Efficient Parallel Skyline Query Processing for High-Dimensional Data

Mingjie Tang,Yongyang Yu, Walid G. Aref, *Senior Member, IEEE,*
Qutaibah M. Malluhi, Mourad Ouzzani, *Member, IEEE,*

Abstract—Given a set of multidimensional data points, skyline queries retrieve those points that are not dominated by any other points in the set. Due to the ubiquitous use of skyline queries, such as in preference-based query answering and decision making, and the large amount of data that these queries have to deal with, enabling their scalable processing is of critical importance. However, there are several outstanding challenges that have not been well addressed. More specifically, in this paper, we are tackling the data straggler and data skew challenges introduced by distributed skyline query processing as well as the ensuing high computing cost of merging skyline candidates. We thus introduce a new efficient three-phase approach for large scale processing of skyline queries. In the first preprocessing phase, the data is partitioned along the Z-order curve. We utilize a novel data partitioning approach that formulates data partitioning as an optimization problem to minimize the size of intermediate data. In the second phase, each computation node partitions the input data points into separate sets, and then performs the skyline computation on each set to produce skyline candidates in parallel. In the final phase, we build an index and employ an efficient algorithm to merge the generated skyline candidates. Extensive experiments demonstrate that the proposed skyline algorithm achieves more than one order of magnitude enhancement in performance compared to existing state-of-the-art approaches.

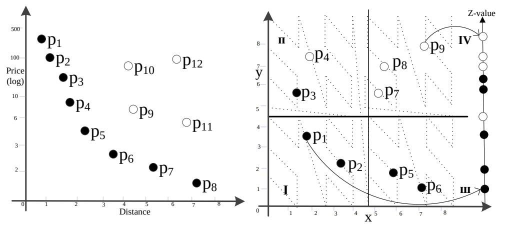
Index Terms—skyline query, query processing, high dimensional data, parallel

1 Introduction

Skyline query processing has been widely studied in centralized systems [1], [2], [3], [4], [5]. A skyline query returns a set of data points that are not dominated by any other points in a given dataset. In a multidimensional space, a point dominates another point if it is better in at least one dimension and not worse in all other dimensions. Figure 1(a) shows a sample dataset of hotels in New York City. The x-axis indicates the distances to the downtown area and the y-axis indicates daily rates of the hotels. For example, Hotel p_1 is the nearest to downtown but has the most expensive daily rate. Hotel p_5 dominates Hotel p_9 since p_5 has a shorter distance and a lower rate.

For very large data sets (e.g., more than 10 million points), skyline queries require expensive computations and exhibit slow response times. To improve the runtime performance, a natural idea is to parallelize the computation. Skyline queries [6], [7], [8], [9], [6], [10] can be executed in parallel via the following three stages: (i) partition the input datasets into blocks with equal size, (ii) perform skyline computations in each worker and output skyline candidates, and (iii) merge skyline candidates to get the final skyline sets. There have been several proposed techniques along these lines. A grid-based partitioning scheme (e.g., [9], [11], [12]), recursively divides some dimensions of the data into multiple parts, and computes skyline candidates for each partition. An angle-based partitioning scheme [8] assumes that skyline points are usually located around the origin. Therefore, skyline points can be distributed in a balanced way if the partitioning scheme is able to transform the points from the original Cartesian coordinates to hyperspherical coordinates. After local processing of the partitions, the generated skyline candidates are merged together to generate global skylines. Although existing approaches address several challenges in skyline query processing, they suffer from two major drawbacks: (i) inability to overcome the *Data skew* and (ii) inability to handle *Data stragglers* introduced by skyline query processing in distributed environments, when the dimensionality of the input data points is high.

Data skew arises when some workers process more



(a) Skyline Example (b) Z-value

Fig. 1: Skyline Example and Z-order Curve

- M. Tang and Y. Yu is with the Department of Computer Science, Purdue University, IN 47906.
  E-mail: tang49@purdue.edu
- W.G. Aref is with the Department of Computer Science and Center for Education and Research in Information Assurance and Security (CERIAS), Purdue University.
- Q. Malluhi is with KINDI Center for Computing Research, Qatar University, Doha, Qatar.
- M. Ouzzani is with the Qatar Computing Research Institute, HBKU, Doha, Qatar.

data points than others. Therefore, we need a new approach that would evenly distribute input data over the workers; i.e., each worker receives a fair share of data points in the order of $|P|/M$, where $|P|$ is the size of the dataset, and M is the number of data partitions. *Data stragglers* refers to the situation where some workers spend significantly longer time than others to execute their tasks, because of reasons such as faulty disk, server failure, and the bad runtime performance of local processing algorithms [13], [14]. For example, the runtime performance of centralized skyline computation approaches (e.g., [2], [3], [4], [5]) degrades greatly with the increase of skyline points. Thus, if one worker is allocated more skyline points, it will experience a significant increase in its execution time. We thus need a new approach that would guarantee that each worker receives a fair share of skyline points. In addition, the size of intermediate skyline candidates, which are generated from the second stage, is another major factor impacting performance. For example, some workers may output large amounts of skyline candidates, which leads to large communication and I/O overhead to move and store intermediate data points. However, in most cases, only a small portion of these computed skyline candidates will contribute to the final skyline result set, since many intermediate points generated from one worker could be dominated by skyline points from other workers. Therefore, we also need to minimize the redundant skyline candidates.

Last but not the least, an efficient algorithm is crucially important to merge skyline candidates, since the size of skyline candidates may still be large, leading to poor performance of traditional skyline algorithms. For example, state-of-the-art skyline algorithms [2], [5] are sensitive to the number of skyline candidates. Therefore, without a specific optimized skyline candidate merging algorithm, the final stage of merging skyline candidates could be a bottleneck. Note that if the number of skyline points is huge, users could rank the computed skyline sets based on user defined functions such as in [15]. However, how to rank the skyline points based on a user's preference is not our focus in this work.

The main contributions of this paper are as follows:

1) We introduce three different approaches to partition input datasets for parallel computation of skyline candidates: (i) Use a simple Z-order curve from the literature. (ii) Based on skyline distribution among partitions, split partitions into separate groups to overcome the data straggler issue for distributed skyline query processing. (iii) Group several partitions into one group based on the skyline dominance volume, which would guarantee that each worker receives an equal amount of input data and skyline points, and would prune redundant intermediate skyline candidates.
2) We propose an efficient algorithm to merge skyline candidates by employing a data index for searching skyline sets, which enhances the query processing time by reducing redundant dominance testing.
3) We realize the proposed approach using a Hadoop MapReduce platform. We conduct large-scale evaluation on well-known benchmarks and compare the newly developed approach against other state-of-art algorithms. Our experiments demonstrate that the proposed algorithms can achieve an order of magnitude speedup over state-of-the-art approaches.

The rest of the paper proceeds as follows. Section 2 discusses related work. Section 3 presents the preliminaries for skyline query and Z-order curves. It also presents an overview of distributed skyline query processing. Section 4 investigates the property of Z-order curves and a data partitioning approach based on these curves. Section 5 gives implementation details about the Z-order curve data partitioning over the MapReduce platform, and introduces a tree-based approach for merging skyline candidates. Section 6 reports the experimental results and Section 7 concludes this paper.

2 Related Work

The skyline database operator was first formalized by Borzsony *et al.* [1] and was widely studied for building users' personalized queries over multi-dimensional datasets (e.g., Web and biological data) [16]. Several sequential skyline algorithms [2], [3], [4], [5] have been been studied. The Z-search algorithm proposed by Ken *et al.* [5] is the state-of-the-art skyline computation algorithm. In our work, we use an algorithm to merge skyline candidates that utilizes a Z-btree index similar to that of the Z-search algorithm. We further enhance performance by introducing new data partitioning and grouping techniques as explained in Section 4.

Distributed skyline computation for big data has received more attention recently. Hose and Vlachou [17] provide a good survey of distributed skyline algorithms. Different from traditional more tightly-coupled parallel platforms, share-nothing platforms (e.g., the Hadoop MapReduce platform) are becoming more attractive because of their simplicity. These platforms require partitioning and distributing the input data over multiple compute nodes. In the random partitioning scheme [18], local data chunks share similar distribution and structure as the original data. A grid-based partitioning scheme [9], [11] recursively divides some dimensions of data into two (or more) parts. The skylines are computed for each partition and local results are merged into the global skylines. However, grid-based partitioning approaches suffer from the unbalanced load distribution over the nodes. To overcome this, the angle-based partitioning scheme [8], [19] has been proposed. This scheme takes into account the fact that points are usually located around the origin. Thus, skyline points can be distributed in a balanced way by transforming the points from the Cartesian coordinates into the

TABLE 1: Definitions of symbols

Symbol	Definition
$\mathbb{R}^d$	d-dimensional vector space
n, $\|P\|$	Number of points in dataset P
$\check{P}$	A sample dataset from dataset P
S	A set of skyline points in dataset P
$\check{S}$	Skyline points from sample dataset $\check{P}$
$\hat{n}$	Skyline candidates
$p.d_j$	j-th dimensional value of point p
$\theta(p_i)$	Z-address for point p_i
$p \vdash q$	Point p dominates point q
$p \perp q$	Point p is incomparable with point q
M	Number of data partitions
Pt_i	A data partition for dataset P
$\hat{Pt_i}$	RZ-region for data partition Pt_i
G_m	A group of several data partitions
$\|Pts_i\|$	Number of skyline points in Partition Pt_i

hyperspherical coordinates. The projection-based partitioning scheme [7] adopts a similar idea to the angle-based approach but projects data onto a hyperplane. The hyperplane based approach maps data onto the hyperplane, based on the assumption that data transformation is affordable when the data size is small. In our experiments, we apply an idea similar to the projection-based approach to normalize data values for each point, and demonstrate that our proposed approach is more than one order of magnitude faster than angle-based and projection-based approaches.

