Conventional Skyline Query Processing

Most conventional skyline query processing approaches focus on the traditional centralized storage, and their algorithms can be divided into two categories. The first category does not require any indexes on the dataset to compute skylines. Borzanyi et al. [4] introduced the skyline query into database systems with algorithms *Block Nested Loop* (BNL) and *Divide-and-Conquer* (D&C). Chomicki et al. [6] proposed a *Sort-Filter-Skyline* (SFS) algorithm as a variant of BNL. Godfrey et al. [11] provided a comprehensive analysis of those aforementioned algorithms without indexing supports, and proposed a new hybrid method *linear elimination sort for skyline* (LESS). Most recently, Bartolini et al. [3] proposed a *Sort and Limit Skyline algorithm* (SaLSa). This algorithm is an alternation of SFS and LESS, which pre-sorts the input data using a monotone limiting function and guarantees that during the query processing the non-fetched data are all dominated by a stop point.

The other category includes those requiring specific indexes for skyline computation. Tan et al. [30] proposed two progressive algorithms: *Bitmap* and *Index*. The former represents points in bit vectors and employs bit-wise operations, while the latter utilizes iMinMax [23] data transformation and B⁺-tree indexing. Kossmann et al. [18] proposed a *Nearest Neighbor* (NN) method. It identifies skyline points by recursively invoking R*-tree based depth-first NN search over different data portions. Papadias et al. [25] proposed a *Branch-and-Bound Skyline* (BBS) method which is IO optimal on R*-tree indexed datasets.

Apart from the centralized contexts, Balke et al. [2] addressed skyline operation in a Web setting where different

dimensions are stored on different web sites. Wu et al. [35] proposed a parallel execution of constrained skyline queries in a shared nothing distributed environment. By using the query range to recursively partition the data region on every data site involved, and encoding each involved (sub-)region dynamically, their method avoids accessing sites not containing potential skyline points and progressively reports correct skyline points. Huang et al. [14] investigated skyline queries on a set of mobile devices communicating via a mobile ad hoc network. Taking into account constraints of unsteady wireless connections and small device capabilities, the authors aim at solving the problems of low communication cost and devices computation cost in a mobile environment. Recently, Cui et al. [9] proposed a filter-based parallel algorithm, called PaDSkyline (Parallel Distributed Skyline query processing), for constrained skyline queries in large-scale distributed environment. It differs from the iSky in that it does not depend on any particular overlay structure.

2.2 Structured P2P Systems

Another area relevant to our work is structured P2P systems. The existing systems can be divided into three classes: DHT based, skip-list based and balanced-tree based. DHT provides a basis for distributing data objects as evenly as possible over peer nodes in the network. The most famous systems in this category are *CAN* [27] and *Chord* [29]. However, this category cannot support complex queries (e.g., similarity queries or skyline queries) efficiently as data locality is destroyed. *Skip graph* [1] and *SkipNet* [12] are two skip-list based systems, which support single-dimensional range queries but not skyline queries that concern multiple dimensions.

BATON (Balanced Tree Overlay Network) [16] is a network overlay based on the well-known balanced binary search tree. It was proposed for facilitating data indexing and data distribution in structured P2P systems. In BATON, each peer maintains a node of the tree and a node connects to other nodes with four different types of network links: a parent link pointing to parent node, children links pointing to child nodes, adjacent links pointing to adjacent nodes (in-order traverse), and neighbor links pointing to selected neighbor nodes at the same level and with a distance equal to a power of two from the node. In BATON, each node, either leaf and internal, is assigned a range of values, which is required to be greater than the range of values managed by its left adjacent node while smaller than the range of values managed by its right adjacent node. That is, if travelling all nodes in BATON network by following adjacent links, data will be listed in ascending order.

Figure 2 illustrates the BATON structure. For example, node *E*'s parent is node *D* and *E* has one child node *F*. The left and right adjacent nodes are also node *D* and *F* respectively since BATON uses in-order traversal to arrange the adjacent relationship among peers. The left routing table of node *E* has one entry that points to node *B* whose children and index range are recorded, while the right routing table of node *E* has two entries that point to nodes *H* and *J* respectively.

Since BATON is a balance-tree-based overlay network, it is considered balanced if and only if the height of its two

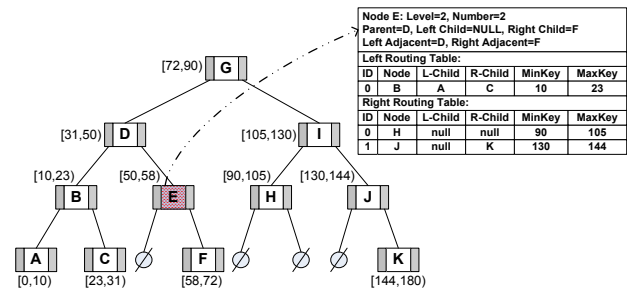


Fig. 2. BATON Structure

subtrees of any node differs by at most one. There are two important properties in BATON. First, a tree is balanced if every node in the tree that has a child also has both its left and right routing tables full. Second, if a node x contains a link to another node y in its left or right routing tables, the parent node of x must also contain a link to the parent node of y unless the same node is parent of both x and y , which gives an efficient way to forward requests among nodes in the network. The following steps take place when a node receives a query. If the search key does not fall into the node's own index range, the query is always forwarded to a node in its left routing table whose upper bound is still greater than the search key, or a node in its right routing tables whose lower bound is still lower than the search key, if such a node exists. Otherwise, the query is forwarded to either its left child/right child or its left adjacent/right adjacent node.

2.3 Skyline Computation in P2P Systems

Recently, skyline query in P2P systems has gained more research efforts. Katja [13] studied how to compute approximate skylines on unstructured P2P networks. Zinn [37] proposed an R-tree liked hierarchical index structure, called QTree, to facilitate computing skyline queries in P2P systems. In the algorithm, leaf nodes compute their local skylines and then forward the results to internal nodes that cluster and reduce the results, until the network central node is covered. Li et al. [20] designed a skyline algorithm on Semantic Small World (SSW) [21], a semantic-clustered P2P overlay network. Vlachou et al. [31] proposed a subspace skyline computation algorithm on top of super-peer networks, called SKYPEER. Normal peers pre-compute their local skylines and then send them to their corresponding super peers. Finally, the local skylines are propagated between super peers until the final skylines are obtained. Wang et al. [32] proposed *Skyline Space Partitioning* (SSP) approach to compute skylines on BATON. SSP partitions a hyperrectangle search ranges (SR) into subSRs based on the history ranges stored in the local split history. When forwarding a query, SSP computes the region number for a search target and compare its position with the regions of the linked nodes. In this way, SSP can effectively control the query forwarding in the query regions, and consequently reduce the number of nodes involved as well as communication cost.

Though [32] is the closest related work to ours in this paper, our iSky has several distinctive features. First, the iSky is

based on data transformation rather than space partitioning. Second, the iSky exploits the BATON protocol better and hence distributes data portions across peers more deliberately. Third, the iSky can progressively return skyline results and efficiently prunes unpromising peer nodes on the fly. The results of our extensive comparative experiments show that the iSky outperforms SSP in most cases.

3 DATA PREPARATION OF ISKY APPROACH

In this section, we describe the data preparation of our iSky approach, and discuss the important properties of the proposed data assignment that will be exploited to design query processing algorithms in Section 4.

3.1 Data Transformation and Assignment

Without loss of generality, we make three assumptions as follows: i) we assume all values are numeric; ii) the range of each dimension is assumed to be normalized into $[0, 1)$; and iii) we assume that on each dimension larger values are preferred by users in their skyline queries. In practice, we can apply our proposal to the cases where smaller values are preferred by adding a negative sign to each value on the relevant dimension. All symbols used throughout this paper are listed in Table 1.

Symbol	Description
d	Data space dimensionality
Ω	Unit hypercube $[0, 1]^d$
P_i	A peer in the P2P system
\mathcal{D}	Dataset stored in the P2P system
\mathcal{D}_{P_i}	Dataset stored on peer site P_i
x	A point in \mathcal{D}
x_i	The i th attribute of point x
x_{max}/x_{min}	Maximum/Minimum attribute value of x
$d_{x_{max}}/d_{x_{min}}$	Corresponding dimension of x_{max}/x_{min}
$S_{\mathcal{D}}$	Skyline of \mathcal{D}

TABLE 1
Symbols Used in Discussions

Let the data space be a d -dimensional unit hypercube $\Omega = [0, 1]^d$. For any data points $u, v \in \Omega$, u dominates v if $u_i \geq v_i$ where $1 \leq i \leq d$ and there exists at least one dimension j such that $u_j > v_j$. Data point u is a skyline point if u is not dominated by any other data point v in the data space Ω .

