Skyline Processing on Distributed Vertical Decompositions

George Trimponias, Ilaria Bartolini, Dimitris Papadias, Yin Yang

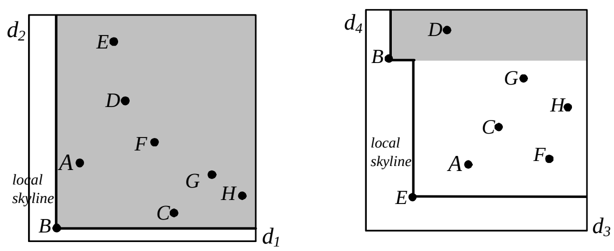
Abstract—We assume a dataset that is vertically decomposed among several servers, and a client that wishes to compute the skyline by obtaining the minimum number of points. Existing solutions for this problem are restricted to the case where each server maintains exactly one dimension. This paper proposes a general solution for vertical decompositions of arbitrary dimensionality. We first investigate some interesting problem characteristics regarding the pruning power of points. Then, we introduce VPS (vertical partition skyline), an algorithmic framework that includes two steps. Phase 1 searches for an anchor point P_{anc} that dominates, and hence eliminates, a large number of records. Starting with P_{anc}, Phase 2 constructs incrementally a pruning area using an interesting union-intersection property of dominance regions. Servers do not transmit points that fall within the pruning area in their local subspace. Our experiments confirm the effectiveness of the proposed methods under various settings.

Keywords—Distributed Skyline, Vertical Partitioning, Query Processing.

1 INTRODUCTION

Given a data set DS of d-dimensional records/points, a record $P \in DS$ *dominates* another $Q \in DS$, if P is no worse than Q on all d attributes/dimensions, and it is better than Q on at least one dimension. The *skyline* $SKY \subseteq DS$ consists of all points that are not dominated. In this paper, we assume that the dataset is vertically distributed among m servers, so that a server N_i stores a subset D_i of the dimensions and the ID of each record. For every two servers N_i and N_j ($1 \le i \ne j \le m$), $D_i \cap D_j = \emptyset$, i.e., the servers do not have common attributes except for the record ID. As a real-world example, consider that a mobile client wishes to compute the skyline over a restaurant dataset based on the following criteria: quality, value, proximity to cinemas, and distance from the current location. The former two attributes are provided by a restaurant rating service, whereas the rest are obtained from an on-line map server. Similarly in e-commerce applications, product prices may be provided by sites that find the lowest price (e.g., *pricegrabber.com*), while technical characteristics reside in specialized libraries (e.g., *cnet*).

Fig. 1 shows an instance with two servers N_1, N_2, and 8 points A-H. N_1 (resp. N_2) maintains the subspace $D_1 = \{d_1, d_2\}$ (resp. $D_2 = \{d_3, d_4\}$). Without loss of generality, throughout our presentation we consider that smaller values are preferred on all dimensions. The *local skyline* SKY_1 at N_1 contains a single point B, which dominates all other records in D_1 (Fig. 1a). Similarly, the *local skyline* SKY_2 at N_2 consists of B and E (Fig. 1b). The *global skyline* SKY over all dimensions comprises all points that appear in SKY_1 or SKY_2 (i.e., B, E), as well as additional records that are not dominated by a single point on all dimensions, i.e., $SKY = \{B, E, A, C\}$. For instance, $A \in SKY$ since it is dominated by different records (e.g., B and E) in the two subspaces. On the other hand, F, G, H and D are not in SKY because they are dominated by a single point (A, C, C, B, respectively) on all dimensions.



(a) Subspace D_1 at server N_1. (b) Subspace D_2 at Server N_2.

Fig. 1. Running example.

In our setting, we assume that there is no central server to materialize SKY. Moreover, the skyline may change when updates occur to one or more servers (e.g., some restaurant ratings are altered), and it may depend on the particular user (e.g., the distance between the restaurant and the client's location). Hence, SKY must be computed on-demand. The skyline algorithm should minimize the points retrieved from each server because the communication overhead constitutes the dominant factor in battery consumption for mobile clients [7][17]. Moreover, more data increase the amount of computations required to process them.

- *G. Trimponias and D. Papadias are with the Dept. of Computer Science and Engineering, Hong Kong University of Science and Technology. E-mail: {trimponias, dimitris}@cse.ust.hk*
- *I. Bartolini is with the Department of Electronics, Computer Science and Systems, University of Bologna. E-mail: i.bartolini@unibo.it*
- *Y. Yang is with the Advanced Digital Sciences Center, Singapore. E-mail: yin.yang@adsc.com.sg*

Manuscript received: June 2011, revised October 2011.

A naïve method would transmit all record coordinates (from every server), based on which the client could derive *SKY* using any centralized skyline algorithm (e.g., [4][5][9]). Clearly, this is very inefficient in terms of both communication and computation overhead. To alleviate this problem, we can take advantage of the points received so far to eliminate records that are guaranteed to be dominated globally. In our example, assume that the client has received point *B*; then, the transmission of *D* by N_1 and N_2 can be avoided, since *D* is dominated by *B* in both subspaces D_1 and D_2. However, *F*, *G* and *H* must still be sent to the client (although they are not in *SKY*) because they are dominated by *B* in only one of the subspaces (D_1).

A natural question is which of the received records to utilize (and how), in order to eliminate the maximum number of false hits (i.e., points such as *F*, *G*, *H* that are not in the global skyline). We propose VPS (short for vertical partition skyline), a general methodology that exploits some interesting skyline properties to maximize pruning in two steps. Phase 1 searches for an anchor point P_{anc}, which has the potential to eliminate a large number of records. Phase 2 uses P_{anc}, possibly in combination with other points encountered during Phase 1, to generate a pruning area for each server; records that fall within this area are excluded from skyline processing. The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 investigates the problem characteristics. Section 4 describes and analyzes the proposed algorithmic framework. Section 5 experimentally confirms the effectiveness of VPS, and Section 6 concludes the paper.

2 Related Work

The skyline operator [4] has received considerable attention in the literature of centralized databases [3][5][10][11] and horizontal decompositions, where each server stores a subset of the records [6][13][14][15]. On the other hand, the only work on distributed skyline processing for vertically partitioned data is [2], which aims at minimizing the communication cost considering that the client retrieves *m* attribute values for a set of records *DS* from *m* servers. Specifically, each server N_i ($1 \le i \le m$) (i) maintains the ID and exactly one dimension d_i of every record in *DS*, (ii) sorts all objects in ascending order of d_i at a preprocessing step, and (iii) allows both *sorted access* (i.e., get the next record with the lowest d_i), or *random access* (i.e., given a record ID, obtain d_i). Balke et al. [2] propose two solutions called *basic distributed skyline* (BDS) and *improved distributed skyline* (IDS).

In BDS the client first retrieves attribute values from the servers in a round-robin manner, using sorted accesses, until it reaches an *anchor point* P_{anc} at all servers. Records not encountered in any of the servers are worse than P_{anc} on every dimension, and therefore dominated by P_{anc}. Thus, the skyline is computed using only the points discovered before P_{anc}. Fig. 2a presents BDS on the dataset of Fig. 1, assuming that the four attributes are distributed over four servers and sorted in ascending order. Anchor point *A* is discovered at the 5th round-robin iteration at server N_2 [1]. At this time, the client stops the sorted accesses, obtains (using random accesses) the remaining dimensions of all records encountered before *A* in some server, and computes *SKY* = {*B*, *E*, *A*, *C*}. In Fig. 2a, the gray cells (resp. circles) denote sorted (resp. random) accesses. Note that there are no circles for point *F*, because it is found after *A* in both N_2 and N_3, and therefore it is dominated by *A*, independent of its unseen dimensions. Thus, the transmission of the coordinates of *F* is avoided for servers N_2 and N_3.

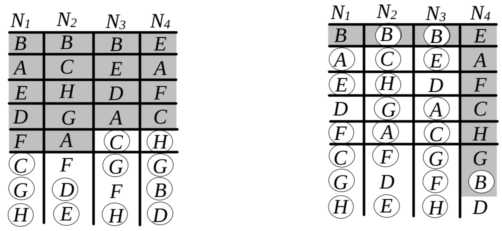


Fig. 2. Examples of BDS and IDS.

As opposed to BDS, which performs round-robin sorted accesses, IDS guides the search towards more promising servers (i.e., where an anchor point is likely to be found early) by interleaving sorted and random accesses. Specifically, whenever an object *P* is first visited using sorted access at a server N_i, the client asks each remaining server N_j ($i \neq j$) for the attribute value $P.d_j$ through a random access. Then, the client estimates the number of additional sorted accesses needed for all servers to reach *P*, from their respective current positions. Note that this number is continuously updated by taking into account the current positions in all lists, in order to properly reflect the remaining sorted accesses. If the anchor point has not been set, or *P* needs fewer sorted accesses than the current P_{anc}, then *P* becomes the new anchor. Servers that have not already found P_{anc} perform sorted accesses in a round-robin fashion. As soon as all servers reach P_{anc} through sorted accesses, the client computes the skyline.