Köhler *et al.* [7] propose a bottom-up merging technique to compute the final skyline. However, this method is not applicable in a Hadoop framework since each round of merging needs to write the intermediate data to HDFS and restart a new MapReduce job. Park *et al.* [20] build a Quadtree for sampling data, and finding the dominance relationships among different partitions. However, a Quadtree index usually fail to partition the high-dimensional input data in a balanced way [21]. More recently, Liu *et al.* [12] used the bit-string to represent grid-based partitioning, which enables pruning more data points before final skyline computation. We compared the proposed approach with the bit-string based approach and we achieved ten times speedup.

The approach developed in this work is different from the previous approaches because of the following three reasons: (1) the Z-order curve based data partitioning maps high–dimensional data into a low-dimensional data, which facilitates dividing the high-dimensional data space evenly; (2) the novel dominance-based data partition grouping enables efficient pruning of skyline candidates; and (3) the introduction of a new index-based approach for merging skyline candidates.

3 Preliminaries and Problem Statement

In this section, we present different concepts that are needed in the rest of the paper.

3.1 Skyline Computation

Given a d-dimensional space $\mathcal{D} \subseteq \mathbb{R}^d$ and a set of data points $P \subseteq \mathcal{D}$, we use $p.d_j$ to denote the value of Point p in the j-th dimension. Table 1 summarizes the symbols used in this paper.

Definition 1: **Dominance**. Point p dominates point q if $p.d_i < q.d_i$ for at least one dimension i, and $p.d_j \leq q.d_j$ for all the other dimensions, $j \neq i$. This is expressed as $p \vdash q$. $p \nvdash q$ means p does not dominate q. If $p \nvdash q$ and $q \nvdash p$, we say point p is incomparable with point q, denoted as $p \perp q$.

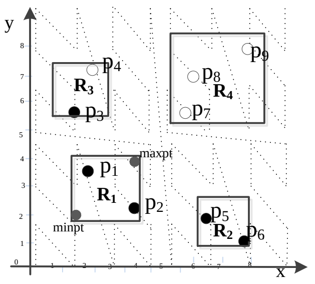
Definition 2: **Skyline**. The skyline of P is a set of points S, $S \subseteq P$, such that points in S are not dominated by any other points in P.

The dominance transitivity and non-comparability properties were presented in [1], [5]. According to the transitivity property, if a point p dominates all the points in P, and another point $\hat{p}$ dominates p, then $\hat{p}$ dominates all the points in P. The transitivity property guarantees the correctness of pruning in parallel skyline computation. After each worker computes skyline candidates from the horizontally partitioned dataset, the merge of all these candidates is guaranteed to produce the correct skyline since all the points in this results will dominate any point in any partition.

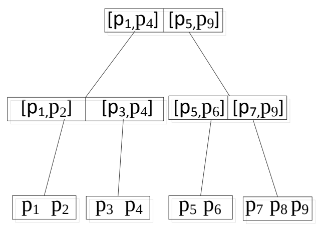
3.2 Z-order Curve for Skyline Computation

A Z-order space filling curve [22] maps a data point from a high-dimensional space to a one dimensional space, where each point is represented by a unique number, called Z-address. A Z-address is a binary string calculated by interleaving the bits of all coordinates of a data point. For example, Point $p = (3, 5)$ with binary values of $(011, 101)$, has the related Z-address of "011011". The first two bits "01" of the Z-address are obtained from the first bits of 011" and 101", Similarly, the rest of the Z-address can be computed accordingly by interleaving the bits of the remaining coordinates. The Z-search algorithm [5] explores clustering and monotonic ordering properties of a Z-order curve, and develops a data structure called ZB-tree for skyline computation. Generally speaking, the Z-search method is a hybrid algorithm which uses a ZB-tree index and RZ-region dominance test. Below we introduce the concept of an RZ-region as defined by [5].

Definition 3: An RZ-region R is the smallest square that covers a region bounded by extreme points' Z-addresses $[\alpha, \beta]$, and R is the smallest square area covered by two Z-addresses $minpt(R)$ and $maxpt(R)$.
Notice, $[\alpha, \beta] \subseteq [minpt(R), maxpt(R)]$. By selecting the common prefix of α and β, $minpt(R)$ is computed by setting all 0's to the rest of bits other than the common prefix, and $maxpt(R)$ is computed by setting all 1's to the rest of bits other than the common prefix. Therefore, all the points in a RZ-region R are dominated by the data point at $minpt(R)$.



(a) RZ-region



(b) ZB-tree

Fig. 2: RZ-region and ZB-tree

For example, suppose an RZ-region R covers 3 different points, whose Z-addresses are "10110", "10011", and "10010". The common prefix is "10" and the boundary values are α = "10010" and β = "10110" after sorting the Z-addresses. Therefore, $minpt(R)$ = "10000" and $maxpt(R)$ = "10111". The rectangle $R1[p1, p2]$ is the RZ-region for Points p_1 and p_2 in Figure 2(a). A naïve way to compute dominance relationships between regions is by performing a dominance test between each pair of data points. However, the computation cost is $O(n^2)$, which is prohibitive for big data. Based on the RZ-region, the dominance test among data pairs can be performed by testing the dominance relationships among the RZ-regions as follows.

Lemma 1: Given two RZ-regions R_i and R_j, there are three possible dominance relationships:

1) if $maxpt(R_i) \vdash minpt(R_j)$, then $R_i \vdash R_j$;
2) if $minpt(R_i) \nvdash maxpt(R_j)$ and $minpt(R_j) \nvdash maxpt(R_i)$, then $R_i \perp R_j$;
3) if $maxpt(R_i) \nvdash minpt(R_j)$, but $minpt(R_i) \vdash maxpt(R_j)$, then R_i dominates a part of Region R_j, we denote this case as $R_i \Vdash R_j$.

***Example** 1:* Consider Figure 2 (a), $maxpt(R_1)$ dominates $minpt(R_4)$. Thus, RZ-region R_1 dominates RZ-region R_4. It is obvious that RZ-region R_2 is incomparable to RZ-region R_3. In this way, the all-pairs dominance test can be avoided between these two RZ-regions.

Based on the RZ-regions, each point is mapped to its Z-address, and a ZB-tree is built for the mapped data in a bottom-up fashion. A ZB-tree is a balanced tree, whose leaf nodes store the data points and internal nodes store the RZ-region of the children nodes, e.g., Figure 2 (b). More details about searching skyline points from a ZB-tree can be found in the work of Lee *et al.* [5]. For skyline computation on big datasets, extending the sequential Z-search algorithm to a parallel counterpart is nontrivial, and raises several challenges as described below.

3.3 Challenges of Parallel Skyline Query Processing

Unbalanced Partitioning. An ideal data partitioning requires each worker to be assigned an equal amount of input data, i.e., $|P|/M$. However, most data partitioning approaches, such as grid-based partition [9], [11], angle-based [8], and quad-tree-based [20], fail to guarantee such balanced assignment when the input data dimensionality is high, e.g., dimension > 5.

Straggling Workers. There are cases where certain workers might run into straggler issues. For example, the data points allocated to one worker may contain a small percentage of skyline points. This worker may generate skyline candidates that would be dominated by skyline points from other workers at the end. Thus, this worker would induce unnecessary disk and network I/O cost to store intermediate results. In addition, one worker might be allocated data points containing more skyline points than others, which would degrade the runtime performance of local skyline algorithms as well. The runtime performance of the state-of-art skyline computation algorithm (i.e., Z-search [5]) is bound by the number of skyline points, that is, its computation complexity is $O(d|S|\log_d n)$, where d is the dimension of the dataset, $|S|$ is the number of skyline points, and n is the number of input data points. Therefore, even if each worker receive an equal amount of input data points, some workers may run into the data straggler issue and become the bottleneck of skyline query processing.

Large Candidate Set. Local skyline computation on each worker may generate a large amount of skyline candidates. For example, in our experiments we were able to compute 2 million skyline candidates from a 12 million input dataset (anti-correlated distribution). It is not feasible to execute the dominance test for all-pairs of skyline candidates.

3.4 Problem Statement

Given an input dataset P, we need to find a good partitioning for P, and an efficient algorithm to merge the skyline candidates from the different partitions. Intuitively, to overcome the data skew and stragglers challenges, data points should be mapped from a high-dimensional space to a low-dimensional space. Therefore, we introduce the space-filling curve (in this case, z-order curve) to achieve this goal and then utilize the index over the mapped data for skyline candidates merging. The role of this tree index is to prune unnecessary data dominance testing by executing the dominant test over the index rather than over skyline candidates, which makes the operation space- and time-efficient.

4 Z-order Curve Based Partitioning

In this section, we first present a new z-order curve based data partitioning strategy to overcome data skew and tackle the unbalance challenges. Then, we develop the data partition grouping approach to reduce the data stragglers. Finally, we introduce the dominance-based partition grouping approach to minimize the size of intermediate skyline candidates, while overcoming the data skew and data stragglers issues.

4.1 Data Skew Reduction

To handle data skew, it is important that data is equally partitioned and assigned to each worker. Hence, we propose the following procedure. Given an input dataset P containing $|P|$ data points, we divide it into multiple partitions $Pt_m, 1 \leq m \leq M$, s.t. each partition has the number of points $|Pt_m|$. Naturally, $|P| = \sum_{m=1}^{M} |Pt_m|$. To balance data between partitions, we need to minimize the variance of the data distribution in each Pt_m, that is, the function $\sum_{m=1}^{M}(|Pt_m| - |P|/M)^2$ needs to be minimized.

