iSky: Efficient and Progressive Skyline Computing in a Structured P2P Network *

Lijiang Chen[1] Bin Cui[1] Hua Lu[2] Linhao Xu[3] Quanqing Xu[1]
[1]Key Laboratory of High Confidence Software Technologies (Peking University)
Ministry of Education, China
EECS, Peking University, China {*clj,bin.cui,xqq*}*@pku.edu.cn*
[2]Department of Computer Science, Aalborg University, Denmark *luhua@cs.aau.dk*
[3]IBM China Research Lab, China *xulinhao@cn.ibm.com*

Abstract

An interesting problem in peer-based data management is efficient support for skyline queries within a multi-attribute space. A skyline query retrieves from a set of multi-dimensional data points a subset of interesting *points, compared to which no other points are better. Skyline queries play an important role in multi-criteria decision making and user preference applications. In this paper, we address the skyline computing problem in a structured P2P network. We exploit the iMinMax(θ) transformation to map high-dimensional data points to 1-dimensional values. All transformed data points are then distributed on a structured P2P network called BATON, where all peers are virtually organized as a balanced binary search tree. Subsequently, a progressive algorithm is proposed to compute skyline in the distributed P2P network. Further, we propose an adaptive skyline filtering technique to reduce both processing cost and communication cost during distributed skyline computing. Our performance study, with both synthetic and real datasets, shows that the proposed approach can dramatically reduce transferred data volume and gain quick response time.*

1. Introduction

In the recent years, peer-to-peer (P2P) computing has become extremely popular. It has a number of desirable features like low deployment cost, but high scalability, flexibility and computing capability. Because of these, P2P has been widely used in various applications such as information retrieval and data management [8, 15, 16]. Among these, marrying P2P to data management technologies and exploiting P2P to process various queries are of special interests.

So far, conventional queries including nearest neighbor (NN) queries and range queries [10, 11] have been successfully adapted to P2P networks. In these precedents, queries against large number of distributed sites have been facilitated by the flexible and powerful functionalities of P2P networks. The success of such queries undoubtedly encourages further attempts to adapt other kinds of queries to P2P networks. Skyline query is a good target in that direction.

Given a set of d-dimensional points, a skyline query [2] 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 no worse than p_2 in any dimension but better in at least one dimension. Skyline queries are well studied in centralized systems [2, 4, 5, 9, 12, 13]. However, little work has concentrated on efficient skyline query processing in a P2P network. The problem is that it is hard to elegantly adopt a centralized index scheme to a distributed network for computing skyline query. Though well-known DHT-based P2P networks [15, 16] 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 skyline query on DHT-based P2P systems.

Wu et al. [19] first attempted progressive processing of skyline queries on a CAN [15] based P2P network. The proposal controls query propagation based on the partial order of CAN's zones. Unfortunately, it focuses on constrained skyline queries [13, 14, 19]. Wang et al. [18] proposed Skyline Space Partitioning (SSP) approach to compute skylines on a structured P2P network called BATON [8]. The advantage of SSP is that it relaxes both network organization and skyline query propagation, compared to the proposal in [19]. However, SSP has two disadvantages. First,at the early stage of each skyline query, SSP misses a considerable number of peer nodes that have potential skyline points, which renders the result reporting less progressive. Second, while during the query processing, the filtering points used by SSP are not adaptive to the intermediate query results, which hurts the filtering capacity.

*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.

Motivated by these observations, we in this paper address efficient and progressive skyline computing on the structured P2P network BATON [8]. We use BATON as the underlying P2P platform that naturally supports one-dimensional index scheme. Thus we can adopt a centralized index scheme to a distributed P2P network, so as to efficiently perform skyline query and make the maintenance of the workload balance easier. Essentially, each d-dimensional data point is transformed into a 1-dimensional value and then stored in the 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 [17] then can be easily adapted to such a setting. Aside from the progressiveness gained from the data transformation and distribution, 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 skip peers without expected data points and remove finally unqualified intermediate answers on further involved peers.