In the example of Fig. 2b, at the first iteration, server N_1 retrieves point *B* with a sorted access. Then, N_2-N_4 perform random accesses to obtain the attribute values of *B*, as well as calculate the number of sorted accesses to reach *B*, which are 1 (for N_2), 1 (for N_3) and 7 (for N_4). The

[1] In fact, BDS would continue to the next point *F*, if it has the same coordinate as *A* on d_2; for simplicity of exposition, we assume that all coordinates are different.

total number of sorted accesses needed for *B* is 9. Since the anchor has not been set, *B* becomes the current anchor. The next two servers N_2 and N_3 also encounter *B* with their respective sorted accesses. As *B* has already been encountered, no further operation is required. Then, N_4 retrieves *E*; N_1-N_3 send to the client the attribute values of *E*, as well as the number of sorted accesses to reach it. Since *E* requires a total of 10 sorted accesses, *B* (with 6 additional accesses) remains the anchor. Given that N_1-N_3 have reached *B*, only N_4 continues to perform sorted accesses, retrieving *A*, *F*, *C*, *H*, *G*, in this order, all of which necessitate more additional sorted accesses than *B*. Finally, when N_4 reaches *B*, the client computes the skyline using the 7 points *A-C*, *E-H* it has encountered. Note that the transmission of the coordinates of *D* is avoided for all servers.

Similar to BDS and IDS, we exploit a sorted order of points in each server and an anchor point P_{anc} to prune the search space. However, whereas in 1D decompositions there is a single choice of ordering per server (i.e., on the corresponding coordinate), for arbitrary dimensionality there are numerous possible orders, with variable pruning power. Multidimensional sorting functions have been explored in the literature of sort-based algorithms for centralized skyline computation [3][5][9]. In that setting, a single server stores *DS* ordered on some monotone function f_S. The goal is to compute the skyline by scanning the list and terminating as early as possible. While scanning the points in sorted order, the server maintains a *stop point* P_{stop} that satisfies an optimization criterion. Search terminates after discovering a point *Q* such that every point after *Q* is guaranteed to be dominated by P_{stop}.

Bartolini et al. [3] prove that among all symmetric[2] sorting functions f_S, the one leading to the earliest termination is *minC*, which orders points in increasing order of their minimum coordinate. Moreover, P_{stop} is the point with the minimum, maximum coordinate. As an example, assume that *DS* contains 2D points (1,4), (5,1), (2,3), (4,2), (4,4) ..., sorted on f_S = *minC*. When (2,3) is encountered it becomes P_{stop}. The minimum coordinate (4) of point *Q* = (4,4) exceeds the maximum coordinate of P_{stop} = (2,3) and the search terminates because all subsequent points are dominated by P_{stop}. The skyline is computed using only the points up to *Q* = (4,4). As opposed to [3], which aims at minimizing the computational cost for a single server, in our setting we wish to minimize the transmission overhead for multiple servers. Furthermore, for vertical decompositions, there are multiple sorted lists (one per server), and P_{anc} serves a different purpose compared to P_{stop}. Consequently, the optimality results of [3] are not applicable in our setting.

In addition, our work is related to subspace skyline computation [8][12], which assumes that a single server stores the entire dataset, and returns the skyline in a given subspace of the domain, while minimizing the I/O and CPU costs. The main challenge in subspace skyline processing is that the dataset often has rather high dimensionality; consequently, algorithms using a single index over all dimensions are inefficient. *Subsky* [12] tackles this problem by first selecting a set of cluster centers, and then mapping each record in the dataset into a single value, which is its L_∞-distance to its corresponding center. The server then indexes these values with a B-tree, and processes a subspace skyline query by scanning the leaves of the tree, until reaching certain termination criteria. This methodology is clearly inapplicable to our problem, since it requires a central server to compute the 1D mappings, index the results, and answer all incoming queries.

The method of [8], called STA, partitions the data into low-dimensional subspaces, and indexes each such subspace with an individual R-tree. A subspace skyline is computed using the trees that cover the query subspace. To minimize node accesses, STA introduces pruning strategies, using a single point or multiple points. The single point strategy resembles IDS, and selects as anchor the nearest neighbor p_{NN} of the "lower-left" corner with respect to L_1-distance (i.e., the point that minimizes the sum of coordinates). Nodes dominated by p_{NN} are eliminated. Regarding multi-point pruning, STA assigns to each subspace D_i a set of pruning points DIS_i, so that index nodes that are dominated by any point in DIS_i are pruned. Let *m* be the total number of subspaces. The pruning sets satisfy the *common point condition*, which states that for any combination of *m* pruning points $P_1 \in DIS_1$, $P_2 \in DIS_2$, … $P_m \in DIS_m$, one of these points P_k must dominate all remaining *m*–1 points, in all subspaces except for its own D_k. To compute the pruning sets, STA examines the *m* subspaces in a round-robin fashion; each round retrieves a new point with minimum subspace L_1-distance to the query lower-left corner, and attempts to add it to the corresponding pruning set. Multi-point pruning in STA does not always eliminate more nodes than a single point [8]. Moreover, the CPU cost of verifying the common point condition increases exponentially with the number of subspaces, due to exhaustive verifications of all point combinations. In Section 4.3, we compare in detail the proposed techniques with previous work.

3 PROBLEM CHARACTERISTICS

We consider distributed vertical decompositions of arbitrary dimensionality. A client wishes to compute the global skyline *SKY* by retrieving the minimum number of points from the servers. Since all points in *SKY* must be transmitted to the client anyway, our goal is to minimize the transmission of *false hits*, i.e., points received by the

[2] A function *f* is symmetric if it is invariant under any rearrangement of its variables. This implies that *f* does not privilege any attribute, which is natural for skyline computation.

client that do not belong to *SKY*. These records incur unnecessary communication cost and burden the skyline computation overhead. Section 3.1 defines the decomposition lattice, which represents all possible partitions. Section 3.2 describes pruning with a single point. Section 3.3 utilizes multiple points to further reduce the search space. Table 1 illustrates common symbols used in the rest of the paper. For ease of presentation, we consider that smaller values are preferable on every dimension, but the proposed methods can be used for every combination of minimization and maximization of attribute values on different dimensions.

TABLE1
FREQUENT SYMBOLS

Symbol	Meaning
m	Number of servers
DS, $\lvert DS\rvert$	Dataset and its cardinality
D	Global space
SKY	Set of global skyline points
N_i	The i-th server
D_i, $\lvert D_i\rvert$	Subspace at server N_i and its dimensionality
PP_J	Set of pruned points for decomposition J
VS	Set of visited points transmitted at phase 1
PS	Set of pruning points
IS	Set of incomparable points transmitted at phase 2
$P.D_i$	Projection of point P in subspace D_i
$PA.D_i$	Projection of pruning area PA in subspace D_i

3.1 Decomposition Lattice

We first investigate the possible vertical decompositions and their relationships. Note that in our problem, the decomposition is already given; thus, the discussion regards the link between different settings. Specifically, we use the term $J_{single} = (\{d_1, ..., d_{|D|}\})$ to denote the special case, where a single server maintains all attributes (i.e., the setting of [3]). On the other side of the spectrum, the special decomposition $J_{full} = (\{d_1\}, ..., \{d_{|D|}\})$ corresponds to full partitioning, where each server maintains exactly one attribute (i.e., the setting of [2]). In general, we write $J = (D_1, \ldots, D_m)$ to denote the decomposition J of the global space D into m servers, where each server N_i ($1 \le i \le m$) maintains the ID and a set D_i of attributes for every record in DS. A decomposition $J' = (D_{1'}, \ldots, D_{m'})$ of D into m' servers ($m<m'$), derived by further partitioning some local subspace(s) of J into two or more subspaces is called a *refinement* of J. If we refine only one of the subspaces into exactly two, then we call it *one-step refinement*. For instance, in the 4-dimensional space of our example, the decomposition $J_2 = (\{d_1\}, \{d_2,d_3\}, \{d_4\})$ is an one-step refinement of $J_1 = (\{d_1\}, \{d_2,d_3,d_4\})$, generated by partitioning subspace $(\{d_2, d_3, d_4\})$ into $(\{d_2, d_3\}, \{d_4\})$.

Any decomposition of m subspaces can be described as $m - 1$ successive one-step refinements of J_{single}, also called a *decomposition chain*. In our running example, J_2 can be generated by first refining J_{single} to J_1, and then J_1 to J_2 by decomposing the second subspace of J_1. Usually, the decomposition chain is not unique; e.g., we can refine J_{single} to $J'_1 = (\{d_1,d_2,d_3\}, \{d_4\})$, and, subsequently, J'_1 to J_2. Moreover, by further refining $J_2 = (\{d_1\}, \{d_2,d_3\}, \{d_4\})$ we derive the full decomposition $J_{full} = (\{d_1\}, \{d_2\}, \{d_3\}, \{d_4\})$. The set of all possible decompositions forms a lattice, where the top element is J_{single} and the bottom element is J_{full}. Any two elements are connected in the lattice, if and only if there is a decomposition chain starting from one of them and ending at the other. Furthermore, the i-th level of the lattice contains all possible decompositions into exactly i servers. Fig. 3 depicts part of the Hasse diagram of the decomposition lattice when $D = \{d_1, d_2, d_3, d_4\}$. For simplicity, we only include the decompositions mentioned in the above examples.