***Example** 2:* We study skyline distributions in space for two real datasets, e.g., *NBA* and *HOU*, which follow anti-correlated and independent distributions respectively[1]. The *NBA* dataset contains the latest top 350 players' statistics in the 2013-2014 season (2 months). Each record corresponds to an NBA player's performance in 7 aspects, such as scores, rebounds, steal, etc. *HOU* consists of 1k 6-dimensional data points, each record representing the percentage of an American family's annual expense on 6 types of expenditures, such as electricity, gas, and so on. Given the input dataset, we first compute the skyline set, and then analyze its distribution in the space.

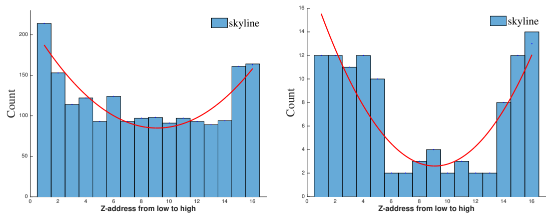
A Z-order curve maps the computed skyline points from a high dimensional space to a one dimensional Z-address space, and skyline points are ordered in the Z-address space similar to Figure 1 (b). Figure 3 shows the histograms of skylines along the Z-order curves. We make the following observations: (i) skyline points for high dimensional data are not only in the low and high buckets (with smaller and higher Z-addresses, respectively), (ii) skyline points are not clustered around the origin in a high dimensional space as in a low dimensional space, and (iii) the skyline points tend to distribute in a more balanced fashion along the buckets of the z-order curve data space.

Motivated by skyline distribution along the Z-address space, we utilize the Z-order curve to map a high-dimensional data point (say p) to one dimensional Z-address, say $\theta(p)$, based on the skyline point distribution along the mapped space , say $\theta(p)$. In the one dimensional space, the input dataset P is partitioned evenly such that each partition contains the same amount of data, i.e., $|P|/M$. Therefore, we can guarantee the input data points distribution in each partition is even based on the following.

Z-order Curve-based Partitioning. This process selects $(M-1)$ points from dataset P as pivots, and split the points of P into M disjoint partitions, where each point is assigned to the partition with its closest pivot by the corresponding Z-address.

Let $p_1, p_2, \ldots, p_{M-1}$ be the $(M-1)$ selected pivot points. Without loss of generality, the Z-addresses of the pivot points can be sorted, i.e., $\theta(p_1) < \theta(p_2) < \ldots < \theta(p_{M-1})$. A data point $p \in Pt_m$ if $\theta(p) \in [\theta(p_{m-1}), \theta(p_m))$.

1. Those datasets are collected from www.nba.com, www.ipums.org respectively.



(a) HOU dataset (b) NBA dataset

Fig. 3: Histograms and quadratic fitting curves for Skyline points along the Z-order curve in two real-world datasets

Note that these pivot are learned via the following steps, (1) computing samples for input data points; (2) sorting samples via their z-order address; (3) calculating the pivots s.t. each partition has an equal amount of samples in the ordered data space. Overall, we split the input data points into partitions with balanced inputs based on the selected pivots.

4.2 Reducing Data Stragglers

The input dataset is partitioned according to the Z-order curve-based partitioning described in Section 4.1. For example, we can partition the input dataset in Figure 1b) into four parts along the mapped Z-address, such that each partition has approximately the same amount of input. However, we observe that Partitions II and IV have very few skyline points (one dark dotted point), while the other two partitions, Partitions I and III, contribute most of the skyline points. The number of skyline points in a partition affects the local computation cost for the corresponding worker. Therefore, as introduced in Section 3.3, this uneven skyline distribution raises the issue of data stragglers through impeding runtime performance for some workers. A natural idea is to divide partitions into disjoint groups based on the skyline distribution among partitions. Formally,

Definition 4: Let dataset $P = \cup_{1 \leq m \leq \hat{M}} G_m$, where G_m is a group consisting of a set of partitions of dataset P, $G_i \bigcap G_j = \emptyset$ for $i \neq j$ and $\hat{M}$ is the number of groups. Each group G_m consists of several partitions, i.e., $G_m = \cup_{1 \leq i \leq m'} Pt_i$, where Pt_i is a partition, and m' is the number of partitions in group G_m.

We observe that if a partition Pt_i contains much more skyline points than a partition Pt_j, then it is more likely that Pt_i contains skyline points that dominate data points in partition Pt_j. For example, in Figure 4(a), partition Pt_1 has more skyline points than partition Pt_{16}. Intuitively, we can group data points belonging to partitions Pt_1 and Pt_{16}, into the same group. Then, for data partitions in the same group (in this case partitions Pt_1 and Pt_{16}), the non-skyline points contained in Partition Pt_{16} will be pruned with high probability. Therefore, we could use this heuristic-based partitioning to cluster different partitions into separate groups such that each group contains similar amount of skyline points, as

Algorithm 1 Heuristic Partition-Grouping

```
Input: $\check{P}$: sample data, $\hat{M}$: number of groups, $\delta$: partition expansion factor
Output: $PGmap$: hashmap from partition ID to group ID
1: $Pt \leftarrow$ PZORDER($\check{P}, \hat{M} * \delta$)
2: $S_s \leftarrow$ computeSkyline($\check{P}$);
3: $Pt \leftarrow$ redistribute($Pt, \check{S}$);
4: $Pt \leftarrow$ sort($Pt$); //sort partitions via skyline points distribution
5: $m \leftarrow 0$;
6: $tcons \leftarrow |P|/\hat{M}$; //constraint 1
7: $scons \leftarrow |S_s|/\hat{M}$ ; //constraint 2
8: $hptr \leftarrow$ head($Pt$);
9: $tptr \leftarrow$ tail($Pt$);
10: initialize $PGmap$, $tcount, scount$ by $hptr$'s content
11: while ($hptr! = tptr$) do
12:     $scount \leftarrow scount + tptr.sc$; //update skyline count
13:     $tcount \leftarrow tcount + tptr.tc$; //update points count
14:     PGmap.put($tptr.pid, m$);
15:     $tptr \leftarrow tptr.previous$;
16:     if ($scount > scons$ or $tcount > tcons$ ) then
17:         $m \leftarrow m + 1$;
18:         $hptr \leftarrow hptr.next$;
19:         initialize $G_m$, $scount, tcount$ by $hptr$ info
20:     end if
21: end while
22: return $PGmap$
```

well as equal amount of data points. The following proposition describes this formally.

Proposition 1: Given a partition set Pt and a group set G, let $|Pts_i|$ be the number of skyline points belonging to partition Pt_i, the necessary condition that a partition Pt_i is joined into group G_m is that (1) the variance of the skyline distribution among groups, $\sum_{m=1}^{\hat{M}}(|Gs_m| - |S|/\hat{M})^2$, is minimized, where $|Gs_m| = \sum_{i=1}^{m'} |Pts_i|$, $\hat{M}$ and $|S|$ are the number of groups and the number of skyline points, respectively, and (2) the variance of the number of points among groups, $\sum_{m=1}^{\hat{M}}(|G_m| - |P|/\hat{M})^2$, is also minimized, where $|G_m| = \sum_{i=1}^{m'} |Pt_i|$.

Heuristic Partition-Grouping. Algorithm 1 shows the pseudo-code of the heuristic for merging partitions into different groups, which proceeds as follows. First, to approximately compute the skyline distribution, a sample dataset is collected from dataset P using reservoir sampling [23]. The sample dataset $\check{P}$ is divided into several partitions (i.e., $\hat{M} * \delta$, such that each partition has the same number of input data points. Note that δ is the partition expansion factor and is bigger than one) in order to get more partitions to merge into groups. The corresponding skyline for the sample data $\check{P}$ is $\check{S}$.

Next, Procedure `computeSkyline` (Line 2) computes the skyline set from the sample data $\check{P}$, and the number of skyline points in each partition is computed. As mentioned earlier, the goal is to group different partitions together such that each group contains approximately an equal amount of skyline points. Thus, partitions with more skyline points (i.e., bigger than $|\check{S}|/|\hat{M}|$), are further divided into separate partitions via Procedure `redistribute` (Line 3). Note that Procedure `redistribute` is called such that the greedy merging procedure can work. For example, if the size of the skyline for Partition Pt_i is bigger than $|\check{S}|/|\hat{M}|$, then Partition Pt_1 in Figure 4(a) is divided into Partition Pt_1' and Pt_1'' in Figure 4(b). In our implementation, Procedure `redistribute` re-partitions one partition based on the sampled skyline distribution in the space, that is, one partition is further divided into partitions such that each partition has a similar amount of sample skyline points.

Then, these further split partitions are sorted decreasingly based on the number of skyline points, i.e., $|Pts_i|$ (Line 4). To approximately minimize the variance (Proposition 1), the upper bounds of the number of skyline points and data points, say $scons$ and $tcons$, are set for each group (Lines 6-7), respectively. Finally, all partitions are scanned until each one is properly assigned to a certain Group G_m, i.e., if either (a) the number of skyline point or (b) the number of data points in Group G_m is beyond the upper bound. If any constraint is not satisfied, a new group is generated (Line 17).