We make the following major contributions in this paper. First, based on the *iMinMax* transformation converting a multi-dimensional dataset to a 1-dimensional one, we distribute data across BATON peers in a way facilitating distributed skyline query processing. We accordingly name our skyline query processing approach *iSky*, which is well adapted to the structured P2P environment and able to report answers progressively. Second, within the iSky approach, we propose an adaptive filtering technique. It can not only prune unpromising peer nodes without involving them in skyline processing, but also identify unqualified local skyline points on involved peer nodes and prevent them from being transferred through the network. Third, we conduct extensive experiments on both real and synthetic datasets to evaluate the performance of our proposal iSky. Our results show that iSky is efficient, robust and progressive.

The rest of this paper is organized as follow. Section 2 reviews the related work and introduces the technical background of BATON. Section 3 presents the data preparation of our iSky approach. Section 4 details the algorithms of iSky approach. Section 5 reports the performance study results, and Section 6 concludes the paper.

2 Preliminaries

2.1 Related Work

Most conventional skyline queries are focused on traditional centralized storage, and their algorithms can be divided into two categories. One category contains those that do not require any indexes on the dataset. Borzonyi et al. [2] introduced the skyline query into database systems with algorithms *Block Nested Loop* (BNL) and *Divide-and-Conquer* (D&C). Chomicki et al. [4] proposed a *Sort-Filter-Skyline* (SFS) algorithm as a variant of BNL. Godfrey et al. [5] provided a comprehensive analysis of those aforementioned algorithms without indexing supports, and proposed a new hybrid method *linear elimination sort for skyline* (LESS). The other category contains those requiring specific indexes. Tan et al. [17] proposed two progressive algorithms: *Bitmap* and *Index*. The former represents points in bit vectors and employs bit-wise operations, while the latter utilizes iMinMax [12] data transformation and B^+-tree indexing. Kossmann et al. [9] 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. [13] proposed a *Branch-and-Bound Skyline* (BBS) method which is IO optimal on R^*-tree indexed datasets.

Apart from centralized contexts, Balke et al. [1] addressed skyline operation in a Web setting where different dimensions are stored on different web sites. Our work differs from that work in several ways. We set skyline computation in a structured P2P system, which is much more complex than the Web system. We deliberately transform the original dataset and distribute different portions to corresponding peer nodes according to a specific P2P protocol BATON [8], because a simple horizontal or vertical dataset partition across peer nodes does not support progressive skyline computation.

Recently, skyline query in P2P systems has gained research efforts. Kiaja [6] studied how to compute approximate skylines on unstructured P2P networks. Wu et al. [19] 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. Wang et al. [18] proposed Skyline Space Partitioning (SSP) approach to compute skylines on BATON. Though [18] is the closest related work to ours in this paper, our iSky has several distinctive features. First, iSky is based on data transformation rather than space partitioning. Second, iSky exploits the BATON protocols better and hence distributes data portions across peers more deliberately. Third, iSky can progressively return skyline results and efficiently prunes unpromising peer nodes on the fly. The results of our extensive comparative experiments show that iSky outperforms SSP in most cases.

2.2 Background of BATON

Intended to facilitate data indexing and data distribution in structured P2P systems, BATON (BAlanced Tree Overlay Network) [8] is an overlay which virtually constitutes a balanced binary search tree. Each peer P in BATON maintains a tree node with a continuous key range, and connects to other peer nodes via four kinds of network links: a parent

link to the parent node, children links to child nodes, adjacent links to adjacent nodes in the in-order traversal, and links to the routing nodes which are at the same level as P and have equal distance of a power of 2 from P. The key range on P is required to be greater than that on P's left adjacent node but smaller than that on P's right adjacent node. Therefore, a traversal in BATON following adjacent links will return all data in the ascending order.

3 Data Preparation of iSky Approach

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

3.1 BATON-Oriented Data Transformation and Assignment

Without loss of generality, we make three assumptions as follows. First, we assume all values are numeric. Second, the range of each dimension is assumed to be normalized into [0, 1). Third, we assume that on each dimension larger values are preferred by users in their skyline queries. In practice, we can apply our proposal to cases where smaller values are preferred by adding a negative sign to each values on relevant dimensions. 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 ith 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 Ω.