Proposed for skyline computation in a centralized environment, the Index approach [30] works as follows. Every d -dimensional point x in Ω is transformed to a 1-dimensional value y according to iMinMax [23], and then all transformed values are indexed by a B^+ -tree. The B^+ -tree is used to facilitate identifying those 1-dimensional values whose corresponding d -dimensional points belong to the skyline of Ω .

We employ the same data transformation mechanism. Specifically, for any d -dimensional point $x = (x_1, x_2, \dots, x_d)$, let x_{max} be the largest value among all dimensions and let the corresponding dimension of x_{max} be $d_{x_{max}}$. If there is a tie, e.g., $x_i = x_j = x_{max}$, then we define $d_{x_{max}} = \min(i, j)$. The same applies to $d_{x_{min}}$. By the equation $y = d_{x_{max}} + x_{max}$,

the data point x is transformed to y over a single dimensional range. Note that, the above transformation actually separates the data space into d partitions $[1, 2), [2, 3), \dots, [d-1, d), [d, d+1)$ and offers an order within each partition.

Instead of using a B^+ -tree to index all 1-dimensional values as the Index approach [30], we take advantage of the balanced binary tree structure of the BATON. Specially, all 1-dimensional values (and their corresponding original d -dimensional points) are assigned to all peers according to the BATON protocols such that:

- All 1-dimensional values on each node P_i (no matter internal or leaf node) form a subrange R_i ; and all such subranges are disjoint with each other;
- The union of all subranges is exactly $[1, d+1)$;
- The in-order traversal of the BATON tree is exactly a sequential scan of all subranges in ascending order.

Example 1: Consider the example shown in Figure 3. The left part is a BATON network with 13 peers. The right part shows a dataset with 30 3-dimensional points, which are partitioned into 3 partitions according to the transformation strategy aforementioned. Each partition corresponds to a specific dimension, and is sorted in a non-ascending order of the maximum value in that dimension. Based on the transformation, the original 3-dimensional data space is mapped into the range $[1, 4)$. Then according to the BATON protocols, each peer manages its own subrange as illustrated in the left part of the figure. For example, peer H 's subrange is $[1, 1.375)$ and hence it has three data points dp_8, dp_9 , and dp_{10} . Note that each data point is assigned to a unique peer. \square

3.2 Important Properties

From the example above, we make two important observations. First, some skyline candidate points can be easily found at the peers whose subranges overlap with two adjacent partitions $[k-1, k)$ and $[k, k+1)$ where k is the dimension index. The reason is that such skyline candidate points have the maximal value in each dimension and therefore are probably at the top of each partition. In Figure 3, there are four points (i.e., dp_1, dp_2, dp_{21} and dp_{22}) that have the largest value 0.9 in some dimensions. Among them, it is clear that data points dp_1, dp_2 and dp_{21} are in the skyline, as dp_{22} is dominated by dp_{21} . This implies that some final skyline points can be identified very quickly from some specific peers (e.g., B and G), which have the data points with the largest value among all dimensions.

Second, some peers can be safely pruned. For all data points on a peer P_i , if their maximum value among all dimensions is smaller than the minimum value among all dimensions in a point x from another peer, then x undoubtedly dominates all data points on P_i and hence P_i can be ignored safely. In our running example in Figure 3, the skyline point dp_{21} dominates all data points whose maximum value are smaller than 0.6. If a node aware of dp_{21} is forwarding the skyline computing to peers F, L and M , it can safely skip F and L for their unpromising subranges.

Before summarizing and formalizing the above observations, we introduce two important concepts: *Initial Skyline*

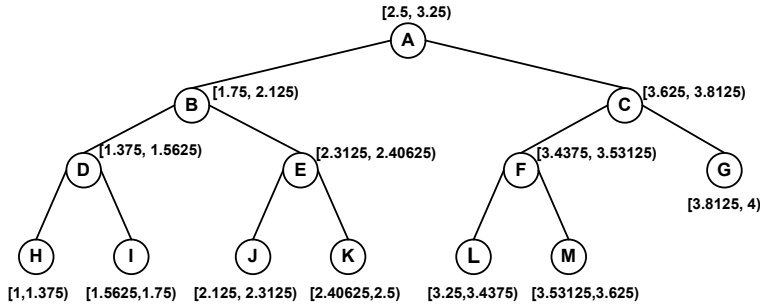


Fig. 3. An Example of Data Transformation and Assignment

Dataset		
Dimension 1	Dimension 2	Dimension 3
dp1: (0.9, 0.8, 0.6)	dp11: (0.7, 0.8, 0.6)	dp21: (0.6, 0.8, 0.9)
dp2: (0.9, 0.5, 0.7)	dp12: (0.3, 0.8, 0.5)	dp22: (0.5, 0.8, 0.9)
dp3: (0.8, 0.2, 0.5)	dp13: (0.6, 0.8, 0.4)	dp23: (0.4, 0.1, 0.8)
dp4: (0.6, 0.5, 0.4)	dp14: (0.5, 0.7, 0.6)	dp24: (0.2, 0.3, 0.7)
dp5: (0.5, 0.4, 0.2)	dp15: (0.3, 0.7, 0.2)	dp25: (0.5, 0.3, 0.6)
dp6: (0.5, 0.3, 0.1)	dp16: (0.1, 0.6, 0.4)	dp26: (0.1, 0.4, 0.6)
dp7: (0.4, 0.1, 0.3)	dp17: (0.2, 0.5, 0.2)	dp27: (0.3, 0.2, 0.5)
dp8: (0.3, 0.2, 0.2)	dp18: (0.2, 0.4, 0.1)	dp28: (0.2, 0.1, 0.4)
dp9: (0.2, 0.1, 0.1)	dp19: (0.2, 0.3, 0.3)	dp29: (0.1, 0.2, 0.3)
dp10: (0.1, 0.1, 0.1)	dp20: (0.1, 0.3, 0.2)	dp30: (0.2, 0.2, 0.3)

Peers and Candidate Skyline Points as follows.

Definition 1: Let m be $\max_{x \in \mathcal{D}}(x_{max})$, we define Initial Skyline Peers \mathcal{P} and Candidate Skyline Points \mathcal{M} as

$$\mathcal{P} = \{P_i \mid \exists x \in \mathcal{D}_{P_i} \text{ such that } x_{max} = m\}$$

$$\mathcal{M} = \{x \mid x \in \mathcal{D}_{P_i} \wedge x_{max} = m\},$$

where $\mathcal{D}_{\mathcal{P}} = \cup_{P_i \in \mathcal{P}} \mathcal{D}_{P_i}$.

From the above definition, we know that the candidate skyline points are these data points with the largest value among all dimensions, since intuitively any of them is unlikely to be dominated. The initial skyline peers are these peers that contain the candidate skyline points.

Now we can summarize our observations as follows. Theorem 1 tells that the skyline of the candidate skyline points is the subset of final skyline. Theorem 2 guarantees that the candidate skyline points can be found at these peers whose subranges overlap with arbitrary two adjacent partitions. Theorem 3 shows that if we use some known skyline points as filters, some specific peers can be pruned away safely without consideration in further processing.

Theorem 1: If $S_{\mathcal{D}}$ and $S_{\mathcal{M}}$ are the skylines of \mathcal{D} and \mathcal{M} respectively, then $S_{\mathcal{M}} \subseteq S_{\mathcal{D}}$ and $S_{\mathcal{M}} \neq \emptyset$.

Proof According to the definition of \mathcal{M} , for any $x \in S_{\mathcal{M}}$, x is not dominated by any other point in \mathcal{M} . For all $x' \in \mathcal{D} - \mathcal{M}$, we define $m' = \max(x'_{max})$. As $m > m'$, x is not dominated by any $x' \in \mathcal{D} - \mathcal{M}$. Therefore, x is not dominated by any other point in \mathcal{D} , i.e., $x \in S_{\mathcal{D}}$. Thus, we have $S_{\mathcal{M}} \subseteq S_{\mathcal{D}}$. Since \mathcal{M} is not empty, hence $S_{\mathcal{M}} \neq \emptyset$. \square

Theorem 2: Let m be $\max_{x \in \mathcal{D}}(x_{max})$ and $\mathcal{M} = \{x \mid x \in \mathcal{D} \wedge x_{max} = m\}$. Suppose \mathcal{D} is divided into d partitions $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_d$ by the $iMinMax(\theta)$ transformation, and the 1-dimensional range at each peer \mathcal{P}_i is denoted as $[min_i, max_i]$, where $min_i = \min_{x \in \mathcal{D}_{P_i}}(d_{x_{max}} + x_{max})$ and $max_i = \max_{x \in \mathcal{D}_{P_i}}(d_{x_{max}} + x_{max})$. Let \mathcal{R} be the set of all peers satisfying $(min_i < k + 1 \leq max_i)$ and $\mathcal{D}_{\mathcal{P}} = \cup_{P_i \in \mathcal{P}} \mathcal{D}_{P_i}$, where $1 \leq k, i \leq d$. Then, $\mathcal{M} \subseteq \mathcal{D}_{\mathcal{P}}$.