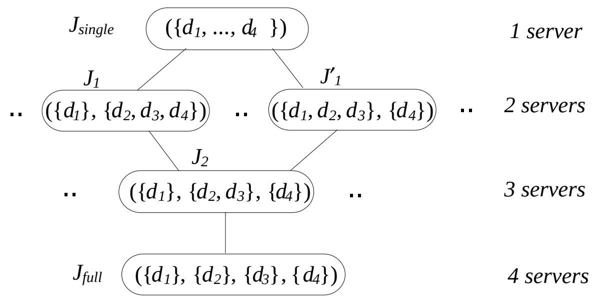


Fig. 3. Part of the decomposition lattice.

3.2 Single-Point Pruning

Similar to BDS and IDS, the client incrementally retrieves points from the servers through sorted accesses. It can also retrieve the local coordinates of points by a random access to the corresponding server. During this process, the client maintains an *anchor point* P_{anc} that is expected to minimize the transmission of false hits (i.e., non-skyline points). The choice of P_{anc} has a significant effect on performance. We first investigate how we can achieve pruning using a single point P_{anc}, for arbitrary vertical decompositions.

Observation 1. *A server N_i does not have to perform sorted accesses for points locally dominated by P_{anc}. If some of these points are in the skyline, they are not dominated by P_{anc} in another subspace D_j, and they will be transmitted by N_j.*

Revisiting the example of Fig. 1, assume that the client has obtained a single point B from both servers, which becomes P_{anc}. B locally dominates all other points in D_1, but only D in D_2. Consequently, N_1 does not need to transmit any other record, unless explicitly requested by the client. On the other hand, server N_2 must send the local coordinates of A, C, E, F, G,H (they are incomparable with B in D_2). The client must ask for the D_1 coordinates of these points from N_1, and compute the skyline among all received records. Points dominated by B globally (in this case, only D) are dominated in each subspace, and hence their transmission is avoided from all servers. The natural question is: which is the anchor point with the highest pruning power (i.e., that can eliminate the largest number of points from skyline consideration)? We refer to the number of points globally dominated by a record P as the *global dominance count* $dom(P)$ of P.

Observation 2. *The optimal anchor is the point P_{maxDC} with the highest dominance count.*

Given that "whenever a point P dominates another Q globally, P must dominate Q in every subspace", P_{maxDC} can eliminate the largest number of points from all servers. Unfortunately, it is impossible to discover P_{maxDC} due to the distributed nature of the problem. Specifically, the global dominance counts cannot be computed in advance since the attributes reside in different servers. Let the *local dominance count* $dom_i(P)$ of P (at server N_i) be the number of points locally dominated by P (in subspace D_i). Even if all $dom_i(P)$ were obtained at a preprocessing step at each server N_i (e.g., using a top-k dominating algorithm [16] locally), they would be of limited help. For instance, in Fig. 1, A dominates 3 and 4 points in subspaces D_1 and D_2, respectively, whereas C dominates 2 (in D_1) and 2 (in D_2). Although A has higher local dominance counts than C in both subspaces, C is a better anchor A because it eliminates both G and H in the 4D space (i.e., $dom(C)$=2), whereas A prunes a single point F (i.e., $dom(A)$=1). Hence, we assume that local dominance counts are not pre-computed; instead, each server only stores the attributes of each record. Although we cannot determine the optimal anchor P_{maxDC}, the following observation provides useful guidelines.

Observation 3. *P_{anc} should be a global skyline point.*

Consider that this is not the case and let $P \in SKY$ be any point that dominates P_{anc} (such a point has to exist; otherwise P_{anc} would belong to SKY). Then, $dom(P) \geq dom(P_{anc})+1$; i.e., by choosing P instead of P_{anc} as the anchor, we can prune more points. Having established that $P_{anc} \in SKY$, the next goal is how to obtain such a point using only the available coordinate information, and without first computing the skyline. To achieve this, we choose as P_{anc} the point that minimizes a function f on the coordinates. We first define the class of functions that we will restrict our attention to:

Definition 1. A function f is *Pareto-consistent* if, whenever P dominates Q, we have that $f(P) < f(Q)$.

Indeed, consider the point $P_{min} \in DS$ that minimizes f. If there was another point P that dominated P_{min}, then, by the definition of Pareto-consistency, $f(P) < f(P_{min})$ and P_{min} cannot minimize f, which is a contradiction. As a result, selecting P_{min} as the anchor respects the desired property of Observation 3.

Observation 4. *The point $P_{min} \in DS$ that minimizes a Pareto-consistent function belongs to the global skyline.*

A notable example of a Pareto-consistent function is the *sum* operator. In order to effectively find P_{anc}, every server N_i must locally sort the records in ascending order of the value of f in the local subspace D_i(e.g., the sum of coordinates in D_i). If P is locally better than Q according to f, P has (locally) greater priority over Q, and will appear before Q in the list. In the remainder, we will refer to f as the target function, or the f-criterion.

Note that common functions such as *max* and *product* are not Pareto-consistent. For instance, a point (e.g., (2, 3)) that minimizes the *max* coordinate is not in the skyline, if there is another (e.g., (1,3)) which has the same maximum, but smaller coordinates on other dimensions. However, such functions can also be used as f-criteria by breaking ties using the *sum*. In case of *max*, every server locally sorts the records in ascending order of their local *max* value. Whenever two records have the same value, the point with the lowest *sum* of coordinates precedes the other on the list. This ensures that the resulting ordering has the desired property that a point cannot be locally dominated by points that come before on the list. The best choice for the f-criterion depends on the data distribution. Figure 4 shows an example, where P_{anc} = (2,2) is the point with minimum *max* coordinate and P'_{anc} (0,3) is the point with the minimum *sum* of coordinates. If the exclusive dominance region of P_{anc} contains more points than that of P'_{anc}, then P_{anc} is a better anchor than P'_{anc}, and vice versa.

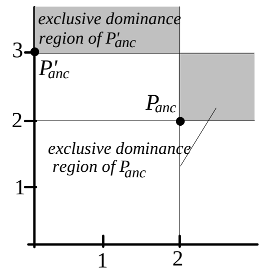


Fig. 4. Pruning power of different anchor points.

A last point of interest is how the decomposition affects the pruning process. The following result shows that the actual decomposition is not important.

Lemma 1. *Let $J = (D_1, \ldots, D_m)$ be an arbitrary decomposition. If we perform single-point pruning with an anchor $P_{anc} \in DS$, then the set of pruned points PP_J for J is the same as the set of pruned points $PP_{Jsingle}$ for J_{single}, i.e., $|PP_J| = |PP_{Jsingle}|$.*

Proof. In the centralized setting, the unique server will prune exactly those points that are globally dominated by P_{anc}. On the other hand, given a distributed decomposition J, each server N_i that stores subspace D_i will transmit all points that are not locally dominated by P_{anc} and only them. Therefore, a point will never be transmitted to the client if and only if it is pruned in every subspace, or, equivalently, if it dominated by P_{anc} in every subspace. Thus, $|PP_J|$ will be exactly the set of points dominated by P_{anc} globally, and coincides with $|PP_{Jsingle}|$. □

Since i) all decompositions are refinements of J_{single} in the decomposition lattice, and ii) the set of pruned points for J_{single} simply consists of the points that are globally dominated by P_{anc}, we obtain the following corollary.

Corollary 1 (Decomposition-Independence Principle). *The pruning power of an anchor $P_{anc}\in DS$ is independent of the decomposition and equal to the dominance count of P_{anc} in the global space D.*

Corollary 1 implies that the crucial factor for pruning power is the *f*-criterion applied, rather than the decomposition. However, different decompositions (of the same space), using the same target function *f* (e.g., *sum*), may yield different performance if there are multiple points minimizing *f* (e.g., there are several candidate P_{anc} minimizing the *sum* of coordinates, and the one chosen depends on the decomposition).

3.3 Multi-Point Pruning

During the selection of P_{anc}, the client may receive numerous points. Intuitively, we could take advantage of these points in order to extend the pruning process. For instance, in the running example, the set {*A*, *B*} can eliminate *F* and *D*, whereas either *A* or *B* alone disqualify a single point (*F* and *D*, respectively). Interestingly, it turns out that multi-point pruning is a rather complicated problem. In the following, we investigate the theoretical foundations of pruning with several points. Let *VS* be the set of (visited) points received by the client before finding P_{anc}. Our goal is to select a *pruning set* $PS\subseteq VS$ that eliminates a large number of points, so that the corresponding records will not be transmitted to the client. Similarly to Observation 3 of single-point pruning, we aim at a pruning set that is a subset of the global skyline *SKY*.

We first focus on the case of two pruning points P_1, $P_2\in SKY$ using the example of Fig. 5, assuming two subspaces D_1= {d_1, d_2}, D_2 = {d_3, d_4} and five records Q_1-$Q_5\in DS$. Let $gdr(P)$ be the *global dominance region* of *P*, which contains all points globally dominated by *P*. Fig. 5 illustrates the projections of $gdr(P_1)$ and $gdr(P_2)$ in the two subspaces. P_1 prunes Q_1 and Q_4 since they are in $gdr(P_1)$ in both D_1 and D_2. Similarly, P_2 eliminates Q_2, $Q_5\in gdr(P_2)$. To utilize both P_1 and P_2, observe that discarding all points in the union of $gdr(P_1)$ and $gdr(P_2)$ may incur false misses. For example, $Q_3\in gdr(P_1)\cup gdr(P_2)$, but Q_3 is incomparable with both P_1 (since $Q_3.d_2<P_1.d_2$) and P_2 (since $Q_3.d_3<P_2.d_3$). On the other hand, pruning with the intersection $gdr(P_1)$ and $gdr(P_2)$ cannot eliminate more points compared to using either P_1 or P_2; in Fig. 5, none of Q_1-Q_5 lies inside $gdr(P_1)\cap gdr(P_2)$.