Proposed for skyline computation in a centralized environment, the Index approach [17] works as follows. Basically, any d-dimensional point x in Ω is transformed to a 1-dimensional value y according to a special case of iMinMax [12], and then all transformed 1-dimensional 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 in this paper employ the same data transformation mechanism. Specifically, for any d-dimensional point $x = (x_1, x_2, \ldots, x_d)$, let x_{max} be the largest value among all dimensions and let the corresponding dimension of x_{max} be $d_{x_{max}}$[1]. By the equation $y = d_{x_{max}} + x_{max}$, the data point x is transformed to y over a single dimensional range. We should notice that the above transformation actually separates the data space into d partitions $[1, 2), [2, 3), \ldots, [d - 1, d), [d, d + 1)$ and offers an order within each partition.

Instead of using a B$^+$-tree to index all 1-dimensional values as the Index approach [17], we take advantage of the balanced binary tree of BATON. Specially, all 1-dimensional values (and their corresponding original d-dimensional points) are assigned to all peers according to BATON protocols such that: (1) all 1-dimensional values on each node P_i (no matter internal or leaf node) form a subrange R_i; (2) all such subranges are disjoint with each other; (3) the union of all subranges is exactly [1, d+1); (4) the in-order traversal of the BATON tree is exactly a sequential scan of all subranges in ascending order.

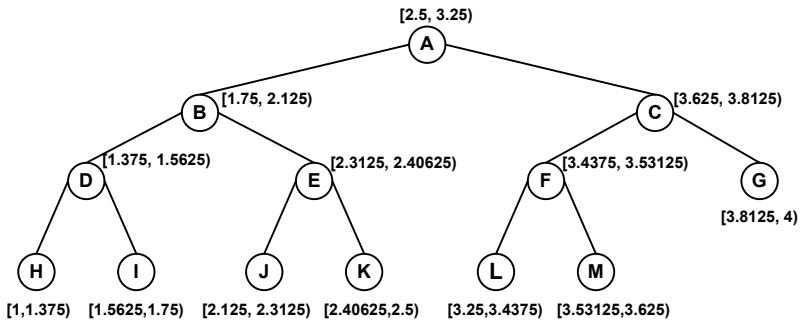
Example 1 *Consider the example shown in Figure 1. The left part of the figure is a BATON network with 13 peers. The right part shows a dataset with 30 3-dimensional data points, which are partitioned into 3 parts according to the transformation aforementioned. Each partition corresponds to a specific dimension, and is sorted in 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* $[1, 4)$*, and 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.* □

3.2 Important Properties

From the example above, we make two important observations. First, some skyline candidate points can be largely 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 largely at the top of each partition. In Figure 1, 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 di-

[1] If x_i and x_j have the same value and $x_i = x_j = x_{max}$, then we define $d_{x_{max}} = min(i, j)$, which is same as $d_{x_{min}}$.



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)

Figure 1. An example of data transformation and assignment

mensions 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 1, the skyline point dp_{21} dominates all data points whose maximum value is 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 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*

$$\begin{aligned} \mathcal{P} &= \{P_i \mid \exists x \in \mathcal{D}_{P_i} \text{ such that } x_{max} = m\} \\ \mathcal{M} &= \{x \mid x \in \mathcal{D}_{\mathcal{P}} \wedge x_{max} = m\}, \\ \text{where } \mathcal{D}_{\mathcal{P}} &= \cup_{P_i\in\mathcal{P}}\mathcal{D}_{P_i}. \end{aligned}$$

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 contained in that of the whole dataset of the P2P system, i.e., the 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 $S_{\mathcal{M}} \neq \varnothing$.*

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 $S_{\mathcal{M}} \neq \varnothing$. □