Proof According to the definition of \mathcal{M} , for any $x \in \mathcal{M}$, we have $x_k = x_{max} = m$ and the transformed value $y = d_{x_{max}} + x_{max} = k + m$, indicates $k < k + m \leq k + 1$, where

k is the dimension that the largest value x_{max} belongs to. On the other side, according to the definition of \mathcal{P} , for any $P_i \in \mathcal{P}$, if the condition $(min_i < k + 1 \leq max_i)$ is satisfied, then we must have $min_i \leq k + m \leq k + 1 \leq max_i$, which indicates $y \in [min_i, max_i] \in \mathcal{D}_{P_i}$. Thus, $\mathcal{M} \subseteq \mathcal{D}_{\mathcal{P}}$. \square

Theorem 3: Given a peer P_j and its dataset \mathcal{D}_{P_j} , let $x_{max} = \max_{x \in \mathcal{D}_{P_j}}(x_i)$ and $x'_{min} = \min_{x' \notin \mathcal{D}_{P_j}}(x'_i)$. If $x'_{min} > x_{max}$, then P_j can be pruned away.

Proof For all $x' \notin \mathcal{D}_{P_j}$ and all $x \in \mathcal{D}_{P_j}$, $x'_i \geq x'_{min} > x_{max} \geq x_i$ indicates $x'_i > x_i$, where $1 \leq i \leq d$. This means that any $x' \notin \mathcal{D}_{P_j}$ dominates all points $x \in \mathcal{D}_{P_j}$. Thus, P_j can be pruned away safely. \square

4 ALGORITHMS OF ISKY APPROACH

Based on the data preparation as described in Section 3, we present the detailed algorithms of our iSky approach in this section. We call a peer issuing a skyline query q *query originator*, denoted as P_{org} . Any other peer that processes query q is called a *processing peer*. The objective of our algorithms is to minimize the number of processing peers, and to make local skyline computations fast on those processing peers. The algorithms of our iSky approach take advantage of those properties in Section 3.2 to compute skyline efficiently and progressively on the BATON network. We sequentially present the algorithms of the three aspects of the iSky approach: how to locate the initial skyline peers; how to conduct skyline computing with adaptive filtering on a processing peer; how to manage the overall query processing on P_{org} .

4.1 Locating Initial Skyline Peers

4.1.1 Locating ISPeers Via Root Node

According to Theorem 1, the *initial skyline points (ISP)*, denoted as $S_{\mathcal{M}}$, is part of the final skyline and can be obtained from the candidate skyline points \mathcal{M} . According to Theorem 2, we can quickly obtain the candidate skyline points \mathcal{M} from the initial skyline peers (ISPeers) \mathcal{P} , by checking if their subranges overlap with two adjacent partitions. Clearly, the initial skyline points can be computed quickly and delivered to the end user immediately. Therefore, the first step of the iSky approach is to identify the initial skyline peers and compute the initial skyline points from the candidate skyline points.

To determine if a peer belongs to the initial skyline peers, each peer P_i checks whether its subrange overlaps with two adjacent partitions, which is shown in Algorithm 1. P_i first computes $\max_{x \in \mathcal{D}_{P_i}}(x_{max})$ (line 1). Subsequently, the new index boundary \min_i and \max_i is updated (lines 2-3). Then, for all partitions $[k, k+1)$, P_i checks if the condition ($\min_i < k+1 \leq \max_i$) is satisfied, where $1 \leq k \leq d$ (lines 4-5). If so, P_i reports its identifier to the root of the BATON (lines 6-7).

Algorithm 1: locateISPeers (root)

```

1 compute  $\max_{x \in \mathcal{D}_{P_i}}(x_{max})$ ;
2  $\min_i = \min(d_{x_{max}} + x_{max})$ ;
3  $\max_i = \max(d_{x_{max}} + x_{max})$ ;
4 for ( $k = 1; k \leq d; k++$ ) do
5   if ( $\min_i < k+1 \leq \max_i$ ) then
6     report  $id$  to  $root$ ;
7   break;
```

From Theorem 2, we discover that the number of the initial skyline peers equals to the dimensionality of the data space. For example, the initial skyline peers include three peers A , B and G as shown in our running example in Figure 3. This number is quite small compared with the whole peer population. Therefore, we use the root node of the BATON to maintain the initial skyline peers. The root node is responsible for broadcasting the information of the initial skyline peers to all nodes in the network, when any update happens. Thus, each node knows the initial skyline peers of the whole network. Here we argue that most of nodes in a structured P2P network are stable and seldom live in a transient way like unstructured P2P systems [5], and therefore use the root node to maintain the initial skyline peers is applicable.

4.1.2 Improvement of ISPeers Locating

In the previous case, we use the root node as an organizer to maintain the initial skyline peers (ISPeers) list and broadcast that list to every peer in the P2P network. With this method, any peer can find the ISPeers immediately when it issues a query. However, it has two problems. First, the root node could become a hot spot when the network changes frequently. Second, once an update of the ISPeers occurs, every peer should be informed, which results in a lot of communication cost. In order to solve these problems, we improve our ISPeers locating mechanism to a more reliable and less costly solution. We exploit a distributed and layered approach to maintain the ISPeers list.

In this approach, we use the ISPeers themselves to maintain the ISPeers list. From Section 4.1 and Theorem 2, we know that the ISPeers' subranges overlap with two adjacent partitions. In other words, the boundary of each partition is just in one and only one ISPeer's subranges. Therefore, even though we do not know who are the ISPeers, we can find them by locating each partition's boundary indirectly. At the beginning, a node who identifies itself to be a ISPeer, will create a ISPeers list and add itself in that list. Then it sends a message to inform the other ISPeers to add it in their lists. In order to reach all other ISPeers, this node locates the boundary of each partition from 2 to $d+1$ (except itself). So do other

ISPeers. With this approach, each update of the ISPeers incurs $O(d \times \log N)$ location cost and $O(d)$ message cost compared with $O(N \times \log N)$ location cost and $O(N \times |\mathcal{P}|)$ message cost of the previous approach, where N is the total number of peers in the network, d is the number of dimensionality and $|\mathcal{P}|$ is the size of the ISPeer list. The node who is going to issue a query, will locate its nearest ISPeer node and then fetch the ISPeers list from it.

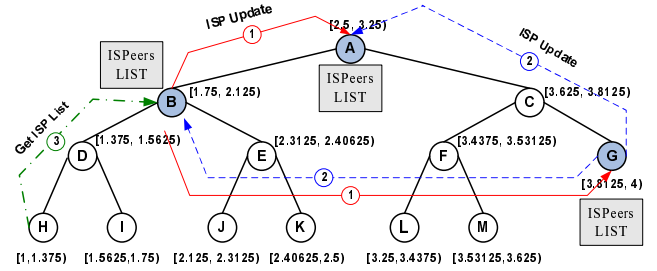


Fig. 4. Enhanced Mechanism for ISPeers Maintaining and Locating

Example 2: Figure 4 illustrates the improved ISPeers list maintaining and locating approach. Nodes A , B and G are the ISPeers and each of them maintains a ISPeers list. The steps ① and ② illustrate the ISPeer updating operation of the nodes B and G respectively. They inform all other ISPeers to update their ISPeers list. Step ③ shows how a node retrieves the ISPeer list when issuing a skyline query.

4.2 Adaptive Skyline Filtering

According to Theorem 3, we can use the minimal value among all dimensions of a data point as a filter, to prune any peer whose maximal value among all dimensions of all its data points is smaller than the filter. Now the problem is how to find such a filter. Since the initial skyline points are computed from the candidate skyline points in the first phase, using values in these skyline points as filters is a good idea. We should notice that, according to Theorem 1, the initial skyline points must be not empty.

For each initial skyline point, we first find its minimum value in all dimensions. Then, we select the maximum value from all minimum values of all initial skyline points as the initial filter. Thus, we define the filter SF_{global} as

$$SF_{global} = \max_{x \in \mathcal{S}_{\mathcal{M}}}(x_{min})$$

where $\mathcal{S}_{\mathcal{M}}$ is the initial skyline points. Subsequently, SF_{global} is sent out with the skyline query from the query originator P_{org} to prune unpromising peers.

However, if SF_{global} cannot prune a peer, we need to perform a local skyline computation. In such a case, the filtering capability of SF_{global} is unlikely to be strong. Therefore, we need another skyline filter to remove all local skyline points that are dominated by some final skyline points. We use *dominating region* (DR) [14] to determine which initial skyline point has the largest dominating capability, and denote such a skyline point as SF_{local} . The query originator P_{org} also sends the SF_{local} out with the skyline query.