Discussion One issue with the proposed Heuristic Partition-Grouping is that it fails to avoid redundant skyline computation in certain cases. For example, in Figure 4 (c), this heuristic merges Partitions Pt_2 and Pt_3 together. However, after this grouping, the dominance testing for data points in these two partitions is wasted, because data points in the corresponding data partition have no dominance relationship. Furthermore, skyline distribution from the sample dataset is only an approximation of its distribution over the whole dataset. Heuristic Grouping tries to equally distribute sample skyline points in each group. However, the sample may not perfectly reflect the actual distribution and some groups may receive much more skyline points than others. In the next section, we propose several techniques to mitigate these issues.

4.3 Dominance-based Partition Grouping

Dominance-based Grouping is developed via the dominance relationships between various partitions. Given partition Pt_i with two pivots, say $[\theta(Pv_m), \theta(Pv_{m+1}))$, we first compute the $maxpt(Pt_i)$ and $minpt(Pt_i)$ of partition Pt_i from $\theta(Pv_m)$ and $\theta(Pv_{m+1})$, as illustrated in Definition 3. Let $\hat{Pt_i}$ be the RZ-region of partition Pt_i. The pivot values of partition Pt_i belong to the interval between the min point and the max point of the partition's RZ-region, i.e., $[\theta(Pv_m), \theta(Pv_{m+1})) \subseteq [maxpt(Pt_i), minpt(Pt_i))$. Then, we can obtain RZ-region's dominance relationships from Lemma 1. Partition Pt_i can be pruned if $\hat{Pt_i}$ is dominated by another partition's RZ-region $\hat{Pt_j}$. For example, partition Pt_5 in Figure 4(c) can be pruned, since $\hat{Pt_1} \vdash \hat{Pt_5}$.

In addition, data partitions can be grouped according to case 3 of Lemma 1, e.g., some points of Partition Pt_j can be dominated by points in partition Pt_i. Suppose

Algorithm 2 Dominance-based Partition-Grouping

```
Input: $\check{P}$: sample data, $\hat{M}$: number of groups, $\delta$: partition
    expanding factor
Output: $PGmap$:hashmap between Partition ID and Group ID
 1: $Pt \leftarrow$ pZORDER($\check{P}, \hat{M} * \delta$)
 2: $S_s \leftarrow$ computeSkyline($\check{P}$);
 3: $DM \leftarrow$ dominate($Pt$); //dominate matrix
 4: $Pt \leftarrow$ redistribute($S_s$,$Pt$);
 5: $m \leftarrow 0$;
 6: $tcons \leftarrow |P|/\hat{M}$ ; // constraint 1
 7: $scons \leftarrow |S_s|/\hat{M}$ ; // constraint 2
 8: sort($Pt$, $DM$);
 9: Initialize group $G_0$ by the first partition of $Pt$
10: while ($Pt$ is not empty) do
11:     if ($tcount > tcons$ or $scount > scons$) then
12:         $m \leftarrow m + 1$;
13:         $hptr \leftarrow hptr.next$ ;
14:         initialize $tcount$, $scount$
15:     end if
16:     $Pt'_i \leftarrow$ MaxDominate($G_m$, $Pt$, $DM$);
17:     remove $Pt'_i$ from $Pt$;
18:     put $Pt'_i$ into $G_m$;
19:     $scount \leftarrow scount + |Pts'_i|$;
20:     $tcount \leftarrow tcount + |Pt'_i|$
21:     PGmap.put($Pt'_i.pid$,$m$);
22: end while
23: return $PGmap$
```

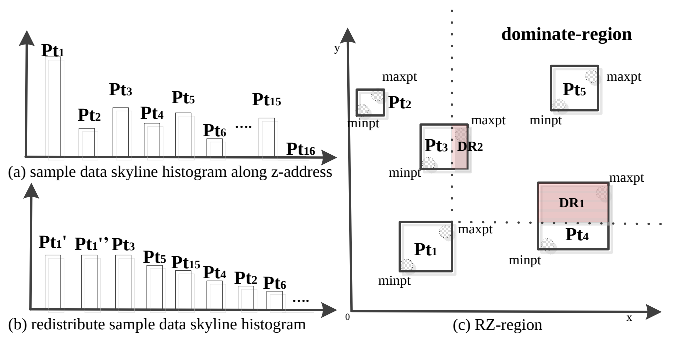


Fig. 4: Sample data skyline histogram and dominance volume

that the number of data points in $\hat{Pt}_j$ can be estimated, then partition Pt_i and Pt_j can be grouped together, if the estimated data to be pruned is large. This results in a high possibility of reducing redundant skyline computation.

***Example** 3:* Consider the three partitions Pt_1, Pt_3 and Pt_4 in Figure 4(c), we want to put these partitions into two groups (say, G_1 and G_2). Note that partition Pt_1 dominates regions of partitions Pt_3 and Pt_4, and these two regions are illustrated in red-shaded color, namely DR_1 and DR_2. The area of DR_1 is bigger than that of DR_2 indicating that more points in partition Pt_4 are likely to be dominated by points in partition Pt_1. Thus, it is more reasonable to merge partitions Pt_1 and Pt_4 into group G_1, rather than grouping partitions Pt_1 and Pt_3 together.

Motivated by the above observation, the *dominance volume* is defined as the RZ-region area of partition Pt_j, which is partially dominated by the max point of partition Pt_i's RZ-region.

Given the RZ-regions of two partitions $\hat{Pt}_i$ and $\hat{Pt}_j$ s.t. $\hat{Pt}_i \Vdash \hat{Pt}_j$ and $i \neq j$. Four Z-address values are computed, namely $maxpt(\hat{Pt}_i)$, $minpt(\hat{Pt}_i)$, $maxpt(\hat{Pt}_j)$, and $minpt(\hat{Pt}_j)$. These four Z-address values can be transformed back to the original k-dimensional points. For each dimension $k \in \{1, 2, \ldots, d\}$, a set X_k is used to hold the point coordinates, $X_k = \{minpt_i[k], maxpt_i[k], minpt_j[k], maxpt_j[k]\}$, where $minpt_i[k]$ denotes the value of $minpt(\hat{Pt}_i)$ on the k-th dimension. Let X_k^ℓ denote the largest element of X_k and X_k^s denote the second largest element of X_k.

Definition 5: The *dominance volume* is defined as

$$V_{\text{dom}}(\hat{Pt}_i, \hat{Pt}_j) = \int_{X_1^s}^{X_1^\ell} \cdots \int_{X_d^s}^{X_d^\ell} \mathrm{d}t_1 \cdots \mathrm{d}t_d.$$

According to the definition, the dominance volume is commutative, i.e., $V_{\text{dom}}(\hat{Pt}_i, \hat{Pt}_j) = V_{\text{dom}}(\hat{Pt}_j, \hat{Pt}_i)$ and $V_{\text{dom}}(\hat{Pt}_i, \hat{Pt}_i) = 0$.

Lemma 2: If $V_{\text{dom}}(\hat{Pt}_i, \hat{Pt}_j) > V_{\text{dom}}(\hat{Pt}_i, \hat{Pt}_k)$, then the possibility that the points in partition Pt_j(or Pt_i) can be dominated by partition Pt_i(or Pt_j) is higher than the possibility that the points in partition Pt_k(or Pt_i) can be dominated by possibility Pt_i(or Pt_k).

Proof: This can be proved by the definition of dominance volume. Refer to Figure 4(c) and Example 3, $V_{\text{dom}}(Pt_1, Pt_3)$ and $V_{\text{dom}}(Pt_1, Pt_4)$ is the area for region DR_1 and DR_2, respectively. □

Based on the dominance volume for partitions, the optimal partition-grouping approach needs to merge partitions such that partitions in the same group have larger dominance volume between each other than any other partition outside the group. At the same time, similar to the grouping heuristic strategy, for a group, say G_m, the corresponding partition size cannot exceed the average partition size, i.e., $tcons = |P|/\hat{M}$ and the number of skyline points contained in this group cannot exceed the average skyline size, i.e., $scons = |S|/\hat{M}$. Therefore, the objective function for dominance based partition-grouping is formalized as follows,

$$\begin{aligned} \text{maximize} \quad & \sum_{m=1}^{\hat{M}} \sum_{Pt_i, Pt_j \in G_m} V_{\text{dom}}(\hat{Pt}_i, \hat{Pt}_j) \\ \text{subject to} \quad & G_m \cap G_n = \emptyset, \bigcup_{k=1}^{\hat{M}} G_k = P, m \neq n, \\ & \sum_{Pt_i \in G_m} |Pt_i| \leq |P|/\hat{M}. \\ & \sum_{Pt_i \in G_m} |Pts_i| \leq |S|/\hat{M}, \end{aligned}$$

The difficulty of the above optimization problem is that the number of skyline points $|S|$ cannot be accurately estimated. We use the sample skyline size $|\check{S}|$ to approximate the true skyline size $|S|$. In order to obtain an approximately optimal grouping for maximizing the above objective function, we adopt a greedy approach. First, we define *dominance matrix* and *dominance power*.