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 $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_d$ by the $iMinMax(\theta)$ transformation, and the 1-dimensional range at each peer 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 $(min_i < k+1 \leq max_i) \wedge ([k, k+1) \cap [min_i, max_i) \neq \varnothing)$ and $\mathcal{D}_{\mathcal{R}} = \bigcup_{P_i\in\mathcal{R}} \mathcal{D}_{P_i}$, where $1 \leq k, i \leq d$. Then, $\mathcal{M} \subseteq \mathcal{D}_{\mathcal{R}}$.*

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. On the other side, according to the definition of $\mathcal{R}$, for any $P_i \in \mathcal{R}$, if the condition $(min_i < k+1 \leq max_i) \wedge ([k, k+1) \cap [min_i, max_i) \neq \varnothing)$ 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, $\mathcal{M} \subseteq \mathcal{D}_{\mathcal{R}}$. □

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. □

4 Algorithms of iSky Approach

Based on the data preparation as described in Section 3, we in this section detail the algorithms of our iSky approach. We call a peer issuing a skyline query *q query originator*, denoted as P_{org}. Any other peer processes query q is called a *processing peer*. The objective of our algorithms is to minimize the number of processing peers, and make local skyline computations fast on those processing peers.

4.1 Locating Initial Skyline Peers

According to Theorem 1, the *initial skyline points (ISP)*, denoted as $\mathcal{S}_{\mathcal{M}}$, 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 $\mathcal{P}$, 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 $(min_i < k+1 \leq max_i) \wedge ([k, k+1) \cap [min_i, max_i) \neq \varnothing)$ is satisfied, where $1 \leq k \leq d$ (lines 4–6). If so, P_i reports its identifier to the root of the BATON (lines 7–8).

Algorithm 1: `locateISP(root)`

```
1 compute max_{x∈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 ≤ d; k++) do
5     if (min_i < k + 1 ≤ max_i) &&
6       ([k, k + 1) ∩ [min_i, max_i) ≠ ∅) then
7         report id to root;
8         break;
```

From Theorem 2, we discover that the number of the initial skyline peers equals to the dimension 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 1. This number is quite small compared to 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 [3], and therefore use the root node to maintain the initial skyline peers is applicable.

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

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

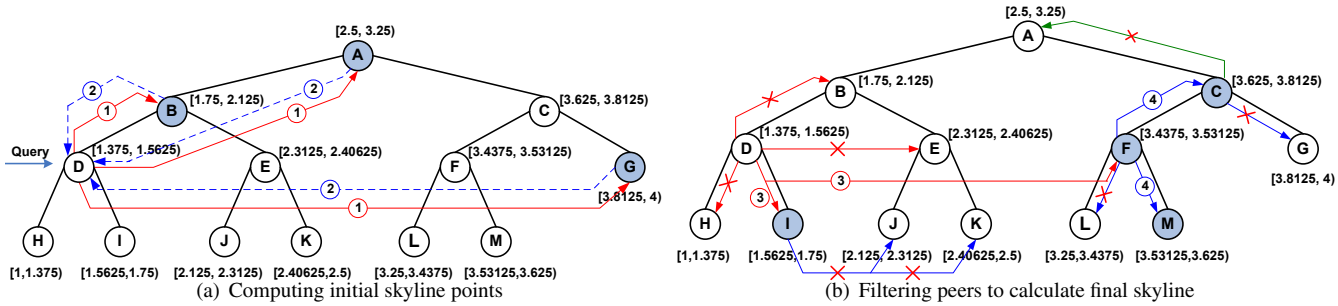
where $\mathcal{S}_{\mathcal{M}}$ is the initial skyline points. Subsequently, SF_{global} is sent out with the skyline query from the query originator P_{org} to prune unpromising peers.

However, if SF_{global} can not prune a peer, we need to do a local skyline computation. For such a case, the filtering capability of SF_{global} is unlikely to be strong. Therefore, we need another skyline filter to remove local skyline points that are dominated by some final skyline points. We use *dominating region* (DR) [7] 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.