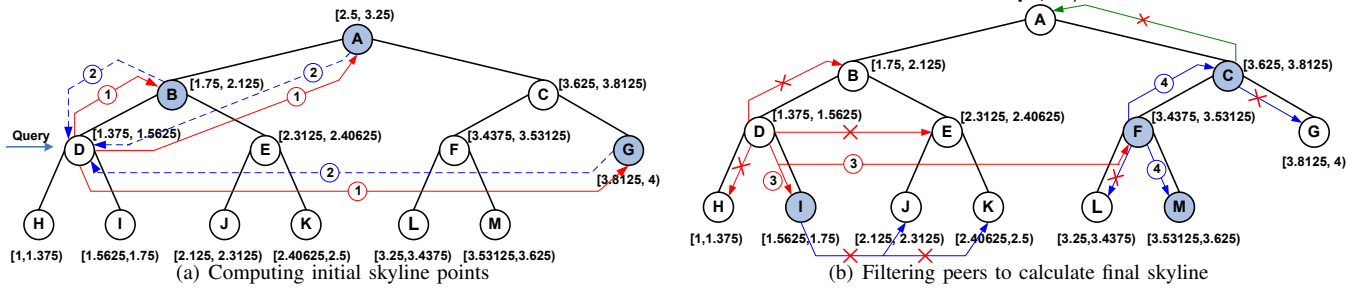


Fig. 5. An example of the iSky approach

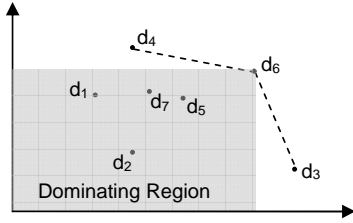


Fig. 6. Dominating region

Figure 6 illustrates the dominating region of the point d_6 (the shaded area). The given dominating region can be used to prune d_1 , d_2 , d_5 and d_7 , as the four points fall into the area. Specifically, given $\Omega = [0, 1]^d$ and a point $x = (x_1, \dots, x_d)$, the DR is defined as $DR = (x_1 - 0) \times \dots \times (x_d - 0)$. Therefore, the selected SF_{local} should have the largest DR value, which has higher probability of dominating more points than other skyline candidates. For example, dp_1 , dp_2 , dp_{11} and dp_{21} are the initial skyline points (see details in Example 3), and dp_1 and dp_{21} have the largest DR. Thus we use them as SF_{local} as they can prune more points, and hence more peers probably.

4.2.1 Skyline Computing Algorithm with Filter

Given a skyline query q and a processing peer P_i , P_i processes q using the *peerSkyline* algorithm shown in Algorithm 2. P_i first uses SF_{global} to examine itself. If P_i is pruned then it just forwards the query to the unchecked peers in its routing table (line 3); otherwise, SF_{local} is used to do the local skyline filtering (line 6). For each data point x , if it cannot be dominated by SF_{local} , then we use the local skyline candidate $result$ to do filtering (line 9). If $result$ is empty we add x into $result$ (line 7). Otherwise, for each skyline candidate r in $result$, we check the relationship between r and x (lines 10-18). If r dominates x the loop breaks (line 10). If x dominates r then r is removed from $result$ (line 12). Or, if r and x cannot dominate each other, x is added into $result$ and two filters SF_{global} and SF_{local} are updated if necessary (lines 14-18). Finally, the query q is forwarded to all the unchecked nodes in P_i 's routing table with the updated filters, and the local skyline result is returned to the query originator (lines 19-21).

4.2.2 Discussion

In Algorithm 2, we first compute the local skyline, and then check updates of the filters, SF_{global} and SF_{local} . Finally,

we forward the query to other peers with new filters. From above processes, we find that the forwarding operation must be performed after the local computation. But if this two operations can parallel, the query will reach the underlayer peers quickly and the query processing time will be reduced. In order to parallel these two operations, we must pre-compute those two filters, SF_{global} and SF_{local} , in each peer. In the implementation, we pre-compute those filters by identifying the *domination region* (DR) of each point preliminarily, select the data point with the maximum DR as SF_{local} and choose the x_{min} of this data point as SF_{global} , if they are better than the original ones. Actually, we only need to process this once and let each peer incrementally maintain its SF_{local} when it performs a skyline computation.

Algorithm 2: *peerSkyline* ($q, SF_{global}, SF_{local}$)

```

1 result = ∅; DRmax = 0; idx = 0;
2 if (SFglobal ≥ maxx ∈ DPi(xmax)) then
3   forward q to all the unchecked peer in routing table;
4 else
5   foreach x ∈ DPi do
6     if (dom(SFlocal, x) == false) then
7       if (result == ∅) then result = result ∪ x;
8       else
9         foreach (r ∈ result) do
10          if (dom(r, x)) then break;
11          else
12            if (dom(x, r)) then
13              result = result - r;
14            else
15              result = result ∪ x;
16              if DRx > DRmax then
17                DRmax = DRx;
18                idx = x;
19              if xmin > SFglobal then
20                SFglobal = xmin;
21 if DRmax > SFlocal then SFlocal = idx;
22 send ⟨q, SFglobal, SFlocal⟩ to all the unchecked nodes in
    routing table;
23 return ⟨result⟩;
```

4.3 The Overall Algorithm on Originator

Now we are ready to present the overall algorithm executed on the skyline query originator peer P_{org} . To ease the presentation, we simply name this algorithm *iSky*, which is shown

in Algorithm 3. P_{org} first sets $result$ as empty and sends its query q to all initial skyline peers in \mathcal{P} (lines 1-2). Once all skyline results from initial peers have been received, they will be merged into $result$ set (lines 3-4). After that, we generate two filters SF_{global} and SF_{local} , and the query q is broadcasted to all the nodes in the routing table of P_{org} , such as the parent node, child nodes and neighbor nodes. For each peer who receives the query, it calls the *peerSkyline* algorithm shown in Algorithm 2, returns its local skyline points to originator and forwards the query to the unchecked peers in its own routing table if necessary (lines 11-12). Finally, the query originator merges all the skyline candidates from the P2P system and returns the answer to the end-user (line 13).

Algorithm 3: *iSky* ()

```

1  $result = \emptyset; DR_{max} = 0; idx = 0;$ 
2 foreach ( $P \in \mathcal{P}$ ) do send  $q$  to  $P$ ;
3 receive the results from all initial skyline peers;
4  $result = \text{merge}(\text{received skyline points});$ 
5 foreach ( $r \in result$ ) do
6   if  $DR_r > DR_{max}$  then
7      $DR_{max} = DR_r;$ 
8      $idx = r;$ 
9  $SF_{local} = idx;$ 
10  $SF_{global} = \max_{r \in result}(r_{min});$ 
11 foreach  $P$  in  $P_{org}$ 's routing table do
12    $\lfloor$  call peerSkyline( $q, SF_{global}, SF_{local}$ );
13  $result = \text{merge}(\text{received skyline points});$ 

```

Example 3: Figure 5 demonstrates an example on how the *iSky* approach works. Suppose that we have 13 travel agencies in a city and the dimensions of the dataset shown in Figure 3 represent for *star level*, *customer review* and *discount*, respectively. Note here the original data has been normalized into the unit space $[0, 1]^3$. A customer in agency D is looking for a popular hotel with high star, better review and high discount. Therefore, agency D issues a skyline query q to retrieve all candidates. In step ①, D sends query q to the initial skyline peers (i.e., agencies) A , B and G . Then, in step ②, agencies A , B and G return their local skylines $\{dp_1, dp_2, dp_{11}, dp_{21}\}$ to D and D computes the initial skyline points $\{dp_1, dp_2, dp_{21}\}$ on the returned results, as depicted in Figure 5(a). After that, Figure 5(b) illustrates how to use filters to calculate the final skyline. Specifically, in step ③, D broadcasts q and two filters $SF_{global} = 0.6$ and $SF_{local} = \{dp_1, dp_{21}\}$ to agencies I and F , without considering agencies B , H and E . This is because agency B has already been visited and the max_i of agencies H and E is smaller than SF_{global} . Finally, in step ④, agency F sends q to agencies M and C , as they cannot be pruned by SF_{global} . Notice that, in the course of query forwarding, the *iSky* can guarantee all promising agencies are visited and prune all unpromising agencies on the fly. For example, unpromising agencies J and K can be pruned by agency I since they are in I 's routing table. \square

4.4 Analysis

In this section, we formalize the query and our query routing protocol. Based on the formalization of the routing protocol,

we intensively discuss the integrity, correctness and progressiveness of the *iSky* algorithm.

