Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 401618, 10 pages
http://dx.doi.org/10.1155/2014/401618

Research Article

Grid-PPPS: A Skyline Method for Efficiently Handling Top-k Queries in Internet of Things

Sun-Young Ihm,[1] Aziz Nasridinov,[2] and Young-Ho Park[1]

[1] *Department of Multimedia Sciences, Sookmyung Women's University, Cheongpa-ro 47-gil 100, Yongsan-gu, Seoul 140-742, Republic of Korea*
[2] *School of Computer Engineering, Dongguk University at Gyeongju, 123 Dongdae-ro, Gyeongju, Gyeongbuk 780-714, Republic of Korea*

Correspondence should be addressed to Young-Ho Park; yhpark@sm.ac.kr

Received 22 January 2014; Accepted 7 April 2014; Published 8 May 2014

Academic Editor: Young-Sik Jeong

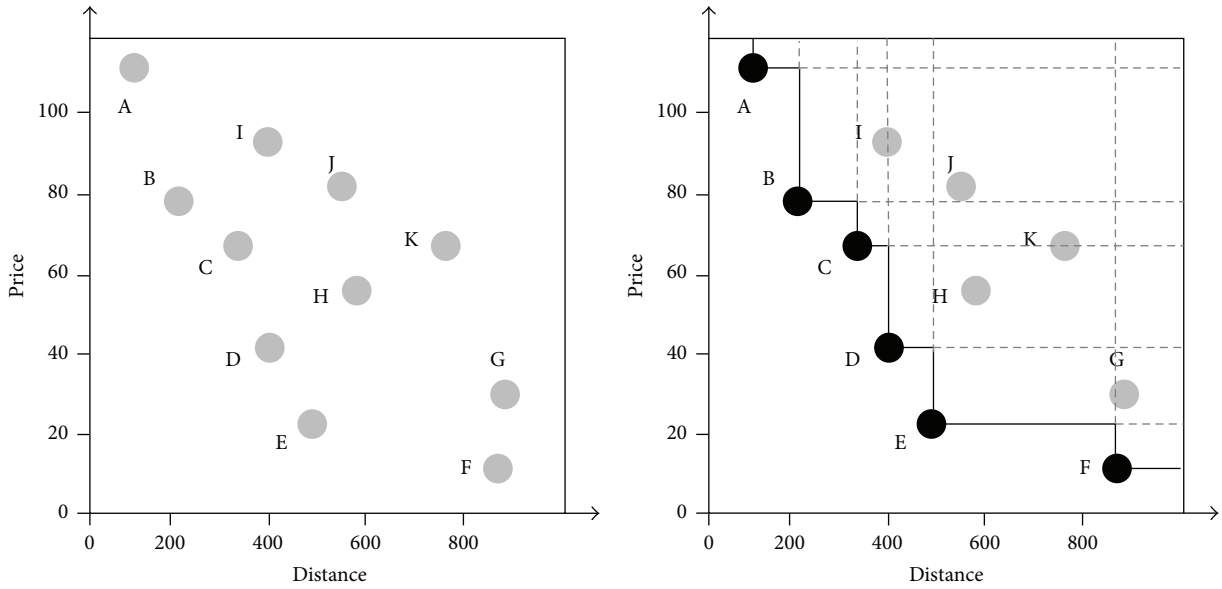
A rapid development in wireless communication and radio frequency technology has enabled the Internet of Things (IoT) to enter every aspect of our life. However, as more and more sensors get connected to the Internet, they generate huge amounts of data. Thus, widespread deployment of IoT requires development of solutions for analyzing the potentially huge amounts of data they generate. A top-k query processing can be applied to facilitate this task. The top-k queries retrieve k tuples with the lowest or the highest scores among all of the tuples in the database. There are many methods to answer top-k queries, where skyline methods are efficient when considering all attribute values of tuples. The representative skyline methods are soft-filter-skyline (SFS) algorithm, angle-based space partitioning (ABSP), and plane-project-parallel-skyline (PPPS). Among them, PPPS improves ABSP by partitioning data space into a number of spaces using hyperplane projection. However, PPPS has a high index building time in high-dimensional databases. In this paper, we propose a new skyline method (called Grid-PPPS) for efficiently handling top-k queries in IoT applications. The proposed method first performs grid-based partitioning on data space and then partitions it once again using hyperplane projection. Experimental results show that our method improves the index building time compared to the existing state-of-the-art methods.

1. Introduction

A rapid development in wireless communication and radio frequency technology has enabled the Internet of Things (IoT) to enter every aspect of our life. The IoT is part of the internet of the future and will comprise billions of intelligent communicating "things" which will have sensing, actuating, and data processing capabilities [1]. For example, the things in IoT can be smart devices in home or home appliances such as refrigerator, washing machine, and air conditioner, which have controllable devices. Restaurants, hotels, and countries can be also considered as the things in IoT, since they are connected and communicate with each other. However, as more and more sensors get connected to the Internet, they generate enormous amounts of data. Thus, widespread deployment of IoT requires development of solutions for analyzing the potentially huge amounts of data they generate [2–4]. A top-k query processing can be applied to facilitate this task.