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

```
1  result = ∅; new_prune=0; DR_max = 0; idx = 0;
2  if (SF_global ≥ max_{x∈D_{P_i}}(x_max)) then
3      forward q to all the unchecked peer in routing table;
4  else
5      foreach x ∈ D_{P_i} do
6          if (dom(SF_local, 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
                           result = result − r;
13                         else
14                             result = result ∪ x;
15                             if DR_x > DR_max then
16                                 DR_max = DR_x;
17                                 idx = x;
18                             if x_min > SF_global then
                               new_prune = x_min;
19     if DR_max > SF_local then SF_local = idx;
20     send ⟨q, SF_local, new_prune⟩ to all the unchecked
       nodes in routing table;
21     return ⟨result⟩;
```

Given a skyline query q and a processing peer P_i, P_i processes q using *peerSkyline()* 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 7–12). If x is not dominated, it is added into $result$ and two filters SF_{global} and SF_{local} are updated if necessary (lines 13–19). 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 20–21).



(a) Computing initial skyline points

(b) Filtering peers to calculate final skyline

Figure 2. An example of the iSky approach

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 sends its query q to all initial skyline peers in $\mathcal{P}$ (lines 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 q is broadcasted to all the nodes in the routing table of P_{org}(lines 6–12). For each peer who receives the query, it calls *peerSkyline()* as 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. Finally, the query originator merges all the skyline candidates from the P2P system and returns the answer to the user (line 14).

Algorithm 3: `iSky()`

```
1  result = ∅; DR_max = 0; idx = 0;
2  foreach (P ∈ 𝒫) do send q to P;
3  receive the results from all initial skyline peers;
4  result = merge(received skyline points);
5  // generate skyline filters;
6  foreach (r ∈ result) do
7      if DR_r > DR_max then
8          DR_max = DR_r;
9          idx = r;
10 SF_local = idx;
11 SF_global = max_{r∈result}(r_min);
12 foreach P in P_org's routing table do
13     call peerSkyline(q, SF_global, SF_local);
14 result = merge(received skyline points);
```

Example 2 *Figure 2 illustrates how the iSky approach works. Suppose node D issues the skyline query q. In the first step, D sends query q to the initial skyline peers A, B and G. Then nodes A, B and G return their local skyline results $\{dp_1, dp_2, dp_{11}, dp_{21}\}$ to D and D computes the initial skyline points $\{dp_1, dp_2, dp_{21}\}$ on the returned results, as depicted in Figure 2(a). In the second step, D broadcasts q and two filters $SF_{global} = 0.6$ and $SF_{local} = \{dp_1, dp_{21}\}$ only to node I and F, without considering nodes B, H and E. This is because node B has already been visited and the max_i of nodes H and E are smaller than SF_{global}. Finally, node F sends q to nodes 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 nodes are visited and prune all unpromising nodes on the fly. For example, unpromising nodes J and K can be pruned by node I since they are in I's routing table.* □

5 Performance Study

We study the performance of our *iSky* approach on BATON with respect to three aspects: dimensionality, network size and cardinality. We compare our iSky approach with the SSP method [18] and the adapted naive approach based on the centralized Index skyline algorithm [17]. In the naive approach, we distribute the 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 SSP yields significantly better performance than the distributed skyline query algorithm DSL [19], we omit the comparison with DSL for the clarity of presentation.

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

Table 2. Parameters Used in Experiments

We consider the following performance metrics: data reduction rate, communication cost, number of processing nodes and response time. They are measured through simulation experiments on a Linux Server with four 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. The average results are reported in this section. We use three kinds of different datasets: a real dataset of NBA players' season statistics from 1949 to 2003[2], which has 16,644 records and approximates a correlated data distribution, and two synthetic datasets of independent and anti-correlated