4.4.1 Query Routing Protocol

Without loss of generality, we formalize the skyline query as follow:

Definition 2: Let QID be the identifier of the query, Org be the identifier of the originator and SF_{local} , SF_{global} be the two filters, we definite query Q as

$$Q = \langle QID, Org, SF_{global}, SF_{local} \rangle.$$

In the query processing, the peers use QID to identify an individual query as well as the results to that query. They use Org as a destination while routing the skyline results. SF_{local} and SF_{global} are filters used to filter local skyline and prune unqualified peers. Whereafter, we abstract our proposed skyline computation algorithms and summarized them as the following Protocol:

Query Routing Protocol

- Step 1: the originator routes Q to the initial skyline peers;
- Step 2: the peer P , who is processing Q , routes Q to all neighbors in its routing table that satisfies the condition

$$P'.x_{max} > SF_{global}$$

where P' is a peer in P 's routing table;

- Step 3, the peer returns results to Org directly if it has skyline points.
- Step 4, the peer drops Q if it had been processed already.

4.4.2 Analysis of Query Routing Protocol

The execution of the *iSky* algorithm follows the previous query routing protocol. In what follows, we will discuss the integrity, correctness and progressiveness of our proposed skyline computation protocol.

Analysis of Integrity

We first discuss the integrity of the skyline results. The integrity of the results means that the final result of every skyline query should contain all skyline points in the P2P network. Our approach guarantees the integrity property because of two reasons. First, the BATON overlay guarantees the reachability of each peer in the network. The BATON protocol makes sure that all peers in the overlay are connected via their routing tables, which ensures that each skyline query will be delivered to the promising peers according to the BATON protocol. In other words, all data points at all promising peers can be checked and no skyline points are missed. Second, according to Theorem 3, all unqualified peers that are pruned by the query routing protocol, do not affect the correctness of the final skyline results. Therefore, the integrity of the skyline results of the query routing protocol is proved.

Analysis of Correctness

With the analysis of Integrity, we guarantee that no individual skyline point is missed. To confirm the correctness of the query protocol, we still need to show we do not have any false positive in our query result. In the protocol, we utilize filter

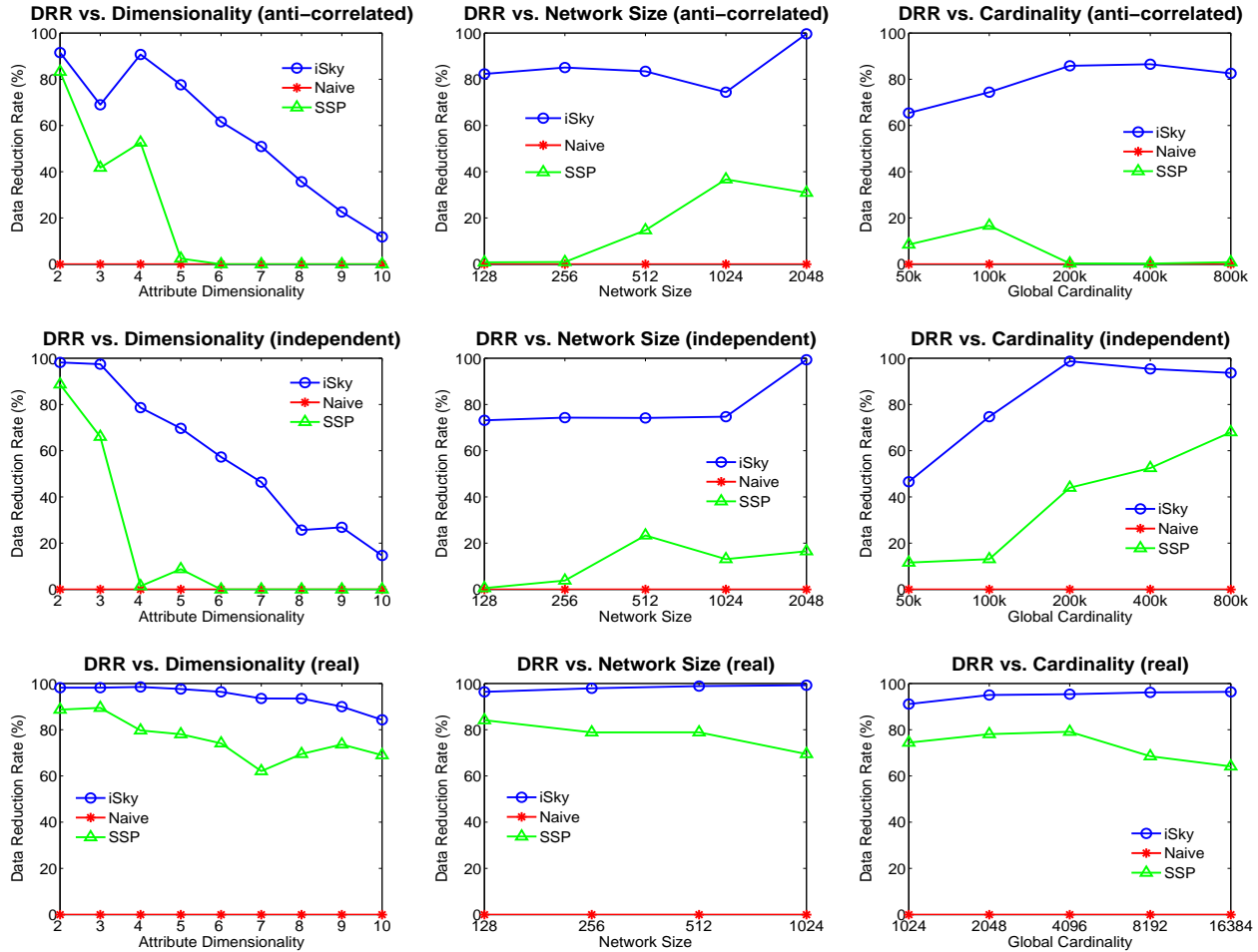


Fig. 7. Data Reduction Rate on Three Datasets

SF_{local} to filter the non-skyline points at each peer. According to Theorem 1 and 2, we have proved that the skyline points retrieved from the initial skyline peers are indeed the global skyline points. These global skyline points are then used as filter SF_{local} , and every returned skyline candidate point will be checked by the global skyline points in the query originator, which rules out any false positive from the final result.

Analysis of Progressiveness

The progressiveness of the query routing protocol is reflected in the following aspects. First, a large number of the final skyline points can be retrieved on the fly, since only one hop query is needed for the initial skyline peers. As we have proved in Section 3.2, these skyline points can be delivered to the end-users immediately which gains a quick response. Second, the self-adaptive filter technique guarantees the minimal number of peers to be involved into the skyline computation, which saves a lot of bandwidth consumption and computational resources. Third, the directly-result-returning behavior assures continuous and quick query response. Fourth, the termination condition in the final step of the protocol can guarantee that no unpromising nodes are visited.

5 PERFORMANCE STUDY

We study the performance of our *iSky* approach on the BATON network with respect to three aspects: dimensionality, network size and cardinality. We compare our *iSky* approach with the SSP method [32] and the adapted naive approach based on the centralized Index skyline algorithm [30]. In the naive approach, we distribute all data points randomly into all peers on the BATON network. Upon receiving a query request from the query originator, each peer computes its local skyline, returns the result back to the originator and sends the query to all its neighbors directly. As the SSP yields significantly better performance than the distributed skyline query algorithm DSL [35], we omit the comparison with the DSL for the clarity of presentation.

Parameter	Setting
Dimensionality	2, 3, ..., 6, ..., 10
Number of peers	$2^7, 2^8, \dots, 2^{10}, 2^{11}$
Cardinality of each peer	50, 100, 200, 400, 800
Number of queries	1000