We can utilize both P_1 and P_2 based on the union of $gdr(P_1)$ and $gdr(P_2)$ in one subspace, and their intersection in the remaining subspaces. For instance, server N_2 avoids transmission of all 5 points because they are inside the projection of $gdr(P_1)\cup gdr(P_2)$ in D_2. N_1 discards Q_4 and Q_5 that are locally dominated by both P_1 and P_2 in D_1, and sends the rest (Q_1-Q_3) to the client. This approach prunes Q_4 and Q_5, which are dominated by either P_1 (i.e., Q_4) or P_2 (Q_5). Note that with *PS* = {P_1, P_2}, the set of eliminated points is different from that using either *PS* = {P_1}(i.e., {Q_1, Q_4}), or *PS* = {P_2} (i.e., {Q_2, Q_5}). Moreover, increasing the cardinality of *PS* does not always achieve higher pruning power. Hence, the selection of an appropriate *PS* is vital for performance.

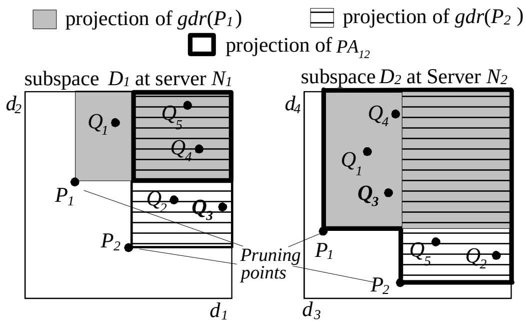


Fig. 5. Pruning with two points.

Definition 2. A *d*-dimensional region *PA* is a *pruning area*, if and only if there cannot be a point *P* in *PA* such that $P\in SKY$.

A pruning area *PA* is used as follows. Each server N_i is aware of the projection $PA.D_i$ of *PA* onto subspace D_i. Similarly to Observation 1 for single-point pruning, *f*or each point *P* whose projection $P.D_i$ falls into $PA.D_i$, N_i transmits *P* to the client if and only if the latter requests information about it (through a random access). In this way, points that are contained in *PA* globally are never sent to the client. The following theorem describes the construction of a new pruning area, by combining two existing ones.

Theorem 1 (Union-Intersection Principle). *Given two pruning areas PA_1 and PA_2, and a subspace D_k ($1\le k\le m$), a new pruning area PA_{new} can be formed as follows:*

$$PA_{new}.D_i = \begin{cases} PA_1.D_i \cup PA_2.D_i & i = k \\ PA_1.D_i \cap PA_2.D_i & 1 \le i \ne k \le m \end{cases}$$

Proof. We show that there cannot be any skyline point in PA_{new}. Let *Q* be an arbitrary point in PA_{new}. According to the above equation, in subspace D_k, it holds that $Q.D_k\in (PA_1.D_k\cup PA_2.D_k)$. Hence, either $Q.D_k\in PA_1.D_k$, or $Q \in PA_2.D_k$ is true. Without loss of generality, assume that $Q.D_k \in PA_1.D_k$. Additionally, in any remaining subspace $D_i\neq D_k$, we have $Q.D_i\in (PA_1.D_i\cap PA_2.D_i)$, which implies that $Q.D_i\in PA_1.D_i$. Therefore, $Q\in PA_1$ holds. Since PA_1 is a pruning area, according to Definition 1, $Q \notin SKY$. □

In Fig. 5, the dominance regions of P_1 and P_2 are combined into PA_{12}, shown in thick frames. The Union-Intersection Principle can be used to construct pruning areas incrementally. Fig. 6 continues the example of Fig. 5, assuming that a new point P_3 is inserted into PS. PA_{12} is combined with $gdr(P_3)$ by taking their union in D_1 and intersection in D_2. The resulting dominance region PA_{123} is shown in thick frames. To prune using PA_{123}, server N_1 avoids the transmission of points that are dominated locally by P_3, or by both P_1 and P_2; N_2 eliminates those locally dominated by either P_1 or P_2. An interesting observation is that PA_{123} is strictly larger than PA_{12}. This occurs because in the subspace D_2, where the intersection takes place, $gdr(P_3)$ completely contains PA_{12}.

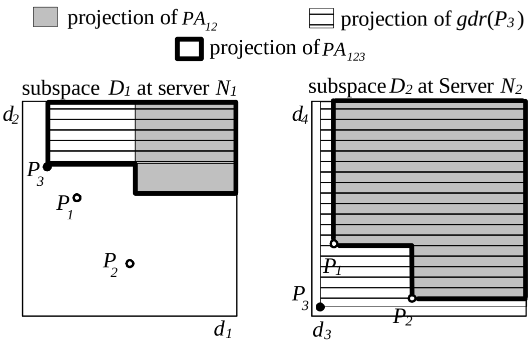


Fig.6. Pruning with three points.

Observation 2 for single-point pruning can be rephrased in the following way for multiple points:

Observation 5. *The optimal pruning set PS is the subset of global skyline points whose pruning area as formed by the Union-Intersection Principle contains the largest possible number of points.*

Computing the optimal PS is infeasible, for the following reasons: (i) there are $2^{|SKY|}$ candidate pruning sets PS; (ii) having determined PS, we can combine its points in all possible permutations using the union-intersection principle, and generate different pruning areas; (iii) given a PA, we cannot accurately measure its benefit in terms of eliminated points without complete knowledge of the entire dataset on all dimensions. Instead, we resort to the greedy algorithm of Fig. 7.

Enlarge_PA(Set VS, Point P_{anc})
1. $PS = \{P_{anc}\}; PA = gdr(P_{anc})$
2. While there is a point $P \in VS$ and a subspace D_k such that (i) $P.D_k \notin PA.D_k$, and (ii) $\forall 1 \leq i \leq m, i \neq k$, $P.D_i$ locally dominates $PA.D_i$
3. Compute PA_{new} according to Theorem 1 with inputs $PA_1=PA$, and $PA_2=gdr(P)$
4. $PS = PS \cup \{P\}$
5. $PA = PA_{new}$

Fig. 7. Algorithm for enlarging the pruning area.

Given the set VS of points received by the client, the algorithm initializes PS to $\{P_{anc}\}$, and PA to $gdr(P_{anc})$, i.e., the global dominance region of the anchor point. Lines 2-5 update PS and PA by adding new pruning points. The loop invariance is that PA can only increase, so that the final pruning area is a superset of that using a single P_{anc}. To achieve this, in each iteration the algorithm identifies a point P and a subspace D_k that satisfy the following two conditions. First, the projection $P.D_k$ of P must lie outside that of the current pruning area $PA.D_k$. This implies that the union of $gdr(P).D_k$ and $PA.D_k$ is strictly larger than $PA.D_k$. Second, in all remaining subspaces $D_i \neq D_k$, $gdr(P).D_i$ must completely contain $PA.D_i$, ensuring that $gdr(P).D_i \cap PA.D_i = PA.D_i$. Accordingly, the new pruning area PA_{new} obtained by combining $gdr(P)$ and PA through the union-intersection principle enlarges PA. The algorithm terminates when PA cannot increase further.

4 Vertical Partition Skyline (VPS)

Given the infeasibility of discovering the optimal anchor, in Section 3 we discussed a selection process based on the minimization of a target function f on the full coordinate space. Next, we clarify how to implement these concepts in our decentralized setting, where each server only maintains information about its local subspace. Section 4.1 introduces the general algorithmic framework. Section 4.2 elaborates on its properties, and Section 4.3 compares it with previous work, qualitatively.

4.1 Framework

Each server N_i maintains the *projection* $P.D_i$ of every point $P \in DS$ in the local subspace D_i. Assuming that f is the target function to be minimized by P_{anc}, each server sorts all points in ascending order of their local f-value, generating a list L_i. Intuitively, any point is locally preferred as an anchor choice to all points that follow it in the list. Furthermore, since f is Pareto-consistent, any point appearing after a point P in the list cannot locally dominate P. We assume that each N_i is capable of both sorted (i.e., fetch the next point in L_i), and random access (i.e., fetch the point with the given ID).

Fig. 8 shows the algorithmic framework of VPS that includes two general phases. The goal of Phase 1 (Lines 2-8) is to obtain the point P_{anc} with the minimum f-value in the global space. For each point P received by a server N_i, the client retrieves the projection $P.D_j$, $\forall N_j \neq N_i$; i.e., the client has complete knowledge of all points $P \in VS$ visited during Phase 1, and can compute their f-value on the global space. If $f(P) < f(P_{anc})$, then P becomes the new anchor. Thus, the anchor is continuously updated to be the point in VS with the minimum f-value. In order to eliminate duplicate transmissions, we assume that every server N_i maintains a bitmap of size $|DS|$, which records whether $P.D_i$ has already been sent to the client through sorted or random access[3]. Note (Line 3) that the server N_i, which encountered the current anchor P_{anc}, stops sorted

[3]Alternatively, duplicates can be avoided using the methods of [1] for scanning multiple sorted lists.

accesses and waits for the other servers to continue with sorted accesses until they also reach P_{anc}.