Definition 6: Given a partition set Pt with M partitions, the *dominance matrix*, denoted as DM, is defined as follows:

$$DM[i][j] = V_{\text{dom}}(\hat{Pt_i}, \hat{Pt_j}), \text{ and } i, j \in \{1, ..., M\}.$$

Definition 7: For a partition Pt_i, its *dominance power* $\Gamma(Pt_i)$, is the sum of all the dominance volumes between Pt_i and other partitions, i.e.,

$$\Gamma(Pt_i) = \sum_{Pt_j \in Pt, j \neq i} V_{\text{dom}}(\hat{Pt_i}, \hat{Pt_j}) = \sum_{j=1}^{M} DM[i][j].$$

Figure4(c) shows that $DM[1][] = [0, 0, S_a, S_b, S_c]$, where S_a and S_b are the areas of DR_2 and DR_1 correspondingly, and S_c is the area of RZ-region of Pt_5. Partition Pt_1 has a larger dominance power than other partitions. We adopt a greedy search strategy to maximize the objective function based on the following procedure: (1) remove the partition with the largest dominance power (say, Pt_i) from Pt, and place Pt_i in a group (say, G_m), (2) choose another partition whose dominance volume with G_m is maximal, i.e.,

$$Pt'_i = \underset{Pt_j \in Pt, Pt_j \notin G_m}{\text{argmax}} \sum_{Pt_i \in G_m} V_{\text{dom}}(\hat{Pt_i}, \hat{Pt_j}).$$

(3) repeat step (1) and (2) until the constraint for group G_m is satisfied. The procedure continues for the next group.

Algorithm 2 shows the details of the Dominance Grouping procedure. Before merging different partitions, partition preprocessing steps are introduced below. The sample data partition is computed as in Algorithm 1. procedure `dominate()` (Line 3) computes the dominance relationships among partitions and builds the dominance table DM. The running time complexity is $O((|\hat{M}|\delta)^2)$, which is affordable since $\hat{M}\delta = |Pt|$ is usually small (about 1k). Next, procedure `redistribute()` (Line 4) removes the dominated partitions, and also splits the partitions that have more skyline (i.e., $|Pts_i| > |\check{S}|/\hat{M}$). Next, procedure `sort()` sorts the partitions in descending order according to their dominance power and the number of skyline points (i.e., $|Pts_i| * \Gamma(Pt_i)$) (Line 7). Then procedure `maxDominate()` finds the partition (i.e., Pt'_i) whose dominance volume $Pt_i \in G_m$ is maximal. Then, partition Pt'_i is removed from the partition set Pt and put into current group G_m (Line 19-21). Meanwhile, group G_m is updated (Lines 20-22). Algorithm 2 is repeated until group G_m is beyond its capacity. A new group G'_m is created. After each partition is assigned to one group, Algorithm 2 stops and returns the mapping rule between partitions and groups $PGmap$.

***Example** 4:* We use data partitions in Figure 4 to illustrate Algorithm 2. To merge partitions, the algorithm initially chooses partition Pt_1 and assigns it to an empty group G_0. Then partition Pt_1 is removed from the data partition set and put into group G_0. Next, procedure `maxDominate()` chooses partition Pt_4 and adds Pt_4 to G_0, because dominance volume $V_{\text{dom}}(\hat{Pt_1}, \hat{Pt_4})$ is the largest. After Pt_4 is added to group G_0, G_0 no long satisfies the constraint that the group size is smaller than $|P|/\hat{M}$. So a new group G_1 is created and Pt_2 is assigned to G_1. This procedure continues until each partition is properly assigned.

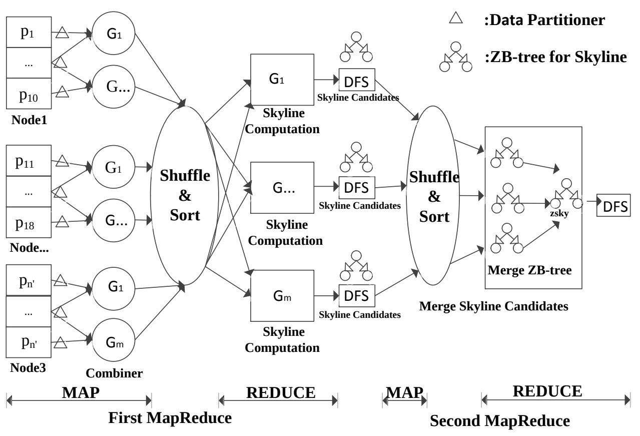


Fig. 5: An Overview of Skyline Query in MapReduce

5 PARALLEL SKYLINE QUERY PROCESSING

In this section, we show how we implemented our proposed skyline query processing using MapReduce (Figure 5).

Algorithm 3 1st MapReduce Job

```
1:  function MAP($k1, v1$)
2:      SZB-tree ←buildZBtree($\check{S}$);
3:      if not Dominate(v1, SZB-tree) then
4:          $zaddress$ ← zaddress($v1$);
5:          $pid$ ← searchPT($zaddress, Pv$); //binary search Partition ID
6:          $gid$ ← $PGmap.get(pid)$; // get group ID
7:          if m is not NULL then
8:              output($gid$, v1);
9:          end if
10:     end if
11: end function
12:
13: function REDUCER($k2, v2$)
14:     ZB-tree ← compute skyline via Z-search
15:     output(ZB-tree);
16: end function
```

5.1 Preprocessing for Data Partitioning

As introduced in Section 4, the data preprocessing is invoked at the master node for selecting a set of sample data using reservoir sampling [23]. After the sampling step, we adopt the data partition-grouping strategies in Algorithm 1 and Algorithm 2 to learn a data partitioning policy. Finally, the data preprocessing step outputs the data partition-grouping rules.

5.2 Compute Skyline Candidates

Algorithm 3 shows the pseudo-code of the first MapReduce job. Specifically, before launching a map function, the partitioning pivot set Pv, sample data skyline $\check{S}$, and partition-grouping rule (e.g., $PGmap$), are loaded into the main memory of each mapper by using the distributed cache of Hadoop. Notice that a ZB-tree, say SZB-tree, is built for the sample data skyline $\check{S}$. Then the dominance testing between SZB-tree and input data points is carried out, which enhances the naïve all-pairs dominance testing to filter out non-skyline data points in the mapper. The qualifying tuples are mapped from a partition ID pid to a valid group ID gid, and the key-value pairs $(gid, v1)$ are output (Line 8). Between a mapper and a reducer, a combiner computes skylines on the mapper side by executing a skyline algorithm for data points in the same group, and outputs skyline candidates locally. In this way, the combiner filters out non-skyline tuples and reduces the shuffling cost. Next, reducers are launched after all mappers and combiners finish. Finally, skyline candidates are computed and output to the distributed file system.

5.3 Merge Skyline Candidates

The second MapReduce Job performs the skyline candidate merging. The main task of a mapper in the second MapReduce job is to shuffle the skyline candidates into a reducer, and the reducer performs the skyline merging based on the tree-based approach introduced below. Tree-based skyline computation is driven from the following observations. Skyline candidates are stored as a tree structure i.e., a ZB-tree. Merging indices of skyline candidates can reduce the pair-wise computation for incomparable skyline points. Thus, we propose the tree-based based skyline candidate merging algorithm *Z-merge*, which is described below.

Algorithm 4 shows the details on how to merge two ZB-trees, say `Zsrc` and `Zsky`, where `Zsrc` represents new coming data points, and `Zsky` corresponds to the existing skyline set. Algorithm `Z-merge` traverses ZB-tree `Zsrc` in breadth-first search order (short as BFS). BFS starts from the head of `queue`. In each step, a Node `n`, is retrieved from the head of `queue`.

Procedure `UDominate()` performs dominance testing between node `n` and ZB-tree `Zsky` (Line 6) as in [5]. According to the dominance relationship, if node `n` dominates node `n'` in ZB-tree `Zsky`, then node `n'` is removed from the ZB-tree `Zsky` by procedure `UDominate()`.

A non-leaf node of a ZB-tree is essentially an RZ-region R encoded by $minpt(R)$ and $maxpt(R)$ of the region. Next, if ZB-tree `Zsky` dominates bode `(n.minpt, n.maxpt)`, the children nodes of node `n` are discarded. If ZB-tree `Zsky` is incomparable with node `(n.minpt, n.maxpt)`, `Z-merge`, the algorithm inserts node `n` into `dominate-branches` (Lines 9-10), and merges the subtree of node `n` with `Zsky` (Lines 25-26). For the third case, if node `n` is not a leaf node, the children nodes of node `n` are inserted into `queue` (Lines 12-15). On the other hand, if node `n` is a leaf node, a dominance testing is carried out between the related value of node `n` and ZB-tree `Zsky` (Lines 16-20).

Algorithm 4 Z-merge

```
Input: Zsbrc: ZB-tree for the source data, Zsky: ZB-tree for the skyline
Output: Zsky: ZB-tree by merging the skyline of source data
1:  $queue \leftarrow empty$
2:  queue.enqueue($Zbsrc.root$);
3:  var dominate-branches;
4:  while (queue is not empty) do
5:      node $n \leftarrow queue.dequeue()$;
6:      $drl \leftarrow$ UDominate($Zsky, n.minpt, n.maxpt$))
7:      if Zsky dominate ($n.minpt, n.maxpt$) then //case 1
8:          continue;
9:      else if (Zsky incomparable ($n.minpt, n.maxpt$)) then
10:         dominate-branchs.add(n); //case 2
11:     else//case 3
12:         if (n is not leaf node) then
13:             for (each children $n'$ of node $n$) do
14:                 queue.enqueue($n'$); //searching on children
15:             end for
16:         else
17:             for (each children $p$ for node $n$) do
18:                 if (Zsky not dominate $p$) then
19:                     insert $p$ into Zsky //$p$ is a skyline point
20:                 end if
21:             end for
22:         end if
23:     end if
24: end while
25: for (each node $n'$ in dominate-branches) do
26:     append($n'$, Zsky)
27: end for
28: balanceZbtree(Zsky);
29: return Zsky
```

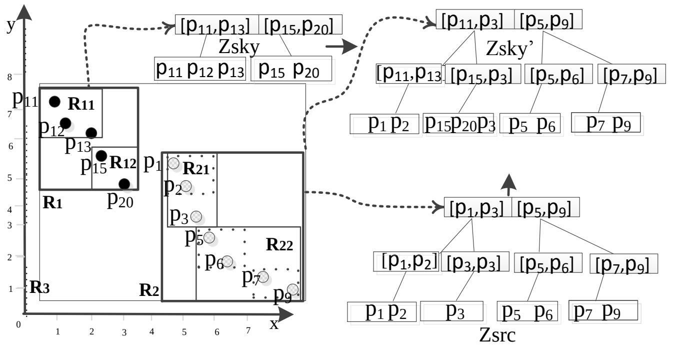


Fig. 6: Merge the ZB-tree of Existing Skyline Points with New Input Data

After traversing ZB-tree `Zsrc`, the sub-ZB-tree in `dominate-braches` is appended to ZB-tree `Zsky` (Lines 25-27), and ZB-tree `Zsky` is balanced again (Line 28).