TABLE 2
Parameters Used in Experiments

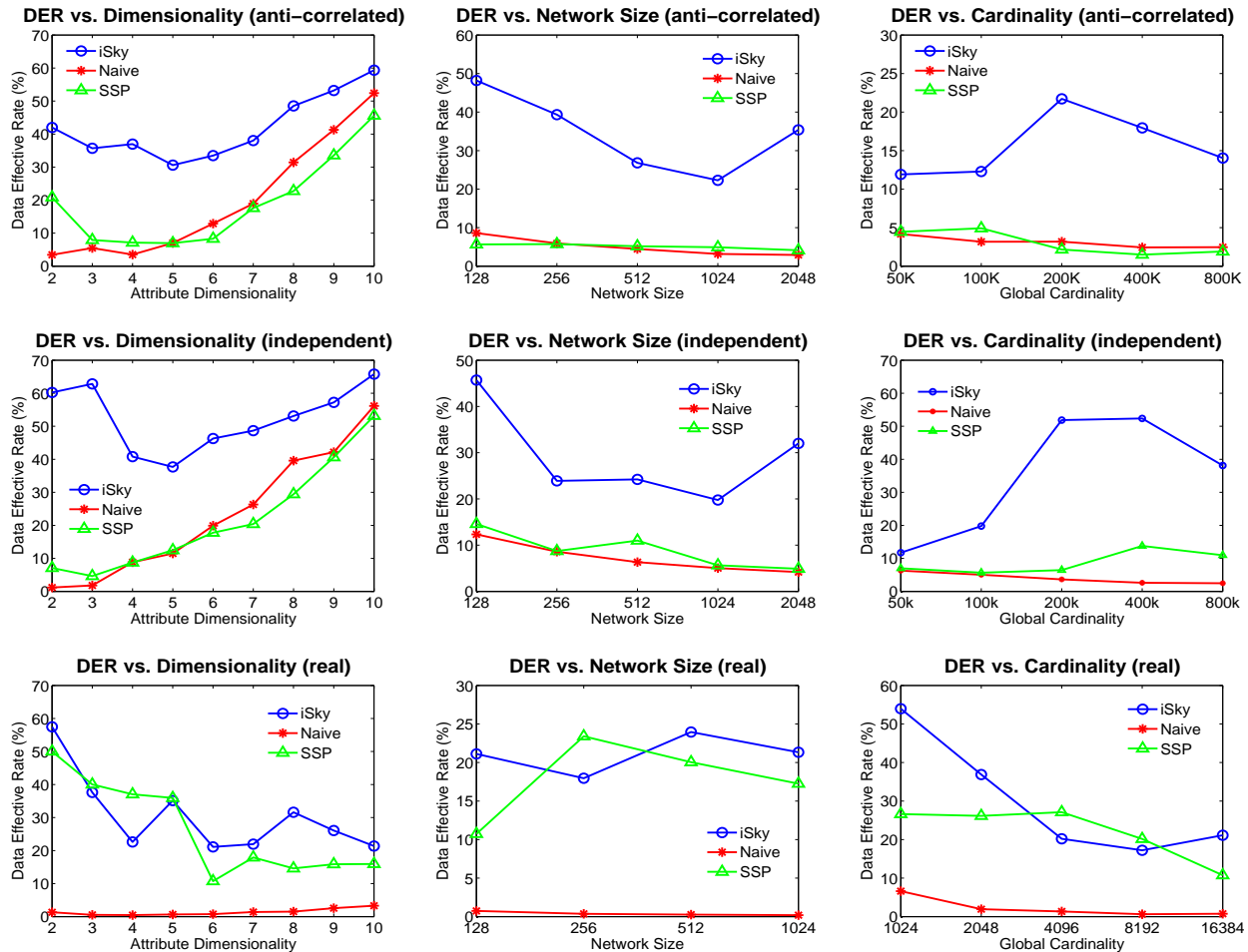


Fig. 8. Performance on Data Effective Rate

We consider the following performance metrics: data reduction rate, communication cost, number of processing nodes and response time. They are measured through the simulation experiments on a Linux Server with 4 Intel Xeon 2.80GHz processors and 1.0GB RAM. All experiments are repeated 10 times, each of which issues 1000 skyline queries from a randomly selected originator node, and the average results are reported. We use three kinds of different datasets: a real dataset of NBA players' season statistics from 1949 to 2003¹, which has 16,644 records and approximates a correlated data distribution, and two synthetic datasets of both independent and anti-correlated distribution respectively. The parameters used in the experiments are listed in Table 2. Unless stated explicitly, the default parameter values, given in bold, are used.

5.1 Data Reduction Efficiency

We first study the efficiency of the *filter points* in the local skyline computing in terms of the data reduction rate DRR [14], the proportion of the data points reduced by the filter points to the number of points in the unreduced skyline. This is a metric to measure the power of the filter technique. The DRR

is defined as

$$DRR = \frac{\sum (|unredSK_i| - |redSK_i| - 1)}{\sum |unredSK_i|}$$

where the sum is calculated over all processing peer nodes except those initial skyline peers.

Figure 7 shows how the DRR varies with respect to dimensionality, network size and cardinality on all three datasets, respectively. We compare the iSky with the SSP and the naive approach. We can see that the iSky is better than the two baselines, because of its adaptive filter capability, as it can effectively prevent the unqualified local skyline from being transmitted within the network. Particularly, when the network size or the cardinality increases, the proposed filtering capability becomes more conspicuous. However, the performance of the iSky drops when the dimensionality rises. The reason is that the dominating region or filtering capability becomes worse due to the curse of dimensionality [33]. Even in such a situation, the iSky still outperforms the SSP. Furthermore, we can discover that the SSP approach has a marked deficiency when the dimensionality becomes high (e.g., 5 for anti-correlated dataset, 4 for independent dataset and 6 for real dataset). This is because the SSP does not employ any filtering

1. <http://databasebasketball.com>

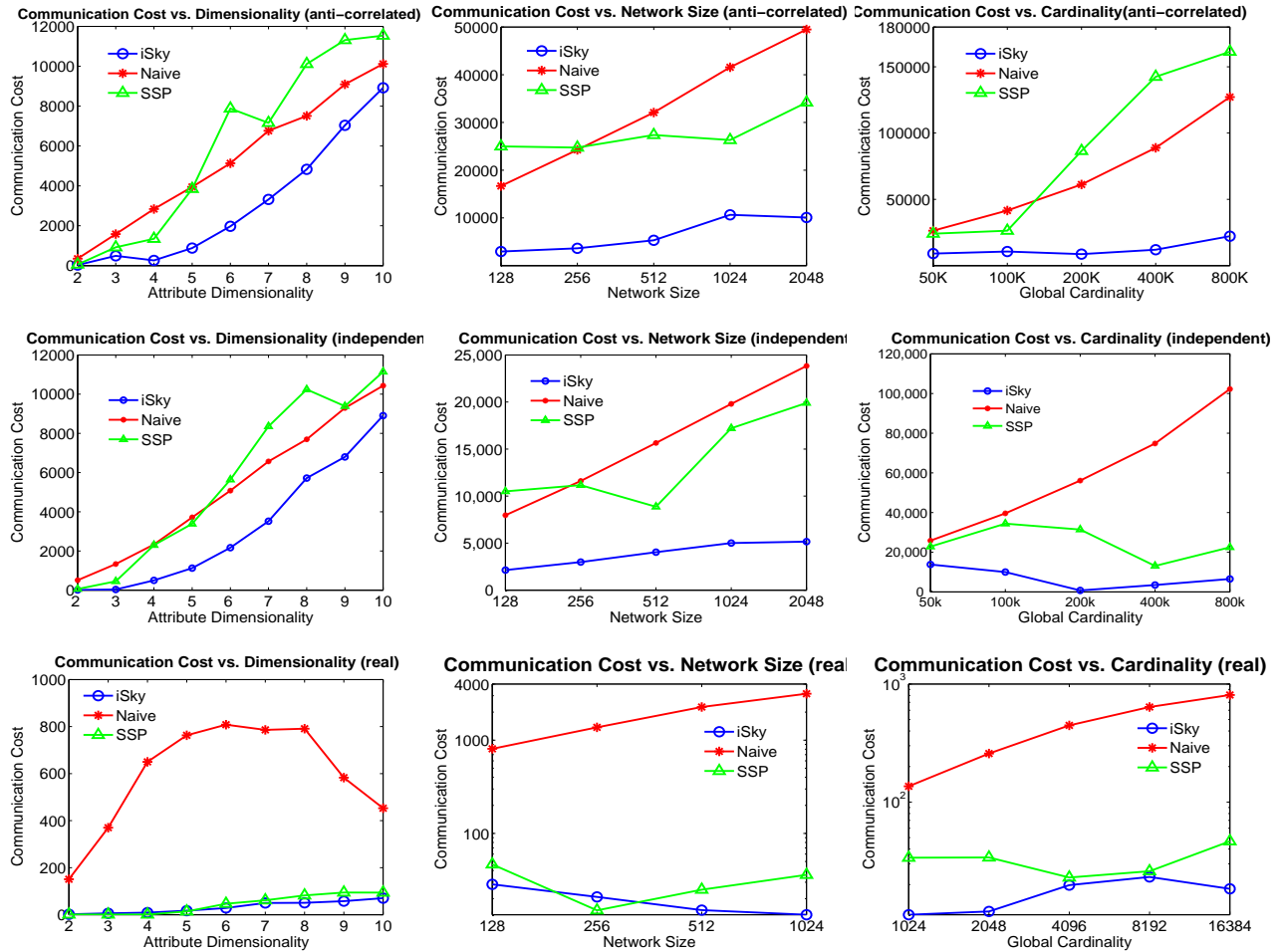


Fig. 9. Performance on Communication Cost

techniques as the iSky does.

We also notice that the iSky and SSP perform better in the real dataset than in the other two datasets. There are two reasons. First, the real NBA dataset has a correlated data distribution, in which the data values are highly duplicated in many dimensions. Therefore, a skyline point may filter a large amount of points with identical values. Second, since the real dataset has only 16,644 records, the total number of skyline points is limited. Due to the above reasons, all the algorithms perform better in the real dataset than in the other two datasets. Overall, the iSky still outperforms the SSP and native solution in this situation.