[2]http://databasebasketball.com

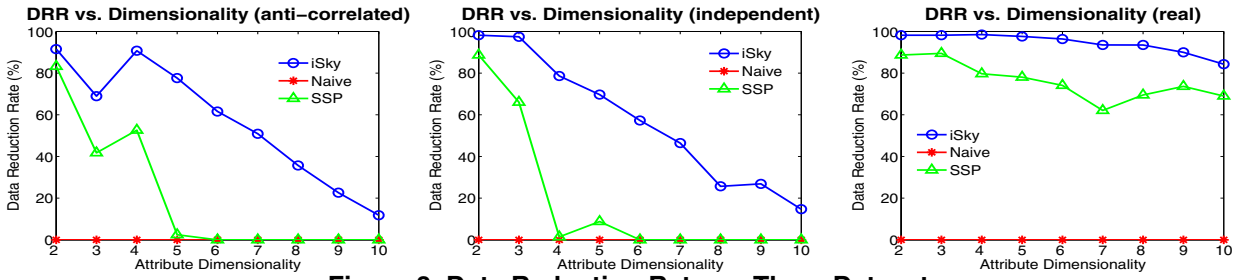


Figure 3. Data Reduction Rate on Three Datasets

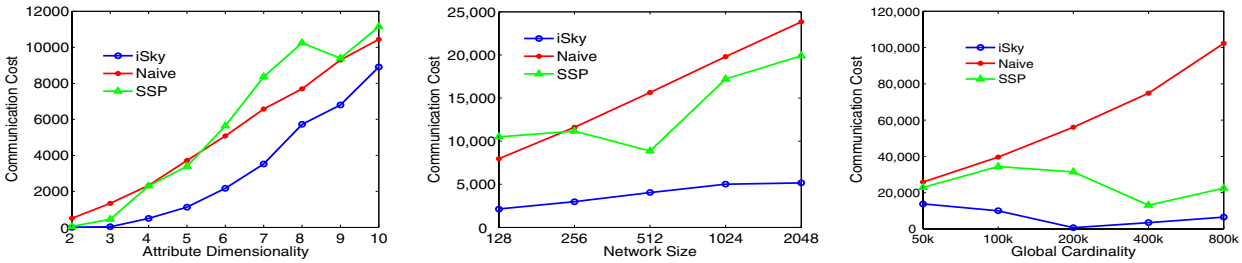


Figure 4. Performance on Communication Cost

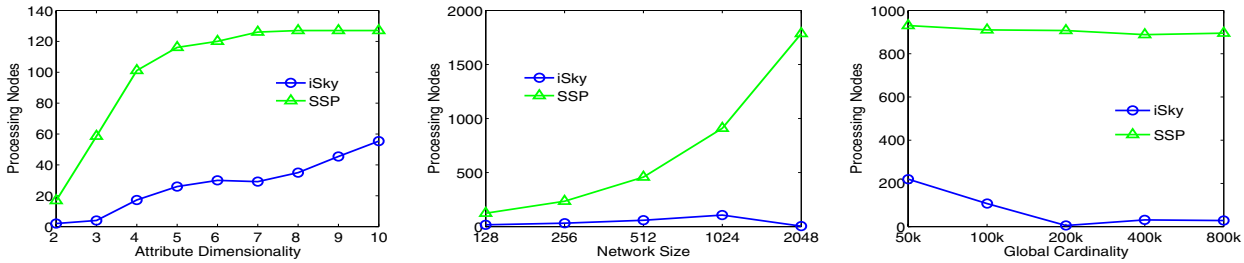


Figure 5. Performance on Processing Nodes Involved

distribution respectively. The parameters used in the experiments are listed in Table 2. Unless stated otherwise, 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 [7], the proportion of data points reduced by the filter points to the number of points in the unreduced skyline. 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 but those initial skyline peers.

Figure 3 shows how DRR varies with respect to dimensionality on all three datasets. We compare our iSky with SSP and the naive approach. Note that, since the data size of real NBA dataset is limited, we set the default network size as 128, and vary cardinality from 8-128. We can see that iSky is better than the other approaches, because of its adaptive filter technique. It prevents the unqualified local skyline from being transmitted through the network, and because the filter is adaptive its filtering effect becomes stronger and stronger during the processing. Another clear phenomenon is that the SSP approach has a marked deficiency when the dimensionality is high (5 in the anti-correlated, 4 in the independent and 6 in the real dataset). This is because SSP uses no adaptive filtering technique as iSky does.