***Example** 5:* Figure 6 shows an example of merging source data ZB-tree `Zsrc` with an existing skyline set ZB-tree `Zsky`. The root node of `Zsrc`, denoted as {[p_1, p_3] [p_5, p_9]}, represents the region R_2 in Figure 6. By evaluating the minimum point and maximum point of regions R_2 and R_1, node {[p_1, p_3] [p_5, p_9]} cannot be dominated by ZB-tree `Zsky`. Thus, we put those children nodes {[p_1, p_2] [p_3, p_3]}, { [p_5, p_6] [p_7, p_9]} of node {[p_1, p_3] [p_5, p_9]} into the queue. For the children node {[p_1, p_2] [p_3, p_3]} and its corresponding region R_{21}, we can observe that region R_{12} dominates part of the points in region R_{21}, similar to the third region dominate case as introduced in Algorithm 4. Thus, this invokes the dominate test among the leaf nodes, i.e., $\{p_1, p_2\}$, $\{[p_3, p_3]\}$ with region R_{12}. As a result, points p_1 and p_2 are removed from the skyline set, while point p_3 remains. On the other hand, node $\{[p_5, p_6]\ [p_7, p_9]\}$ is incomparable with `Zsky`. Finally, `Zsky′` is the final merged ZB-tree for the skyline points.

5.4 Discussion

Analysis of Data Pruning

We analyze the amount of data that could be pruned by the dominance-based partition-grouping approach. We focus on the number of data points to be pruned, because it directly impacts the whole query processing time, i.e., the more points are pruned, the less disk overhead and network communication are incurred to store redundant non-skyline data points, and the less computation time it takes to merge skyline candidates, the shorter skyline query processing time is needed. Notice that the number of partitions M mentioned below is the number of groups, which is referred to as $\hat{M}$ in the previous section. We define the *total dominance volume* in order to show the number of points that can be pruned during the computation.

The *total dominance volume* is defined as

$$V_\text{t} = \frac{1}{2}\sum_{i}^{M}\sum_{j}^{M} V_\text{dom}(\hat{Pt}_i, \hat{Pt}_j).$$

Thus, the total number of pruned point is $N_\text{t} = \bar{\rho}V_\text{t}$, where $\bar{\rho}$ is the average density of points in the space. To estimate the cost of skyline query processing, we need to consider various data distributions, i.e., correlated, anti-correlated, and independent. Our focus is on the number of data points that can be pruned by the first MapReduce job. Let n_p be the number of data points pruned during the first MapReduce job.

For independent data distribution, we estimate the density of data points in the d-dimensional space. Let $Z_i = \{p_1[i], \ldots, p_n[i]\}$, where $p_k[i]$ is the value of i-th dimension of point p_k. Thus, the total volume of the whole dataset can be computed as

$$Q = \int_{\min(Z_1)}^{\max(Z_1)} \cdots \int_{\min(Z_d)}^{\max(Z_d)} \mathrm{d}t_1 \cdots \mathrm{d}t_d.$$

Since data is independently distributed, $n_\text{p} = nV_\text{t}/Q$.

For correlated data distribution, the first MapReduce job divides the whole dataset into M partitions. Consider the case that there is only one skyline point in each partition. This indicates there are only M skyline candidates for the second MapReduce job. Thus, the number of pruned data points is $n_\text{p} = n - M$.

For the anti-correlated data distribution, we consider two extreme cases. The first case is that every data point is a skyline. This indicates that the first MapReduce job outputs the same data points as the input, which means $n_\text{p} = 0$. For the other extreme, each partition has exactly one skyline. Therefore, the first MapReduce Job outputs M skyline candidates, which means $n_\text{p} = n - M$.

Runtime Analysis of Z-merge

With the number of pruned data points, we discuss the processing time of Z-merge. Let $|\hat{n}|$, d, $|S|$ be the number of skyline candidates, the data dimensionality and the size of the skyline, respectively, where $\hat{n} = n - n_\text{p}$. Based on the dominance based grouping strategy, we assume skyline candidates distribute uniformly in each data Group G_m, and denote this number as $|\hat{n}|/M$. The processing time of Z-merge depends on the overhead of the procedure `UDominate()`, i.e., the height of the ZB-tree of existing skyline points [5], which is $O(\log_d |S|)$, and the number of times procedure `UDominate()` is invoked.

We analyze the processing time w.r.t different data distributions. For independent and anti-correlated data distributions, most of the data points cannot be dominated by others. We consider the worst case scenario where all skyline candidates are parts of the skyline points, i.e., $S = \hat{n}$. Therefore, the number of times procedure `UDominate()` is invoked is the same as the number of internal nodes of `Zsrc`, i.e., $O(|\hat{n}|/M)$, and the overhead of procedure `UDominate()` is $O(d \log_d |\hat{n}|)$ for d-dimensional data. Thus, the running time to merge all ZB-trees is $O(|\hat{n}| d \log_d |\hat{n}|)$. For correlated data distribution, the size of the skyline is small. Suppose each partition outputs one skyline point, and the size of the skyline is $|S| = M$. The overhead of `UDominate()` is $O(d \log_d M)$, so the processing time is $O(Md \log_d M)$, this save the computation cost to merge skyline candidates while pruning some branches of built index.

6 Experimental Study

6.1 Experimental Setup

Datasets.

We evaluate the performance of the proposed techniques using the following two types of datasets. The first type is synthetic data, e.g., independent, anti-correlated and correlated distributed data, that is widely used in the literature ([8], [9], [11], [12]). In our experiments, we vary the data size from 10 million to 110 million, and the data dimensions from 2 to 10. We only show the results for the independent and anti-correlated cases, because results for correlated data exhibit similar trends to the anti-correlated case. We also use three real-world high dimensional datasets [24]: (1) NUS-WIDE[2] is a web image dataset containing 269,648 images. We use 225-D block-wise color moments as the image features, thus obtaining a 225-dimensional data. (2) Flickr[3] is a an image hosting website. We crawled 1 million images and extracted 512 features via the GIST Descriptor [25] (the number of data dimensions is 512). (3) DBPedia[4] data aims to extract structured content from Wikipedia.

2. http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm
3. htttp://www.flickr.com
4. http://wiki.dbpedia.org/About

We extracted 1 million documents, and then applied standard NLP techniques to pre-process the documents, e.g., to remove stop words. We use the Latent Dirichlet Allocation (LDA) model to extract topics, and we keep 250 topics for each document. To evaluate the performance on larger data sizes, we synthetically generate more data while maintaining the same distribution as the original data distribution, e.g., as in [26], [24]. We use $\times s$ to denote the increase in dataset size, where $s \in [5, 25]$ is the increase or scale factor.

Computing Platform. We use a Hadoop cluster consisting of 6 computing nodes (namely Hathi[5]). Each node has an Intel(R) Xeon (R) E5640 2.66 GHz 4-core processor, 32GB of memory. Each node runs Ubuntu 14.04 operating system, Java 1.6.0 with a 64-bit sever VM, and Hadoop 2.4. Meanwhile, we setup one Hadoop cluster (version 2.6) based on the the Amazon EC2 with 48 nodes, each node has an Intel Xeon E5-2666 v3 (Haswell) and 8GB of memory. The performance testing for the synthetic datasets (e.g., independent, anti-independent and correlated data distribution) is carried on Hathi, while the experimental study for real-world datasets, i.e., NUS-WIDE, Flickr and DBpedia datasets is carried out on the Amazon EC2.

Evaluated Approaches. We evaluate the following data partitioning approaches:

Grid-based partitioning [9], [11]. In particular, we normalize the value of each data point by the projection-based method in [7].

Angle-based partitioning [8]. We implemented this dynamic partitioning approach to learn a data partitioning rule such that each partition has the same amount of input data points.

MR-GPMRS [12]. This is the latest MapReduce skyline computing approach based on grid partitioning and bitstring. It uses multiple reducers to compute global skyline from skyline candidates. We use the implementation generously provided by the authors.

Z-order curve based partition and grouping. This is our approach introduced in Section 4. We implement three strategies: (1) **Naïve-Z**: Z-order data partitioning in Section 4.1; (2) **ZHG**: Z-order based partition+Heuristic Grouping in Section 4.2; (3) **ZDG**: Z-order based partition+Dominance-based Grouping in Section 4.3.