5.2 Data Transmission Efficiency

We proceed to study the data transfer efficiency of our proposed method. For this purpose, we propose the performance metric Data Effective Rate (DER). It reflects the efficiency of data transmission from the processing peers to the query originator. i.e., how much data transmitted really contributes to the final skyline. Naturally, the closer the total number of local skyline points transmitted is to the number of final skyline points, the more efficient is the data transmission. The DER

is defined as

$$DER = \frac{|finalSK|}{\sum |transmittedSK_i|}$$

where the sum is calculated over all processing peer nodes but those initial skyline peers.

Figure 8 shows the results of the DER of all three approaches. The figures shown in the left are the experiment results of different dimensionality in three datasets. In the first two datasets, when the dimensionality increases, the DER of all three approaches rises. This is because, when dimensionality increase, the final skyline contains more skyline points and therefore more skyline candidates are transmitted for the final skyline computation. This is another evidence for why the data reduction rate becomes lower as dimensionality increases. The story changes for the real dataset. We know that in the real dataset, data values are highly duplicated in many dimensions. When more dimensions are taken into consideration, more points with identical value appear. Therefore, the total amount of skyline points increases, and hence more unqualified candidates need to be pruned at last.

For an overall comparison, the iSky outperforms the baselines in all cases. Because of the goodness of the initial skyline

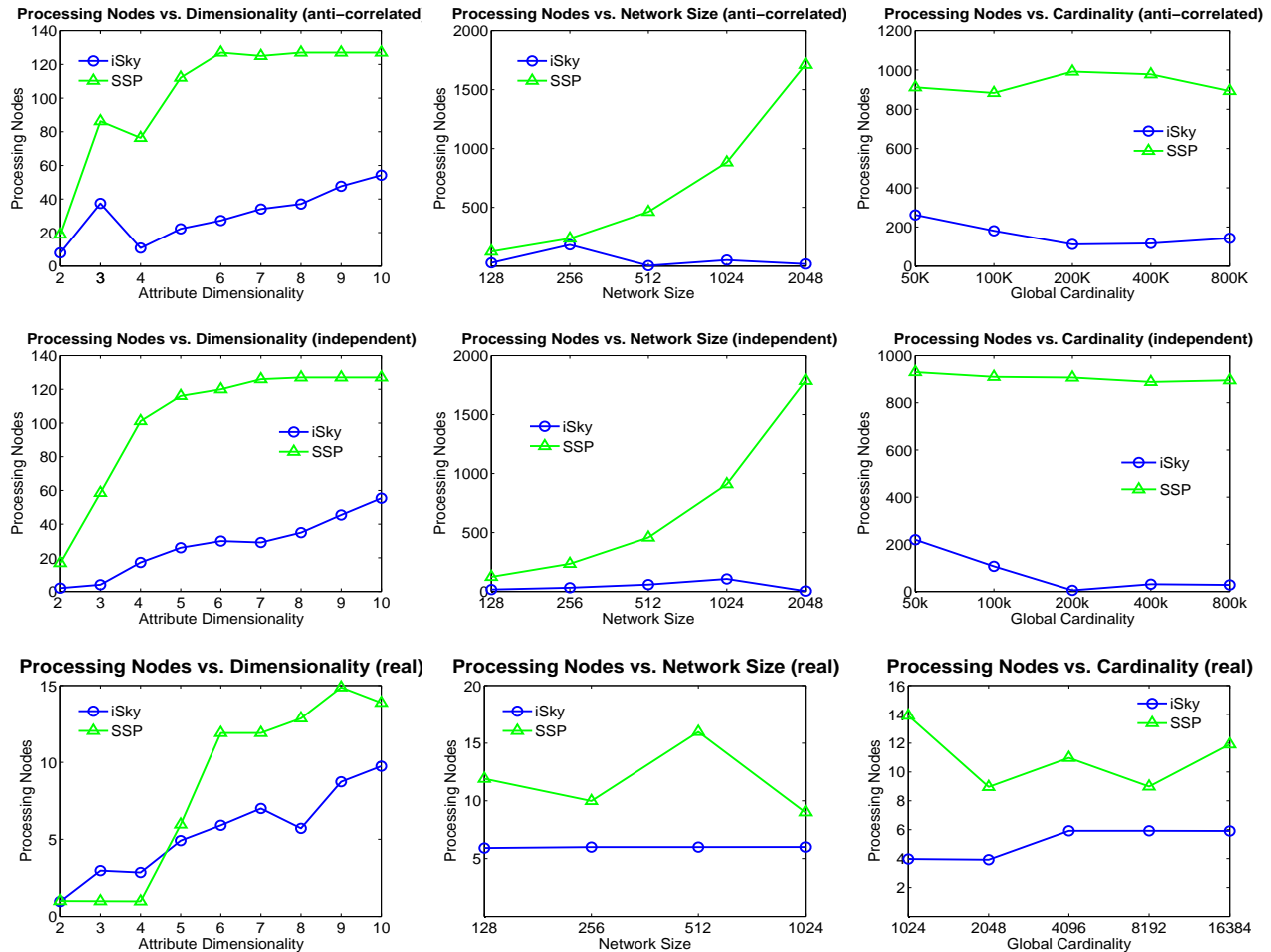


Fig. 10. Performance on Processing Nodes Involved

points in the iSky, the SF_{global} selected from them is very effective to filter unpromising peers. On the other hand, the SF_{local} filters in the iSky are selected on the fly at processing peers, and therefore they are effective to identify unqualified local skyline points. As a result of these two factors together, a considerable portion of data points transmitted through the network are in the final skyline. Thus, the DER of the iSky is larger than those of the SSP and native approach.

5.3 Communication Cost

The next performance metric we consider is the communication cost. The communication cost is measured by the number of messages transferred for the skyline computation. From Figure 9, we can see that the communication cost of the iSky is stable with respect to the increase of network size and cardinality. But it is sensitive to dimensionality change, because the filtering capability of the iSky decreases when the number of dimensions increases. That is, more nodes the skyline computation involves, more messages the query processing needs. However, the iSky outperforms both SSP and naive approach. The naive approach deteriorates notably when the scale increases, as it does not employ any optimization strategies in query processing. As the network size increases,

the SSP incurs a higher communication cost compared with the iSky. The reason is that the SSP lacks the adaptive skyline filtering technique, which can effectively reduce the number of processing nodes and the transmitted data volume, and hence cut down the communication cost of the iSky.

5.4 Processing Nodes Involvement

We now consider the number of processing nodes involved in answering skyline queries. Nodes that compute their local skylines, deliver messages or transfer intermediate results, are all processing nodes. The naive approach is excluded as it always involves all peers.

According to the results reported in Figure 10, the number of processing nodes in the SSP increases dramatically when the network size grows: it approximates 2000 when the network size is 2048. Whereas, our iSky involves no more than 100 nodes in the same situation in the independent and anti-correlated datasets. This is because the SSP exploits a partition technique to reduce the searching space. Though the number of query computation nodes are limited in a reasonable range, it still needs a great number of additional nodes to help deliver messages and transfer intermediate results. This performance superiority demonstrates that our iSky approach

is very efficient due to its BATON-oriented data assignment and adaptive skyline filtering technique.

5.5 Response Time

In this subsection, we first investigate the response time on progressive results reporting of all approaches. The experimental results are shown in Figure 11.

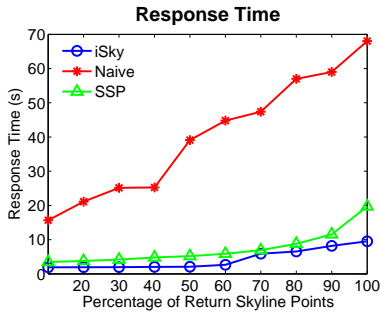


Fig. 11. Performance on Progressiveness

The iSky returns more than half of the final skyline points within only 3 second, because most local skylines at the initial skyline peers belong to the final skyline. To obtain all skylines, the iSky takes only 10 seconds. Therefore, the iSky enables a quick response at the initial stage. By contrast, the SSP only returns around 70% answers and the naive algorithm is still waiting for the first report till 10 seconds. Obviously, the above performance difference undoubtedly shows the advantage of the iSky at progressive skyline reporting, which is a very attractive feature in the real-life applications.

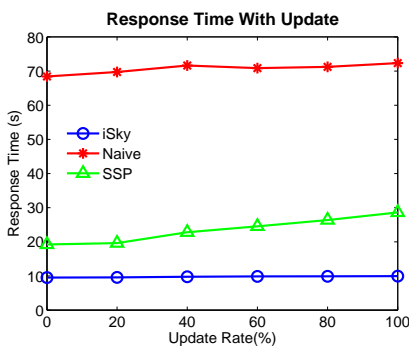


Fig. 12. Performance on Data Update

In the last experiment, we show the performance of approaches on data updates. We vary the data updates from 0% to 100%, and each update is randomly selected from new data insertions, data updates and deletions. We take the response time as the performance metric to evaluate the efficiency of different approaches. As shown in Figure 12, the performances of iSky and Naive approaches are quite stable. The iSky can maintain the index very efficient for data update, and data update has negligible effect on query efficiency. The SSP mechanism is deteriorated when the update rate increases, and performs worse than iSky because it incurs more node accesses

and higher communication cost. Additionally, the SSP adopts Z-value and space partition mechanisms for data management, which is much more expensive.