The top-k query finds k tuples with the lowest or the highest scores among all of the input tuples. When a database is large, it may take long computing time to find a complete answer to a query. Most users, however, are interested in looking at just a few top results, which are ranked by a small set of attribute values, and they want to see the results immediately after they issue the query [5]. We can apply this notion to find the top-k results in huge amounts of data in IoT applications. Example 1 presents the scenario to find the top-k results in IoT applications.

Example 1. Consider a user John, who wants to have a dinner in an Italian restaurant. He defines the following criteria for the search: the distance of restaurant from his home should be less than 800 meters and price should be less than



(a) Data of Example 1

(b) Skyline of Example 1

Figure 1: Graphical representation of data and skyline.

Table 1: The list of restaurants in Example 1.

Number	Name	Distance (100 m)	Price ($10)	Score
A	Mago	1.0	11.0	No score
B	Little Pasta	2.0	8.0	5.6
C	La Tavola	3.5	7.0	5.6
D	Olive Garden	4.0	4.0	4.0
E	Alto	5.0	2.0	3.2
F	Melrose	8.8	1.0	No score
G	Morton House	9.0	3.0	No score
H	Applebee's	6.0	6.0	6.0
I	Boulevard	4.0	9.0	No score
J	IHOP	5.3	8.5	No score
K	Shadow Brook	7.6	7.0	7.24

85 US Dollars. In order to find a restaurant that best suites his interest, John makes a scoring function f as $f(t) = 0.4 * \text{distance} + 0.6 * \text{price}$, where t is the tuple of database. Here, we can think of the restaurant as a thing in IoT. All restaurants are connected to the Internet, which forms the network of IoT. Finding the top-k results among a large amount of restaurants could save John's time. The query shown below is based on the PostgreSQL syntax.

SELECT $*$
FROM Restaurant R
WHERE R.distance <8.0 AND R.price <8.5
ORDER BY $f(t)$

The list of restaurants and their scores are shown in Table 1. These restaurants can be represented in two-dimensional space as shown in Figure 1(a). The Alto, E, is top-1 answer to the query with a score of 3.2 and Olive Garden, D, is top-2 answer to the query with a score of 4.0. Since restaurants A, F, G, I, and J have higher values for distance and price, they do not satisfy the requirements provided by John. Thus, the scores for these restaurants are not calculated.

To answer the top-k queries efficiently, building an index by accessing the subset of database is needed. The skyline methods are representative methods for answering the top-k queries by constructing skyline as an index. These methods express data tuples as objects in a d-dimensional space and then construct a skyline. Here, d is the number of attributes of a database. The skyline methods are efficient for queries in a database with a large number of attributes and data. In Figure 1(b), the rectangular black line, composed of black points, represents the skyline. The skyline points do not dominate each other. We can answer top-k queries only by reading the skyline points, since the skyline can be considered as an index. The soft-filter-skyline (SFS) algorithm [6], which is the state-of-the-art method, presorts the objects by calculating entropy value of object. The angle-based space partitioning (ABSP) [7] and plane-project-parallel-skyline (PPPS) [8] partition data space into a number of subregions in order to reduce the computing time. PPPS improves ABSP by partitioning data space into a number of spaces using hyperplane projection. However, PPPS has a high index building time in high-dimensional databases.

In this paper, we propose a new skyline method for efficiently handling top-k queries in IoT applications. This paper focuses on the effectiveness of grid-based partitioning. More precisely, the contributions we make in this paper are as follows.

(i) We propose a new skyline method (called Grid-PPPS) for efficiently handling top-k queries in IoT applications. The proposed method first performs grid-based partitioning on data space and then partitions it once

again using hyperplane projection. This reduces the time complexity of the PPPS.

(ii) We show the performance advantages of the Grid-PPPS through the comparison of the index building time and number of dominating objects compared to PPPS.

The rest of this paper is organized as follows. Section 2 describes existing work related to this paper. Section 3 presents the proposed method for computing Grid-PPPS and Section 4 demonstrates the results of performance evaluation. Section 5 summarizes and concludes the paper.

2. Related Work

In this section, we discuss the existing literature. In Section 2.1, we review data management solutions in IoT, and, in Section 2.2, we explain the index building methods for top-k queries.

2.1. Data Management Methods in IoT. Generally speaking, all things on the IoT may generate a huge amount of data that contains different kinds of useful information. However, how to handle such big data and how to retrieve the valuable information have become hot research topic in recent years. Several index building methods for handling massive amount of IoT data are proposed. Ma et al. [9] proposed an update and query efficient index framework (UQE-Index) based on key-value store that can support both multidimensional query and high insert throughput. In order to effectively reduce the index update times and decrease the index maintenance cost, the authors proposed a dynamic data partition strategy that can make sure that the data is evenly distributed into each region in HBase and the data that is close in time and space dimension is usually stored in the same regions.