For each partitioning approach, we adopt two centralized skyline computation algorithms. The first one is sorting the data first, then computing the skyline via the Block-Nest-Loop [1] (short as **SB**). The second one is the state-of-the-art centralized skyline algorithm called Z-search [5] (short as **ZS**). Based on the data partitioning approaches and the skyline computation algorithm, we ended up with several computation strategies. For example, **Grid+SB** and **Grid+ZS** means grid partitioning data, then **SB** and **ZS** are applied for each data partition, respectively. Similarly, **Angle+SB** and **Angle+ZS** follow the same way of definition. In addition, **ZDG+ZS** means **ZDG** divides the dataset into different partitions, then **ZS** algorithm is used to compute skyline candidates in each reducer. We denote by **ZM** the **Z-merge** algorithm developed in Section 5 to merge the skyline candidates. We also compare the runtime performance of **Z-merge** against **ZS** when merging the skyline candidate, where **ZDG+ZS** means that **ZS** is used to merge skyline candidates in the third stage.

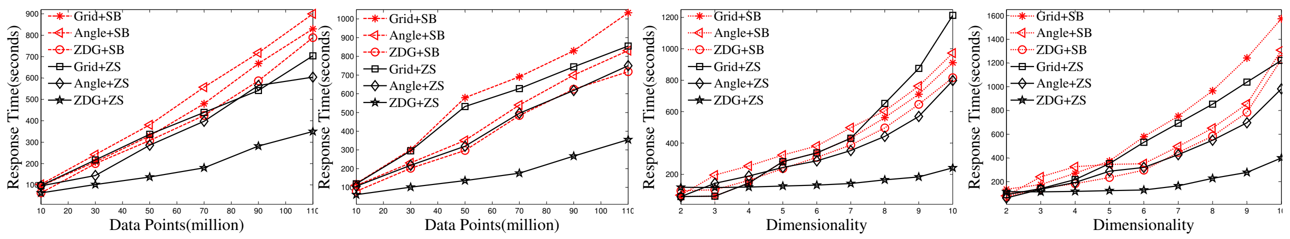
5. https://www.rcac.purdue.edu/compute/hathi/

In order to reduce the shuffling cost between the mappers and the reducers, all the approaches use the same combiners to compute skyline candidates. We measure several parameters, including the query processing time, the number of skyline candidates and the number of partitions. The query processing time includes the data preprocessing time (i.e., time to learn the data partitioning rule from the sampling data), launching the job, network communication, computation time and results writing to the distributed file system (HDFS). For each data partitioning mechanism, i.e., Grid, Angle and Z-order based, the same amount of sample data is collected.

6.2 Effect of Load Balancing

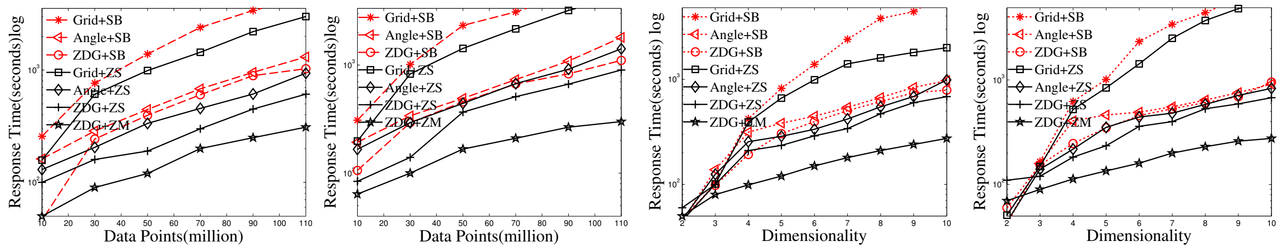
Figures 7a and 7b show the execution time as the dataset size increases (from 10 million to 110 million) for the synthetic dataset, where the data dimensionality is 5, and the number of data partitions is fixed to 32. The plotted lines with the same color use the same centralized skyline algorithm. When the skyline algorithm **SB** is applied, **ZDG** performs better than the other two partitioning approaches, but the improvement is not significant. The reason is that **SB** performs pair-wise dominance testing for incomparable data points, and the runtime time of skyline computation becomes the bottleneck of the MapReduce Job. Therefore, to clearly understand how the partitioning approach influences load balance, the state-of-art skyline computation algorithm **ZS** is adopted. **ZS** indexes incomparable data points into different RZ-regions, and reduces the dominance testing among data points. Its performance is bound by the number of input data points and skyline candidates. We observe **ZDG+ZS** achieves more than 5 times speedup against the other two partitioning approaches. The reason is that **ZDG** guarantees that each reducer performs the computation on a similar amount of dataset points, and receives a similar amount of skyline points.

Figures 7c and 7d plot the response time by varying data dimensionality from 2 to 10 for independent and anti-correlated data, when the number of data points is fixed as 50 million. As observed in the figure, Grid-based and Angle-based partitioning approaches show similar performance for lower number of dimensions. However, the execution time increases exponentially when the dimension varies from 5 to 10, which is due to the curse of dimensionality and both data partitioning approaches fail to guarantee that each partition is allocated the same amount of input dataset. However, the



(a) Independent (b) Anti-correlated (c) Independent (d) Anti-correlated

Fig. 7: Effect of load balancing by changing data cardinality and dimensions



(a) Independent (b) Anti-correlated (c) Independent (d) Anti-correlated

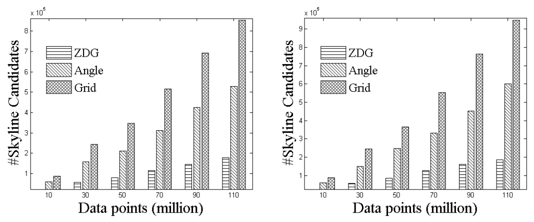
Fig. 8: Effect of data pruning by changing data cardinality and dimensions

execution time of **ZDG** grows smoothly when the data dimensionality varies from 5 to 10, and it gains 5 times speedup against the other two methods (i.e., Grid-based and Angle-based). This confirms our discussion that data partition-grouping algorithm **ZDG** can efficiently filter out non-skyline data points by merging different partitions, and those partitions merged into the same group stand higher possibility to dominate each other. This dominance comparison is based on the Z-address of data, which is not sensitive to data dimensionality. Furthermore, for the same data partitioning approach, **ZS** algorithm shows the best performance, when the data has higher dimensions (i.e., larger than 7). However, it is slightly slower than **SB** for lower number of dimensions, because of the overhead of managing the ZB-trees.

From the results in Figure 7, we also notice that the Angle-based partitioning method does not always perform better than the Grid-based partitioning method, because the Angle-based method is sensitive to the spatial distribution of skyline points. When the data size increases, the runtime of **Angle+SB** degrades more than that of Grid-based and Z-order-based approaches. This reflects the fact that skyline points do not stay near the origin of axes in high-dimensional spaces for uniform distribution.

6.3 Effect of Data Pruning

Figure 9 shows the number of skyline candidates for the different approaches. For **ZDG**, our proposed data partition grouping rule allows each reducer to filter out skyline candidates based on the dominance relationship. Thus, **ZDG** gains more than 5 and 3 times of pruning ability than Grid-based and Angle-based methods, respectively. The power of data pruning can also be observed from the running time to merge skyline

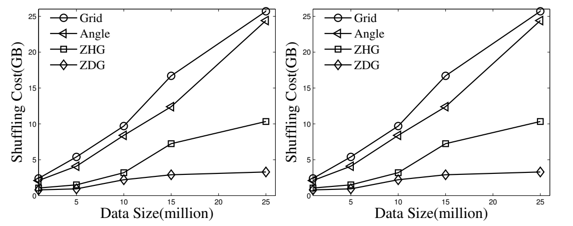


(a) Independent (b) Anti-correlated

Fig. 9: Effect of pruning: skyline candidates w.r.t data size

candidates. Naturally, the more skyline candidates, the more time it takes to compute the skyline. Figure 8a and Figure 8b show the corresponding running time to merge skyline candidates as the dataset size increases (i.e., from 20 million to 110 million). **ZDG+ZM** always shows the best performance, followed by Angle-based and Gird-based methods. To test the effect of dimensionality on performance of computing skyline candidate, we illustrate the results in Figure 8c and Figure 8d. For Grid- and Angle-based approaches, when the number of dimensions increases, the running time increases quadratically. However, for **ZDG+ZM**, when the number of dimension varies from 4 to 10, the running time increases smoothly because the Z-order curve maps data into one dimension, and the skyline computation algorithm does not depend on the data dimensionality.

From Figure 8, we observe that the proposed **Z-merge** approach improves the final skyline computation, as the number of input data points and data dimensionality increase. As shown in Figure 8a and Figure 8b, the running time of **ZM** method is always shorter than that of **SB** by more than 10 times. This is determined by the fact that merging the index is more efficient than searching the



(a) Nus-Wide (b) DBPedia

Fig. 10: Shuffle cost of parallel skyline query processing

skyline set from skyline candidates based on the sorting approach. From Figure 8c and Figure 8d, we also observe that **ZM** always outperforms **SB** as the number of data dimensions increases. Finally, the newly proposed **Z-merge** achieves one order of magnitude speedup against **Z-search** for merging the skyline candidates, since Z-merge utilize multiple indexes to reduce the all-pairs comparison testing in the Z-search algorithm.

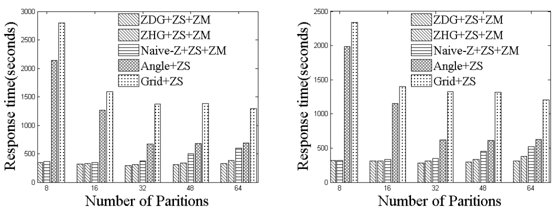
6.4 Effect of Grouping Strategy

Figure 11 shows the response time w.r.t the number of data partitions. Note that the number of reducers is set to be the number of data partitions, the dataset size is 60 millions with 6 dimensions, and the number of partitions for **ZHG** and **ZDG** approaches is the number of groups. Angle-based and Grid-based partitioning methods spend less time as the number of partitions increases, and remain stable when the number of partitions is greater than 32. The running time of **Naïve-Z** grows quickly once the number of partitions is greater than 32. This is attributed to the fact that the Z-order curve would cluster data points with higher similarity into one partition, and the possibility that data points dominate each other is low, leading to a larger number of skyline candidates and degrading the runtime performance. However, **ZDG** obtains stable runtime performance as the number of partitions increases, because dominance based partition grouping can merge different partitions into groups based on their dominance relationship rather than the similarity of data points.