5.6 Discussion

So far, we have demonstrated the efficiency and effectiveness of our approach for skyline query in P2P networks via extensive experimental evaluation. Our proposed iSky method outperforms the competitors in most cases, and shows the advantages especially in a volatile environment, which is more general for P2P systems. Although mapping multi-dimensional data into single-dimensional space is efficient for skyline query, it introduces too much information loss and cannot preserve the spatial locality of data in the original data space. The weakness of iSky is that the index is constructed according to apriori skyline query pattern, and hence it may not effectively support skyline query variants, such as subspace skyline query and dynamic skyline query. To effectively manage the multi-dimensional data while preserving the spatial locality, the VBI-tree [17] based on BATON overlay could be a promising framework for skyline query processing in P2P networks. However, how to design a suitable filtering scheme for the skyline computation on VBI-tree is still an open problem, and is beyond the research content of our work.

6 CONCLUSION

In this paper, we have addressed the skyline computation problem in a balanced-tree-based P2P overlay network. We proposed the iSky approach including distributed data preparation and efficient skyline computation. Specifically, in the iSky approach, we first transform multi-dimensional data into 1-dimensional values with the iMinMax(θ) method, and then distribute values to BATON nodes. Subsequently, we improve data transmission efficiency and reduce both processing cost and communication cost, by using both local filters and global filters. The analysis is given to justify the integrity, correctness and progressiveness of the proposed iSky algorithm. Finally, we conduct extensive experiments on both synthetic and real datasets to compare the iSky approach with the existing alternatives. The experimental results demonstrate that our proposed iSky is efficient, robust and progressive.

REFERENCES

- [1] J. Aspnes and G. Shah. Skip graphs. In *Proc. SODA*, 2003.
- [2] W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *Proc. EDBT*, pages 256–273, 2004.
- [3] I. Bartolini, P. Ciaccia, and M. Patella. Efficient Sort-based Skyline Evaluation. In *ACM Transactions on Databases Systems*, 33(4):1–45, 2008.
- [4] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. ICDE*, pages 421–430, 2001.
- [5] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like P2P systems scalable. In *Proc. SIGCOMM*, 2003.
- [6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *Proc. ICDE*, pages 717–719, 2003.
- [7] L. Chen, B. Cui, H. Lu, L. Xu and Q. Xu. iSky: Efficient and Progressive Skyline Computing in a Structured P2P Network In *Proc. ICDCS*, 2008.
- [8] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. P-tree: A P2P index for resource discovery applications. In *Proc. WWW*, 2004.

- [9] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai and Y. Zhou. Parallel Distributed Processing of Constrained Skyline Queries by Filtering In *Proc. ICDE*, 2008.
- [10] S. Datta and H. Kargupta. Uniform data sampling from a peer-to-peer network. In *Proc. ICDCS*, page 50, 2007.
- [11] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *Proc. VLDB*, pages 229–240, 2005.
- [12] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A scalable overlay network with practical locality properties. In *Proc. USITS*, 2003.
- [13] K. Hose. Processing skyline queries in P2P systems. In *Proc. VLDB PhD Workshop*, 2005.
- [14] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi. Skyline queries against mobile lightweight devices in MANETs. In *Proc. ICDE*, page 66, 2006.
- [15] Y.-J. Joung, C.-T. Fang, and L.-W. Yang. Keyword search in DHT-based peer-to-peer networks. In *Proc. ICDCS*, pages 339–348, 2005.
- [16] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. Baton: A balanced tree structure for peer-to-peer networks. In *Proc. VLDB*, pages 661–672, 2005.
- [17] H. V. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, and A. Y. Zhou. VBI-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In *Proc. ICDE*, 2006.
- [18] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *Proc. VLDB*, pages 275–286, 2002.
- [19] D. Li, X. Lu, B. Wang, J. Su, J. Cao, K. C. C. Chan, and H. V. Leong. Delay-bounded range queries in DHT-based peer-to-peer systems. In *Proc. ICDCS*, page 64, 2006.
- [20] H. Li, Q. Tan, and W. Lee. Efficient Progressive Processing of Skyline Queries in Peer-to-Peer Systems In *Proc. INFOSCALE*, 2006.
- [21] M. Li, W. Lee, and A. Sivasubra. Semantic small world: An overlay network for peer-to-peer search In *Proc. ICNP*, 2004.
- [22] B. Liu, W.-C. Lee, and D. L. Lee. Supporting complex multi-dimensional queries in P2P systems. In *Proc. ICDCS*, pages 155–164, 2005.
- [23] B. C. Ooi, K.-L. Tan, C. Yu, and S. Bressan. Indexing the edge: a simple and yet efficient approach to high-dimensional indexing. In *Proc. PODS*, pages 166–174, 2000.
- [24] J. Pang, P. B. Gibbons, M. Kaminsky, S. Seshan, and H. Yu. Defragmenting DHT-based distributed file systems. In *Proc. ICDCS*, page 14, 2007.
- [25] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proc. SIGMOD*, pages 467–478, 2003.
- [26] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Databases Systems*, 30(1):41–82, 2005.
- [27] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM*, 2001.
- [28] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Middleware*, 2001.
- [29] I. Stoica, R. Moris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM*, 2001.
- [30] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *Proc. VLDB*, pages 301–310, 2001.
- [31] A. Vlachou, C. Doukeridis, Y. Kotidis, and M. Vazirgiannis. SKYPEER: Efficient Subspace Skyline Computation over Distributed Data In *Proc. ICDE*, 2007.
- [32] S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu. Efficient skyline query processing on peer-to-peer networks. In *Proc. ICDE*, pages 1126–1135, 2007.
- [33] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high dimensional spaces. In *Proc. of VLDB*, pages 194–205, 1998.
- [34] K. Wei, A. J. Smith, Y.-F. R. Chen, and B. Vo. Whopay: A scalable and anonymous payment system for peer-to-peer environments. In *Proc. ICDCS*, page 13, 2006.
- [35] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi. Parallelizing skyline queries for scalable distribution. In *Proc. EDBT*, pages 112–130, 2006.
- [36] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu and Q. Zhang. Efficient computation of the skyline cube In *Proc. VLDB*, 2005.
- [37] D. Zinn. Skyline Queries in P2P Systems In *Diploma Thesis, TECHNISCHE UNIVERSITAT ILMENAU*, 2005.



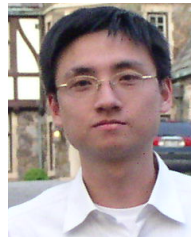
Bin Cui received the PhD degree in computer science from the National University of Singapore in 2004. Currently, he is a Professor in Department of Computer Science, Peking University. His major research interests include index techniques, multi/high databases, multimedia retrieval, and database performance. He has served on program committees of several database conferences, including SIGMOD, VLDB and ICDE. He has published more than 40 conference/journal papers in international conferences and journals. Dr. Cui is a member of ACM, and a senior member of the IEEE and the IEEE Computer Society.



Lijiang Chen is a PhD student in the Computer Networks and Distributed Systems Lab, Department of Computer Science, Peking University, China. He is supervised by Professor Bin Cui and Professor Yafei Dai. His current research interests include P2P data management, P2P search and distributed systems. He is a student member of IEEE.



Linhao Xu is a staff research engineer of the IBM China Research Lab (IBM CRL) and now he focuses on the research work on semantic data management. His research interests includes semantic data management, Web information extraction, indexing and query processing in peer-based data management system, spatial-temporal indexing and query processing on moving objects database.



Hua Lu received both his BSc and MSc degrees from Peking University, China, in 1998 and 2001, respectively. He received the PhD degree in computer science from National University of Singapore in 2007. He is currently an Assistant Professor in the department of Computer Science, Aalborg University, Denmark. His research interests include skyline queries, spatio-temporal databases, geographic information systems, and mobile computing.



Guojie Song received a PhD from Peking University, China, in 2004. Currently, he is an associate professor in Department of Computer Science, Peking University. His major research interests include data mining, machine learning, wireless sensor network, and intelligent transportation system. He has published more than 20 conference/journal papers in related conferences and journals.



Quanqing Xu received his MSc from Beijing Institute of Technology, China, in 2004, and his BSc from QingDao Technological University, China, in 2001. He began his PhD study in the Department of Computer Science at Peking University in 2005. He is supervised by Professor Yafei Dai and Professor Bin Cui. His research interests include Peer-to-Peer information retrieval, Peer-to-Peer data management.