Phase 1 terminates when the current anchor P_{anc} has been found through sorted access by all servers (Line 8). This implies that each server N_i has already sent to the client all points that precede P_{anc} in list L_i, but has not yet seen points that appear after P_{anc} in the local ordering. Since points that dominate P_{anc} in any subspace D_i precede P_{anc} in list L_i (property of Pareto-consistency), all points that dominate the anchor in any subspace will have been sent to the client by the termination of Phase 1. On the other hand, points that come after P_{anc} on any list L_i are either (i) locally dominated by P_{anc}, or (ii) locally incomparable with P_{anc}. According to our previous discussion, we can avoid transmitting the former, but we have to send the latter, since they are candidate (global) skyline points. The task of retrieving the locally incomparable remaining points is left to Phase 2. Given the set *VS* of points encountered during Phase 1, Phase 2 first selects the *pruning set* $PS \subseteq VS$ according to the greedy algorithm of Fig. 7. This step is optional; e.g., if the client has limited computational power, $PS = \{P_{anc}\}$.

VPS

//Preprocessing step: Each of the *m* servers has locally generated a list $L_i$that orders the points in ascending order of the local *f*-value, where *f* is the target function

1. Initialize P_{anc}=∅, *VS*=*IS*=∅ //*VS* (resp. *IS*) is the set of points retrieved during phase 1 (resp. phase 2)

// Phase 1: Retrieval of initial points and P_{anc}

2. Repeat
3. Choose any server N_i where P_{anc} has not been encountered through sorted access
4. Retrieve from L_i the projection $P.D_i$ of next point $P \notin VS$
5. Obtain all unseen projections $P.D_j$ through random accesses
6. if $f(P) < f(P_{anc})$, then $P_{anc} = P$
7. Add *P* to *VS*
8. Until P_{anc}has been encountered through sorted accesses at every server

// Phase 2: Retrieval of remaining points

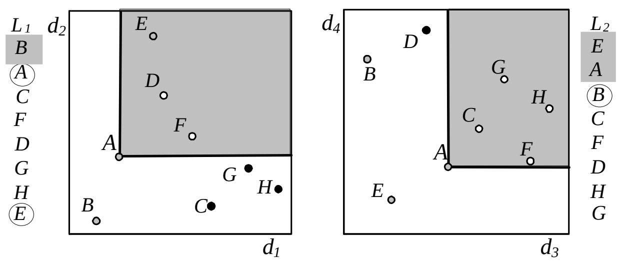
9. Compute pruning set *PS* and pruning area *PA*// multi-point pruning (optional)
10. For each server N_i
11. Compute pruning area $PA.D_i$ that contains records locally dominated by *PS*
12. Retrieve from N_i the projection $P.D_i$ for every point *P* such that: $P \notin \{VS \cup IS\}$ and *P* not in $PA.D_i$
13. Obtain all unseen projections $P.D_j$ through random accesses
14. Add *P* to *IS*

15. Compute the skyline *SKY* among the points in $VS \cup IS$

Fig. 8. General framework of VPS.

After the pruning set has been determined, the client computes, for each server N_i, the pruning area $PA.D_i$, which is the area locally dominated by *PS*. Every server N_i sends the projection $PA.D_i$ for all points *P* such that $P.D_i$ does not fall in $PA.D_i$ and $P.D_i$ has not been transmitted before. The client retrieves the remaining coordinates of these points and inserts them in a set *IS* (for incomparable set). The utilization of bitmaps at the servers ensures that *VS* and *IS* have no duplicates and no overlap. Finally, the client computes the global skyline using the records of $VS \cup IS$. The effectiveness of VPS depends on the size of $VS \cup IS$. Ideally, $VS \cup IS = SKY$, while in the worst case $VS \cup IS = DS$. The set $\{VS \cup IS\} - SKY$ corresponds to false hits, i.e., it is the set of points received by the client although they are not necessary for the skyline computation.

Fig. 9 illustrates VPS on the running example assuming that each server sorts the points in increasing order of the *sum* of their local coordinates. The client retrieves *B* with a sorted access at N_1, and obtains its D_2 coordinates with a random access at N_2. Since *B* is the first point discovered, it becomes the current P_{anc}. The second point obtained (by a sorted access at N_2) is *E*, whose sum of coordinates exceeds that of *B*; thus, *B* remains P_{anc}. Next, the client receives *A* (by a sorted access at N_2), which becomes the new P_{anc}. Since *A* is the next point at N_1, Phase 1 terminates with $P_{anc} = A$. Phase 2 selects a pruning set $PS \subseteq VS = \{A, E, B\}$; for simplicity, assume that $PS = \{A\}$. $PA.D_1$ and $PA.D_2$ correspond to the grey rectangles in Fig. 9a and 9b, respectively. N_1 first transmits the local projections of {*C*, *G*, *H*}; points *B* and *E* have been sent in Phase 1, whereas *D* and *F* fall in $PA.D_1$. The client obtains the remaining coordinates of {*C*, *G*, *H*} from N_2 and inserts these points in *IS*. Similarly, N_2 transmits the local projection *D*, after which *IS* = {*C*, *G*, *H*, *D*}. The skyline *SKY* = {*B*, *E*, *A*, *C*} is computed among the points in $VS \cup IS$ = {*A*, *E*, *B*} ∪ {*C*, *G*, *H*, *D*}. *G*, *H* and *D* are false hits.



(a) Server N_1 (b) Server N_2

Fig 9. Example of VPS.

4.2 Analysis

Compared to BDS/IDS, VPS is more general since (i) it allows for decompositions of arbitrary dimensionality, while BDS/IDS are restricted to the full partitioning setting J_{full}; and, (ii) it enables anchor selection based on a wide range of target functions *f*, whereas BDS selects the anchor based on a round-robin scheme, and IDS can pick the anchor only according to a heuristic that uses the *sum* operator. A more detailed comparison with previous work can be found in Section 4.3. Next, we investigate the properties of VPS. Observation 3 states that the anchor point should belong to *SKY*. The following result ensures this property.

Lemma 2. *The anchor point P_{anc} discovered by VPS is a global skyline point.*

Proof. (by contradiction) Suppose that $P_{anc} \notin SKY$. Thus, there must be a point $P \in SKY$ that dominates P_{anc} globally. This implies that P dominates P_{anc} in at least one subspace D_i. In the corresponding list L_i, P must precede P_{anc}, since L_i is ordered in a Pareto-consistent manner. Hence, VPS would have encountered P before P_{anc}, and would have set it as the anchor point instead, since the former has a lower f-value than the latter (by the Pareto-consistency), leading to a contradiction. □

A desirable property of VPS (and any skyline algorithm) is *progressiveness* [11], which enables the client to determine whether a newly received point is in the global skyline based only on the previously collected records. This is important for real time applications, where the client must output skyline points incrementally, without having to collect all candidates first.

Lemma 3. *VPS is progressive.*

Proof. (by contradiction) Progressiveness can only be violated if there is a point P received by the client, which is dominated by another Q that the client has not collected so far from any server. Indeed, if there is no such point Q, then VPS can immediately determine whether $P \in SKY$ by comparing it against the already received points; otherwise, the client has to wait until it receives Q to conclude that P does not belong to *SKY*. We now prove that that such a point Q can never exist. Let N_i be the server that first encountered P (either at Phase 1 through sorted access or at Phase 2) and subsequently sent it to the client. Since the f-criterion is Pareto-consistent, any point Q that dominates P would have been encountered by N_i before P and would have been sent first, leading to a contradiction. □

The following theorem refers to the soundness and completeness of VPS.

Theorem 2 (Completeness and Soundness). *VPS outputs all global skyline points (completeness) and only those points (soundness).*

Proof. Given an anchor point P_{anc}, we first prove that the client receives a superset of the global skyline points, i.e., $(VS \cup IS) \supseteq SKY$. Recall that during Phase 2 each N_i transmits all points incomparable with P_{anc} in D_i, unless they have been encountered before. Thus, the only potential false misses may occur in the areas that dominate P_{anc}, or are dominated by P_{anc}. Each point P that dominates P_{anc} in subspace D_i has been already transmitted by a sorted access at N_i (because of the Pareto-consistency property), and $P \in VS$. In order for a point P' that is dominated by P_{anc} in D_i to be in *SKY*, there must exist a subspace D_j such that (i) P' dominates P_{anc} in D_j or (ii) P' is incomparable with P_{anc} in D_j. In case (i), $P' \in VS$ because it dominates P_{anc} in D_j and has been found by sorted access at N_j. In case (ii), P' is transmitted by N_j during phase 2, so that $P' \in IS$. Thus, the client obtains all possible skyline points, without false misses.