We also measure the effect of data size on the shuffling cost. Figure 10 gives the data shuffle costs when the data size varies. The smaller the shuffle costs, the better the performance is. We observe that the shuffle costs for Angle and Grid based data partition, are 10 times bigger when compared to the Z-order based approaches (e.g., **ZHG** and **ZDG**). The reason is that the dominance based partition technique groups data points based on their dominance relationship, hence, more redundant intermediate data are pruned. Notice that the data shuffling cost for PGBJ increases linearly with the data size.

6.5 Scalability

We now investigate the scalability for four approaches on the synthetic data and real-world dataset. Figure 12



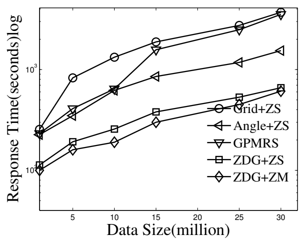
(a) Independent (b) Anti-correlated

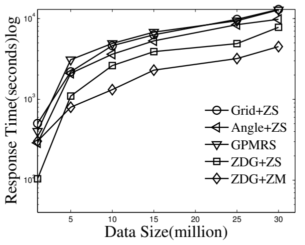
Fig. 11: Response time against the number partitions

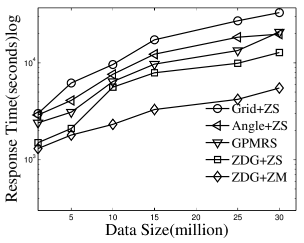
presents the results by varying the data sizes from 2 million to 30 million. We only show the results of Z-search for local skyline computation, since it outperforms the sort based approach as presented in Section 6.2. Meanwhile, the experimental results for anti-distribution show the similar trend as the independent distribution, we omitted it here. From Figure 12, we see that the overall execution time of all the three existing approaches (i.e., Grid, Angle and MR-GPMRS) grow quadratically when the data size increases. This is determined by the fact that the number of incomparable data points increases quadratically with the data size, and existing approaches cannot prune skyline candidates effectively. However, the response time of **ZDG+ZM** increases smoothly as the dataset size increases. For instance, **ZDG+ZM** achieve 5, 8, 10 times speedup against **MR-GPMRS**, **Angle+ZS** and **Grid-ZS**, respectively. This is attribute from the facts z-order based partition grouping is able to partition the high dimensional data in a balanced way.

6.6 Effect of Data Sampling

We now study how data sampling influences the number of skyline candidates and query processing time. Figure 13 presents the results by varying sampling ratio from 0.5% to 4% on the dataset with independent distribution. We observe that **ZDG** always performs best on the number of skyline candidates and skyline query processing time, followed by **ZHG** and **Naïve-Z**. The skyline candidates can be pruned more as the sampling percentage increases for all three Z-order based partitioning approaches. Because the approach **ZHG** and **Naive-Z** are more sensitive to sampling data sizes, their runtime performance is impacted more by the number of sampling data. However, **ZDG** behavior more stable, since it depends on the dominance-volume, which is not influenced by data sampling. In the experiment, we find that the **ZDG** spends longer time on learning the data partitioning rule. For example, when the sample ratio is 0.5%, the **Naïve-Z**, **ZHG** and **ZDG** takes 60, 120, 150 seconds for data preprocessing, respectively. Although **ZDG** incurs higher preprocessing time than the other two approaches, **ZDG** achieves the best skyline query processing time, since the rumtime of data preprocessing is circumvented by gains in the runtime of stage 2 and 4, i.e., computing skyline candidates in parallel and the final skyline candidates merging.


(a) Independent


(b) Nus-Wide


(c) Fliker

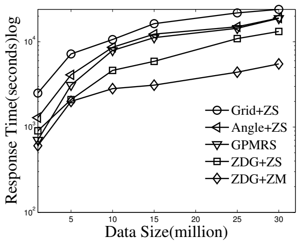
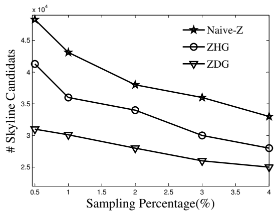

(d) DBPedia

Fig. 12: Speedup and scalability: Running time of Parallel Skyline Query Processing.


(a) Skyline Candidates

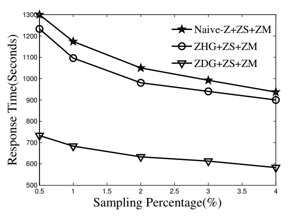

(b) Running Time

Fig. 13: Effect of sampling w.r.t skyline candidates and execution time

7 Conclusion

This paper demonstrates an efficient solution for the problem of parallel skyline query processing. Extensive experiments on a Hadoop cluster demonstrates that our proposed method is efficient, robust, scalable and can achieve up to one order of magnitude speedup over existing state-of-the-art approaches.

References

[1] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001.

[2] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database System*, 2005.

[3] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *ICDE*, 2003.

[4] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: an online algorithm for skyline queries," in *VLDB*, 2002.

[5] K. C. K. Lee, B. Zheng, H. Li, and W.-C. Lee, "Approaching the skyline in z order," in *VLDB*, 2007.

[6] W.-T. Balke, U. Gntzer, and J. Zheng, "Efficient distributed skylining for web information systems," in *EDBT*, 2004.

[7] H. Köhler, J. Yang, and X. Zhou, "Efficient parallel skyline processing using hyperplane projections," in *SIGMOD*, 2011.

[8] A. Vlachou, C. Doulkeridis, and Y. Kotidis, "Angle-based space partitioning for efficient parallel skyline computation," in *SIGMOD*, 2008.

[9] P. Wu, C. Zhang, Y. Feng, B. Zhao, D. Agrawal, and A. Abbadi, "Parallelizing skyline queries for scalable distribution," in *EDBT*, 2006.

[10] K. Hose, C. Lemke, and K.-U. Sattler, "Processing relaxed skylines in pdms using distributed data summaries," in *CIKM*, 2006.

[11] B. Zhang, S. Zhou, and J. Guan, "Adapting skyline computation to the mapreduce framework: Algorithms and experiments," in *EDBT*, 2011.

[12] K. Mullesgaard, J. L. Pederseny, H. Lu, and Y. Zhou, "Efficient skyline computation in mapreduce," in *EDBT*, 2014.

[13] P. Koutris and D. Suciu, "Parallel evaluation of conjunctive queries," in *PODS*, 2011.

[14] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skew-resistant parallel processing of feature-extracting scientific user-defined functions," in *SoCC*, 2010.

[15] J. J. Levandoski, "Extensible preference evaluation in database systems," Ph.D. dissertation, 2011.

[16] B. Catania and L. C. Jain, *Advanced Query Processing: Volume 1 Issues and Trends*. Springer Publishing Company, Incorporated, 2012.

[17] K. Hose and A. Vlachou, "A survey of skyline processing in highly distributed environments," *VLDB*, 2012.

[18] A. Cosgaya-Lozano, A. Rau-Chaplin, and N. Zeh, "Parallel computation of skyline queries," in *HPCS*, 2007.

[19] J. Zhang, X. Jiang, W. S. Ku, and X. Qin, "Efficient parallel skyline evaluation using mapreduce," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 1996–2009, July 2016.

[20] Y. Park, J.-K. Min, and K. Shim, "Parallel computation of skyline and reverse skyline queries using mapreduce," in *VLDB*, 2013.

[21] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *VLDB*, 1998.

[22] F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, and R. Bayer, "Integrating the ub-tree into a database system kernel," in *VLDB*.

[23] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, 1985.

[24] M. Tang, Y. Yu, W. G. Aref, Q. M. Malluhi, and M. Ouzzani, "Efficient processing of hamming-distance-based similarity-search queries over mapreduce." in *EDBT*, 2015, pp. 361–372.

[25] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," pp. 145–175, 2001.

[26] R. Vernica, M. J. Carey, and C. Li, "Efficient parallel set-similarity joins using mapreduce," ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 495–506.

Mingjie Tang won his PhD at Purdue University. His research interests include database system and big data infrastructure. He has an MS in computer science from the Graduate University of Chinese Academy of Sciences, BS degree from Sichuan University, China.

Yongyang Yu is a PhD student at Purdue University. His research focus is on big data management and matrix computation.

Walid G. Aref is a professor of computer science at Purdue. He is an associate editor of the ACM Transactions of Database Systems (ACM TODS) and has been an editor of the VLDB Journal. He is a senior member of the IEEE, and a member of the ACM. Professor Aref is an executive committee member and the past chair of the ACM Special Interest Group on Spatial Information (SIGSPATIAL).

Qutaibah M. Malluhi joined Qatar University in September 2005. He is the Director of the KINDI Lab for Computing Research. He served as the head of Computer Science and Engineering Department at Qatar University between 2005-2012.

Mourad Ouzzani is a Senior Scientist with the Qatar Computing Research Institute (QCRI), Qatar Foundation. He is interested in research and development related to data management and scientific data and how they enable discovery and innovation in science and engineering. Mourad received the Purdue University Seeds of Success Award in 2009 and 2012.