We also investigate into how DRR is affected by network size and global cardinality on all datasets. In those cases, our iSky still outperforms both SSP and naive approaches. Due to the space constraint, we omit those results. For the same reason, we in the sequel only report results obtained on the independent dataset, while iSky is still the best in almost all cases on other datasets.

5.2 Communication Cost

The communication cost is measured by the number of messages transferred for skyline computation. From Figure 4, we can see that the communication cost of iSky is stable with respect to the increases of dimensionality, network size and cardinality. As previous cases, iSky outperforms SSP and the naive approach. The naive approach deteriorates notably when the scale increases, as it does not employ any specific optimization strategies in query processing. As the network size increases, SSP incurs considerably higher

communication cost compared to iSky, because SSP lacks effective and adaptive skyline filters, which reduce the number of processing nodes and that of transmitted data points and cut the communication cost of iSky.

5.3 Processing Nodes Involvement

We now consider the number of processing nodes involved in answering queries. The naive approach is excluded as it involves all peers in any query. According to the results reported in Figure 5, the number of processing nodes in 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 most other cases, iSky roughly involves only as 20% processing nodes as does SSP. This performance superiority demonstrates that our iSky approach is very efficient due to its BATON-oriented data assignment and adaptive skyline filtering technique.

5.4 Report Progressiveness

We finally investigate the report progressiveness of all approaches. The experimental results are shown in Figure 6. iSky reports more than half of the final skyline points within

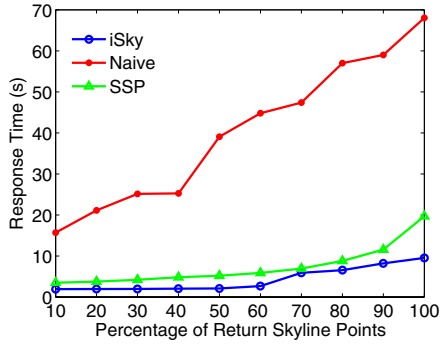


Figure 6. Performance on Progressiveness

only 3 second, because most local skylines of the initial skyline peers in our method are contained in the final skyline. iSky returns all answers within only 10 seconds. By contrast, SSP only returns around 70% answers and the naive algorithm is still waiting for its first report at that moment. This performance difference undoubtedly demonstrates the good progressiveness of iSky. iSky permits the immediate generation of a few initial answers while additional ones are still being searched for. In other words, iSky supports progressive query result reporting, an important feature desired in interactive query and data analysis applications.

6 Conclusion

In this paper, we studied skyline query processing on a balanced-tree-based P2P overlay called BATON. We propose an approach called iSky, which consists of specific data preparation and detailed query processing algorithms. All multi-dimensional data points are first mapped into a 1-dimensional space using the iMinMax transformation, and then are carefully assigned to peers in BATON. We further develop an adaptive filtering technique to enhance data transmission efficiency and cut both processing cost and communication cost. We conduct extensive experiments on both synthetic and real datasets to compare our iSky approach with existing alternatives. The experimental results demonstrate that iSky is efficient, robust and progressive.

References

[1] W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *Proc. EDBT*, pages 256–273, 2004.

[2] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. ICDE*, pages 421–430, 2001.

[3] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like P2P systems scalable. In *Proc. SIGCOMM*, 2003.

[4] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *Proc. ICDE*, pages 717–719, 2003.

[5] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *Proc. VLDB*, pages 229–240, 2005.

[6] K. Hose. Processing skyline queries in P2P systems. In *Proc. VLDB PhD Workshop*, 2005.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] B. Liu, W.-C. Lee, and D. L. Lee. Supporting complex multi-dimensional queries in P2P systems. In *Proc. ICDCS*, pages 155–164, 2005.

[12] 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.

[13] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proc. SIGMOD*, pages 467–478, 2003.

[14] 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.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM*, 2001.

[16] 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.

[17] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *Proc. VLDB*, pages 301–310, 2001.

[18] 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.

[19] 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.