Soundness is proven by contradiction. Assume that a point P' in the final output of VPS is not in *SKY*, which implies that P' must be globally dominated by an actual skyline point P. Based on the completeness of VPS, P must have been received by the client during Phase 1 or 2. Since the client computes *SKY* using all points of $(VS \cup IS)$, P' is eliminated by P and cannot exist in the final skyline. □

In order to guarantee certain properties of the anchor selection process, we need to place some additional constraints on the class of admissible target functions.

Definition 3. A function f is *distributive*, if and only if for any decomposition $J=(D_1,\ldots,D_m)$ it holds that $f(P)=f(f(D_1),\ldots,f(D_m))$.

This implies that we can distribute the computation of f among an arbitrary number of subspaces: each subspace computes the local f-value, and then a separate entity (the client, in our setting) can collect the local results and produce the global f-value by applying f on the local results. An immediate consequence of the definition is that f is *decomposition-independent*, i.e., for any 2 decompositions $J = (D_1, \ldots, D_m)$ and $J' = (D_{1'}, \ldots, D_{m'})$, we have that $f(f(P.D_1), \ldots, f(P.D_m)) = f (f(P. D_{1'}), \ldots, f(P.D_{m'})) = f(P)$. *Sum, max* and *product* are all distributive.

We define the *min-value* for a function f as $v_f = min\{f(P), P \in DS\}$. Let S_f be the set of points $P \in DS$ for which $f(P) = v_f$. Intuitively, for a given choice of the target function f, v_f is the minimum value of f for any point in *DS*, whereas S_f is the set of points in *DS* that achieve this value. As discussed in Section 3.2, P_{anc} should belong to the set S_f. The next theorem establishes that the anchor point discovered by VPS is indeed minimal with respect to f.

Theorem 3 (Anchor Selection). *If the target function f is distributive, VPS always selects as anchor a point $P_{anc} \in S_f$, irrespective of the actual decomposition $J = (D_1, \ldots, D_m)$.*

Proof. (by contradiction) Recall that v_f is the minimum value of f for any point in *DS*, whereas S_f is the set of points in *DS* that achieve this value. Suppose that VPS finds an anchor $P_{anc} \notin S_f$. Then, we have that $f(P_{anc}) > v_f$. Consider now any point $P \in S_f$ (i.e., $f(P) = v_f$). P has to appear after P_{anc} in all lists, otherwise the algorithm would have encountered it through a sorted access, and would have chosen it instead as anchor because

$f(P) < f(P_{anc})$. Since points are sorted in ascending order of their local f-value in the corresponding subspace, it holds that $f(P_{anc}.D_i) \leq f(P.D_i)$, for every $i \in \{1, \ldots, m\}$. Given that f is distributive we have:

$$f(P_{anc}) = f(f(P_{anc}.D_1), \ldots, f(P_{anc}.D_m))$$
$$\leq f(f(P.D_1), \ldots, f(P.D_m)) = f(P) = v_f.$$

This leads to a contradiction since we assumed that $f(P_{anc}) > v_f$. □

In general data distributions, the set S_f potentially consists of several points. If, however, S_f contains a unique point P_f, Theorem 3 ensures that P_f will always be selected as anchor by the VPS framework. On the other hand, Corollary 1 guarantees that pruning with a given anchor will achieve the same pruning power, independent of the actual decomposition. As a consequence, we obtain the following result for the single-point pruning version of VPS.

Corollary 2. *If the target function f is distributive and S_f consists of exactly one point, then the single-point pruning version of VPS will always achieve exactly the same pruning power, irrespective of the global space decomposition.*

4.3 Qualitative Comparison with Previous Work

We first compare VPS with the state-of-the-art centralized skyline algorithm SaLSa [3]. Clearly, the two methods apply to different contexts, and do not compete directly with each other. However, the comparison is interesting because both methods involve an anchor point to prune false hits. Furthermore, [3] formally proves that SaLSa always selects the *optimal* point with maximum pruning power in its setting. A natural question is whether the anchor point chosen by SaLSa has similar optimality guarantees in the distributed environment of VPS. The answer is negative, due to the fact that the two frameworks have different goals. In particular, SaLSa aims at minimizing the number of points *retrieved* by the server, while VPS minimizes the number of points *transmitted* to the client.

We elaborate on the differences between the two frameworks with the example of Fig. 10. We first describe the functionality of SaLSa, assuming the decomposition J_{single}. Recall from Section 2 that SaLSa sorts points in increasing order of their minimum coordinate and maintains P_{stop} as the point with the minimum, maximum coordinate. Search terminates after discovering a record Q such that every point after Q is guaranteed to be dominated by P_{stop}. Let (x,y) be the 2D coordinates of the point $P_{min\text{-}max}$ with the minimum, maximum coordinate, i.e., the final P_{stop}. Without loss of generality assume that $x<y$; x and y divide the data-space into R_1 to R_9. For each rectangle, Fig. 10a characterizes the type of points inside. For instance, R_7 (resp. R_3) contains points whose d_1 (resp. d_2) coordinate is less than x, and therefore have been visited before $P_{min\text{-}max}$. R_6 contains points whose coordinates are larger than x, and are visited after $P_{min\text{-}max}$. R_1, R_2, R_4 and R_5 (shown in grey) must be empty because if they contained a point, this would be $P_{min\text{-}max}$ (it would minimize the maximum coordinate). When $P_{min\text{-}max}$ is found, the partitions marked as "after" have not been yet visited. Thus, any single-list sort-based algorithm has to continue until it finds a point whose both coordinates are at least (y,y), i.e., only points in R_9 are pruned. Although points in R_8 are false hits, they cannot be directly eliminated because they are interleaved with potential skyline points from R_6 after $P_{min\text{-}max}$ in the sorted list.

Next, we show how VPS would work in this example, assuming we are interested in the minimization of the *max*-criterion. Since *max* is distributive, VPS selects the anchor point $P_{anc} = P_{min\text{-}max}$, regardless of the domain decomposition, according to Theorem 3. This anchor prunes points in the dominated area $R_8 \cup R_9$. Note that the pruning area $R_8 \cup R_9$ of VPS is larger than that of SaLSa (just R_9). However, pruning in SaLSa means that points in R_9 will never be visited; in VPS it implies that points in $R_8 \cup R_9$ will never be transmitted to the client, while the issue of whether the server visits them locally and then discards them is irrelevant. For instance, the server could visit all remaining points in its corresponding list after identifying P_{anc}, and simply ignore those records that are locally dominated by the anchor.

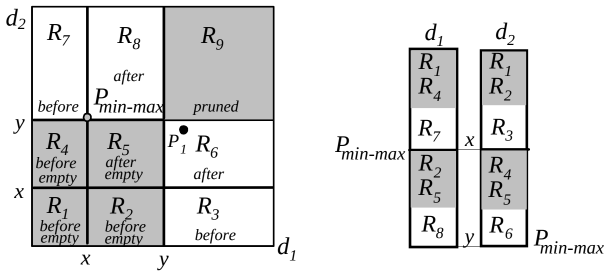


Fig. 10. Pruning with $f_s = minC$ and $f_A = min\text{-}max$.

Next, we compare the round-robin visiting scheme of BDS with the VPS framework, assuming the two dimensions are vertically decomposed into two servers and sorted in increasing coordinate order. In this direction, suppose that $P.D_i = rank_i(P)$, i.e., the coordinate of the first point (on d_1 or d_2) is 1, of the second point 2, and so on. In this case, R_7 contains exactly $x-1$ points, because (i) all points with d_1 coordinate smaller than x lie in $R_1 \cup R_4 \cup R_7$ and (ii) both R_1 and R_4 are empty. Similarly, R_3, $R_7 \cup R_8$ and $R_3 \cup R_6$ contain $x-1$, $y-1$, $y-1$ points, respectively. The client uses BDS to retrieve point coordinates in a round-robin fashion. Fig. 10b illustrates the rectangles of Fig. 10a explored before the discovery of $P_{min\text{-}max}$ by both servers. Specifically, excluding the empty rectangles, N_1 visits the points of R_7, finds $P_{min\text{-}max}$, and then continues in R_8. Similarly, N_2 visits the points of R_3 and R_6 before finding $P_{min\text{-}max}$. Because of the round-robin order of BDS and the fact that $R_7 \cup R_8$ and $R_3 \cup R_6$ contain the same number (i.e., $y-1$) of points, when N_2 encounters $P_{min\text{-}max}$, N_1 has just finished scanning R_8. Since the non-empty rectangles visited by both servers have zero overlap, $P_{min\text{-}max}$ constitutes the anchor point of BDS. Similar to the single list case, points in R_9 are eliminated, but false hits in R_8 are considered during skyline computation.

However, BDS does not find $P_{min\text{-}max}$ as the anchor, when the assumption that $P.D_i = rank_i(P)$ no longer holds. In fact, the pruning power of BDS's anchor is highly sensitive to data skewness. For instance, suppose that in Fig. 10, $R_3 \cup R_6$ contains more points than $R_7 \cup R_8$. Then, after N_1 exhausts R_7 and R_8, it continues to $R_3 \cup R_6 \cup R_9$, while N_2 is still scanning R_6. Hence, it is possible that both servers reach a common point $P_1 \in R_6$ before N_2 encounters $P_{min\text{-}max}$. BDS then sets P_6 as the anchor, which is expected to have less pruning power than $P_{min\text{-}max}$. On the other hand, VPS suspends sorted accesses at the servers where the current P_{anc} has been already encountered through sorted access, and does not suffer from the aforementioned shortcomings.

Next, we compare VPS with IDS. Recall from Section 2 that IDS sets as P_{anc} the point P that minimizes the total number of sorted accesses SA_P required for all servers to reach P. Let SA_i be the number of sorted accesses that server N_i has already performed. The number of additional sorted accesses necessary for N_i to reach P is $|DS|-dom_i(P)-SA_i$, where $dom_i(P)$ in a 1D subspace is the number of points succeeding P in the 1D order. Therefore, the total number of sorted accesses SA_P for all servers to reach P is:

$$SA_P = \sum_{i=1}^{m}\left(|DS| - dom_i(P) - SA_i\right) = m \cdot |DS| - \sum_{i=1}^{m} dom_i^{SUM}(P) - \sum_{i=1}^{m} SA_i$$

In the above equation, only the second term depends on P; the other two terms are the same for all points, and, thus, can be treated as constants. Therefore, minimizing SA_P is equivalent to maximizing $\sum_{i=1}^{m} dom_i(P)$. However, as argued in Section 3.2, the sum of local dominance counts is not an accurate estimator of the actual global dominance count, and IDS may choose as anchor a point with rather poor pruning power. As opposed to IDS, which is restricted to minimization of a single function, VPS can be used with any function depending on the problem characteristics. Moreover, it supports arbitrary decompositions, whereas BDS/IDS are limited to full partitioning.

Finally we compare VPS with STA [8]. We emphasize that pruning has different goals in the two algorithms. Similar to SaLSa [3], a point pruned by STA is not accessed, whereas in VPS pruning signifies a point that is not transmitted to the client (even if it is retrieved in some servers). On the other hand, STA is allowed to "partially" prune a point P, in the sense that avoiding retrieving index nodes containing P in some (but not all) subspaces still leads to I/O savings, while in VPS none of the servers transmits pruned points. Despite these differences, there exist some algorithmic similarities between the two methods. For single-point pruning, STA sets as anchor the point that minimizes the sum of its coordinates in all dimensions. Similar to the case of BDS, such coordinates are highly sensitive to data skewness. Meanwhile, the sum of coordinate values can be viewed as an approximation of the *sum* of local dominance counts used in IDS, which, in turn, is not an accurate estimate of the global dominance count as explained above.

Concerning multi-point pruning, the most prominent difference between VPS and STA is that the former guarantees that multi-point pruning is at least as effective as single-point pruning, whereas the latter may lead to worse performance compared to its single-point counterpart [8]. In addition, the common-point condition used in STA is rather restrictive. For instance, the pruning area in Figs. 5-6 cannot be expressed using the common point condition. Finally, the computation of the pruning area in STA takes exponential time to the number of subspaces as described in Section 2, whereas algorithm *Enlarge_PA* shown in Fig. 7 incurs negligible computation cost.

5 EXPERIMENTS

For our experiments, we use the following datasets. (i) *NBA* (*www.basketballreference.com*) contains 17 statistics about 21K basketball players, e.g., points scored, rebounds. (ii) *Household* (*www.ipum.org*) includes 127K tuples. Each record has 6 attributes that store the percentage of an American family's annual income spent on: gas, electricity, water, heating, insurance, and property tax. (iii) *Corel* (*kdd.ics.uci.edu*) includes data about 68K images. For each image, 9 attributes describe the mean, standard deviation, and skewness of the image's pixels, in the hue, saturation and value channels. (iv) *IND* is a synthetic dataset containing 300K points with 8 independent dimensions, created with the generator of [4]. Table 2 summarizes the properties and skyline cardinality of each dataset.

TABLE 2
STATISTICS ON THE EXPERIMENTAL DATA

Dataset	Cardinality	Dimensionality	Skyline Size
NBA	21,378	17	1195
Household	127,931	6	5774
Corel	68,040	9	1533
IND	300,000	8	9456

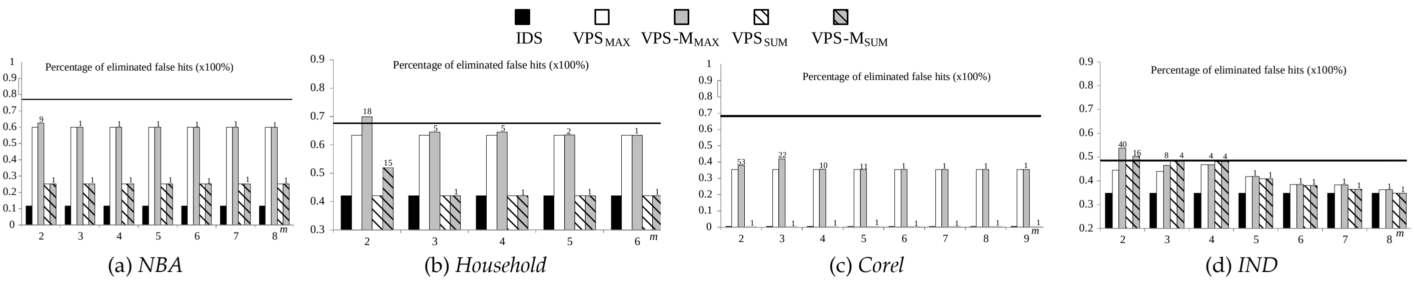
As a benchmark, we use IDS because, according to [2], it consistently outperforms BDS. The original proposal of IDS only applies to the full decomposition J_{full} where each server maintains exactly one dimension. We extend the method to capture arbitrary decompositions through *virtual servers*. Specifically, each physical server that stores more than one dimension acts as multiple virtual 1D servers (e.g., a server that has 4 dimensions creates 4 different lists, each storing the values of the corresponding attribute in ascending order). For VPS, we employ the *sum* and *max* operators as target functions, yielding VPS_{SUM} and VPS_{MAX}, respectively. VPS_{SUM} (resp. VPS_{MAX}) picks as P_{anc} the point $P_{min\text{-}sum}$ (resp. $P_{min\text{-}max}$) that minimizes the sum of coordinates (resp. the maximum coordinate).VPS_{SUM} and VPS_{MAX} apply pruning using one point (i.e., the anchor). In addition, we implement their optimized versions $\text{VPS-M}_{\text{MAX}}$ and $\text{VPS-M}_{\text{SUM}}$ that utilize multi-point pruning as described in Section 3.3.

We vary the number *m* of servers, and compare IDS and VPS in terms of the percentage of pruned false hits. We omit results for CPU and I/O costs at the client because they are dominated by the local skyline computation module (line 15 in Fig. 8), which can be based on any centralized skyline algorithm, e.g., [4][5][9][10][11], and is orthogonal to this work. We follow two approaches to distribute the attributes of a dataset among the servers: *balanced* and *unbalanced*. In the former, every server stores a similar number of dimensions, whereas in the latter approach, two servers (one server when *m*=2) maintain the majority of dimensions, and the rest of the servers store one dimension each. For instance, to distribute the NBA dataset (17 dimensions) to four servers, *balanced* assigns 5, 4, 4, 4 dimensions to the servers, while *unbalanced* requires the servers to store 1, 1, 8 and 7 dimensions.

Fig. 11 evaluates the algorithms on all datasets for balanced decompositions. In each diagram, the horizontal line corresponds to the pruning efficiency of the optimal anchor point P_{maxDC}. Note that as discussed in Section 3.2, it is infeasible to compute P_{maxDC}; we only include its pruning power as the theoretical limit of single-point pruning. Moreover, for experiments involving multi-point pruning, we illustrate the cardinality of the pruning set with a number above each column.

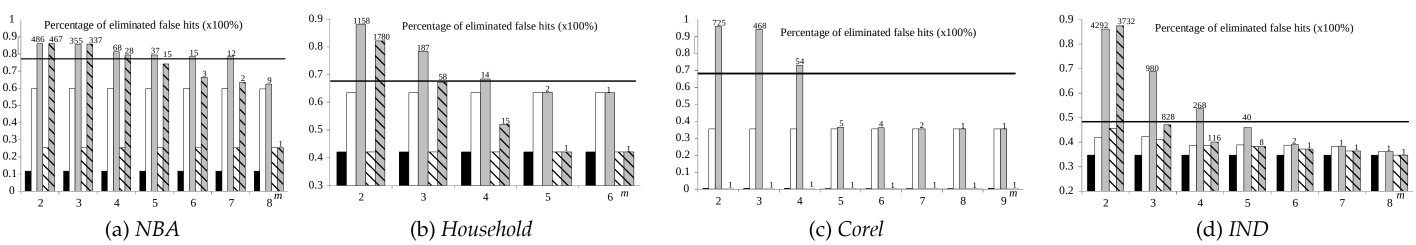
All VPS implementations consistently outperform IDS. Specifically, IDS employs a heuristic in order to identify the point that needs the smallest number of remaining sorted accesses. Whether the heuristic indeed returns this point heavily depends on the data distribution, so the behavior of IDS is unpredictable. In all settings, however, it prunes significantly fewer points than P_{maxDC}, and in some cases (e.g., *Corel*) almost no point. On the other hand, VPS_{MAX} has, in general, the best performance; since the anchor point $P_{min\text{-}max}$ has reasonably small attributes on all dimensions, its dominance region covers a large area in the global space, and is thus expected to dominate several points. In particular, the pruning power of $P_{min\text{-}max}$ is close to the optimal P_{maxDC} in most settings except for *Corel*. VPS_{SUM} is based on a design choice that is less robust because $P_{min\text{-}sum}$ may have large attributes on some dimensions, yielding a small dominance region with limited pruning power (e.g., in *Corel*). It slightly outperforms VPS_{MAX} only when the distribution is independent. Another observation concerns the similarity in the behavior of IDS and VPS_{SUM} for some settings. This is expected because the heuristic of IDS also employs the *sum* operator; instead, however, of taking the sum over all coordinate values, it is restricted to only those where the current anchor has not been encountered yet, and is thus less accurate.

Regarding the decomposition-independence principle, for all real datasets, we observed through manual examination that the optimal anchor point, in terms of



(a) *NBA* (b) *Household* (c) *Corel* (d) *IND*

Fig. 11. Pruning efficiency vs. number of servers *m* for *balanced* decompositions.



(a) *NBA* (b) *Household* (c) *Corel* (d) *IND*

Fig. 12. Pruning efficiency vs. number of servers *m* for *unbalanced* decompositions.

both *max* and *sum*, is unique. Corollary 4 guarantees that VPS always achieves exactly the same pruning power, irrespective of the actual decomposition. This is evident in all real datasets, where each implementation of VPS achieves equal pruning powers across all partitions. On the other hand, *IND* contains several points that optimize the target function. i.e., S_f is non-singleton. In this case, the pruning power fluctuates according to the selected anchor among the points of S_f. Finally, multi-point pruning is not particularly effective in the settings of Fig. 11, except for a very small number (i.e., 2 or 3) of servers. We will elaborate more on this point in the next paragraph.

Fig. 12 illustrates the pruning power of the evaluated methods on the same datasets for unbalanced decompositions. Comparing Figs. 11 and 12, observe that multi-point pruning is significantly more effective when the dimensions are distributed in an unbalanced manner. This happens because the algorithm for enlarging pruning areas requires a point to dominate P_{anc} locally in m-1 subspaces. In the unbalanced setting, most servers maintain single-dimensional subspaces, meaning that every point P encountered before P_{anc} in the sorted list dominates P_{anc} locally. Consequently, VPS is more likely to find pruning points in the *unbalanced* settings compared to the *balanced* ones. Multi-point pruning becomes less pronounced as the number of servers increases, since it becomes more difficult to find a new point that locally dominates P_{anc} in m-1 subspaces.

An important observation is that in many settings where VPSMAX (resp., VPSSUM) performs poorly, multi-pruning eliminates a large portion of false hits. We found through manual examination that the pruning set of VPS-MMAX (resp., VPS-MSUM) contains points that dominate significantly more false hits than the anchor. For such cases, the integration of multiple pruning points greatly improves the robustness of VPS-MMAX (resp., VPS-MSUM). A notable example is *Corel*, where VPS-MMAX significantly outperforms the optimal pruning power of single-point pruning (i.e., with P_{maxDC}). As a concluding remark, we emphasize that although Corollary 2 implies that the actual decomposition does not affect the pruning power in the case of single-point pruning (for singleton S_f), the above discussion on multi-point pruning demonstrates clearly that small and unbalanced decompositions should be preferred to the bigger and more balanced ones, because in the former case multi-point pruning becomes significantly more effective.

6 Conclusion

Skyline computation on vertical decompositions is complicated because the global skyline may contain records that are locally dominated in each subspace. The only existing work assumes that each server maintains a single dimension. In this paper, we propose VPS, a general framework for decompositions of arbitrary dimensionality. VPS first selects an *anchor point*P_{anc} expected to have large pruning power based on a user-specified criterion. Given P_{anc}, and potentially additional points, it then constructs a pruning area such that all records falling in this area are discarded. Finally, the skyline is computed using the rest of the points. In addition to the basic framework, we investigate the performance of sorting and pruning functions, their relationship to previous work, and the benefits of pruning with multiple points. The effectiveness of our contributions is confirmed in various experimental settings using real and synthetic datasets.

References

[1] Akbarinia, R., Pacitti, E., Valduriez, P. Best Position Algorithms for Top-*k* Queries. *VLDB*, 2007.

[2] Balke, W.-T., Güntzer, U., Zheng, J.X. Efficient Distributed Skylining for Web Information Systems. *EDBT*, 2004.

[3] Bartolini, I., Ciaccia P., Patella, M. Efficient Sort-Based Skyline Evaluation. *ACM TODS*, 33(4): 1-45, 2008.

[4] Börzsönyi, S., Kossmann, D., Stocker, K. The Skyline Operator. *ICDE*, 2001.

[5] Chomicki, J., Godfrey, P., Gryz J., Liang D. Skyline with Pre-Sorting. *ICDE*, 2003.

[6] Cui, B., Lu, H., Xu, Q., Chen, L., Dai, Y., Zhou, Y. Parallel Distributed Processing of Constrained Skyline Queries by Filtering. *ICDE*, 2008.

[7] Datta, A., Vandermeer, D., Celik, A., Kumar, V. Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users. *ACM TODS*, 24(1):1-79, 1999.

[8] Dellis, E., Vlachou, A., Vladimirskiy, I., Seeger, B., Theodoridis, Y. Constrained Subspace Skyline Computation. *CIKM*, 2006.

[9] Godfrey, P., Shipley, R., Gryz, J. Algorithms and Analyses for Maximal Vector Computation. *VLDB J.*, 16(1): 5–28, 2007.

[10] Lee, K., Zhang, B., Li, H., Lee, W.-C. Approaching the Skyline in Z Order. *VLDB*, 2007.

[11] Papadias, D., Tao, Y., Fu, G., Seeger, B. Progressive Skyline Computation in Database Systems. *ACM TODS*, 30(1): 41–82 2005.

[12] Tao, Y., Xiao, X., Pei, J. SUBSKY: Efficient Computation of Skylines in Subspaces. *ICDE*, 2006.

[13] Vlachou, A., Doulkeridis, C., Kotidis, Y. Angle-based Space Partitioning for Efficient Parallel Skyline Computation. *SIGMOD*, 2008.

[14] Vlachou, A., Doulkeridis, C., Kotidis, Y., Vazirgiannis, M. SKYPEER: Efficient Subspace Skyline Computation over Distributed Data. *ICDE*, 2007.

[15] Wang, S., Vu Q.H., Ooi, B.C., Tung, A.K.H., Xu, L. Skyframe: A Framework for Skyline Query Processing in Peer-to-Peer Systems. *VLDB J.*, 18(1): 345–362, 2009.

[16] Yiu, M. L., Mamoulis, N. Efficient Processing of Top-*k* Dominating Queries on Multi-Dimensional Data. *VLDB*, 2007.

[17] Zhang, Z., Cheng, R., Papadias, D., Tung A. Minimizing the Communication Cost for Continuous Skyline Maintenance. *SIGMOD*, 2009.

George Trimponias obtained a five-year diploma in Electrical and Computer Engineering from the National Technical University of Athens, Greece. He then worked as a research associate at the Institute of Applied Informatics and Formal Description Methods at the Karlsruhe Institute of Technology, Germany. Currently, he is a PhD candidate at the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology (HKUST). His research interests include spatio-temporal databases and applications of game theory in database systems.

Ilaria Bartolini is an Assistant Professor with the DEIS department of the University of Bologna (Italy). She graduated in Computer Science (1997) and received a Ph.D. in Electronic and Computer Engineering (2002) from the University of Bologna. In 1998 she spent six months at CWI in Amsterdam as a junior researcher. In 2004 she was a visiting researcher for three months at NJIT in Newark, NJ. In January-April 2008 and in September-November 2010 she was visiting HKUST. Her current research mainly focuses on collaborative filtering, learning of user preferences, similarity and preference-based query processing in large databases, and retrieval and browsing of image and video collections. Ilaria Bartolini has published about 40 papers in major international journals and conferences. She served in the program committee of several international conferences and workshops. She is a member of the ACM SIGMOD, the IEEE, and the IEEE Computer Society.

Dimitris Papadias is a Professor of Computer Science and Engineering, Hong Kong University of Science and Technology. Before joining HKUST in 1997, he worked and studied at the German National Research Center for Information Technology (GMD), the National Center for Geographic Information and Analysis (NCGIA, Maine), the University of California at San Diego, the Technical University of Vienna, the National Technical University of Athens, Queen's University (Canada), and University of Patras (Greece). He serves or has served in the editorial boards of the VLDB Journal, IEEE Transactions on Knowledge and Data Engineering, and Information Systems.

Yin "David" Yang is a Research Scientist at the Advanced Digital Sciences Center (ADSC), Singapore, and a Principle Research Affiliate at the University of Illinois at Urbana Champaign, Illinois, USA. He obtained his PhD in Computer Science from the Hong Kong University of Science and Technology (HKUST) in 2009. Before joining ADSC, David worked as an instructor for undergraduate courses at HKUST, and later as a post-doc at the University of Hong Kong. His research interests include database security and privacy, spatial and temporal query processing, and distributed systems.