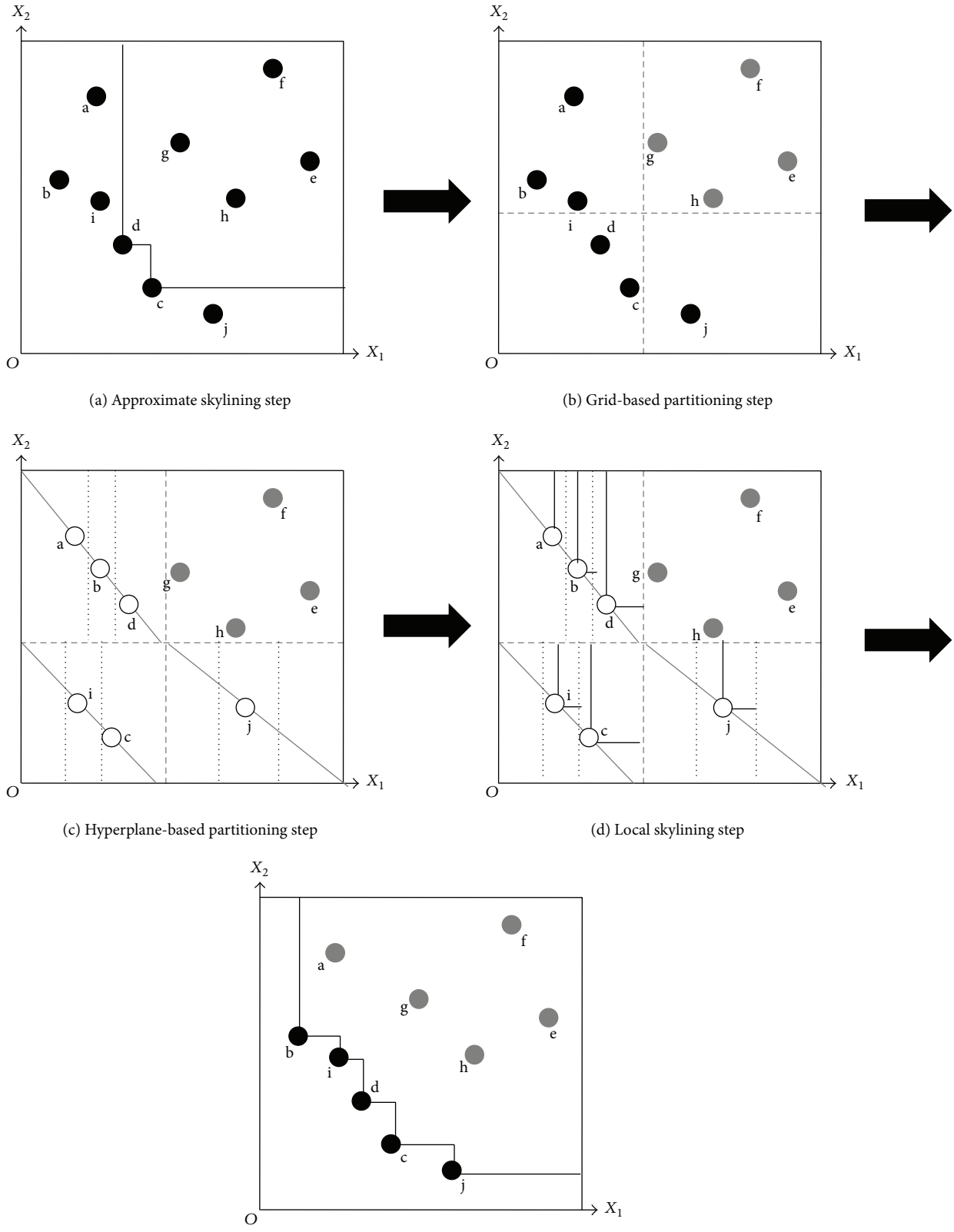
In order to address the problem of high dimensionality in IoT data, Huang et al. [10] proposed dynamic skyline cube (SKYCUBE) computation to efficiently balance the computation update and costs in IoT. The authors proposed an efficient grid-based ADSCIT (algorithm for dynamic SKYCUBE computation in the Internet of Things) which consists of two modules: continuous maintenance module (CMM), which incrementally updates the nonpseudo objects, and progressive computation module (PCM), which can rapidly obtain the skyline cube from the updated nonpseudo objects. In order to integrate the proposed two modules, a grid-based evaluation method that uses regular grid index is proposed.

Elkheir et al. [11] surveyed the data management solutions that are proposed for IoT and proposed a data management framework that takes into consideration the drawbacks of existing approaches. The proposed framework adapts a federated, data and sources centric approach to link diverse things with their abundance of data to the potential applications and services. Data mining technologies can also be used to discover the hidden information in the data of IoT, which can be used to improve the performance of the system or to enhance quality of services this new environment can provide [12]. Tsai et al. [12] surveyed research on how to connect data mining technologies to the IoT, which include clustering, classification, and frequent patterns mining technologies, from a different perspective. The authors also discuss changes, potentials, open issues, and future trends of applying data mining to the IoT.

2.2. Index Building Methods for Top-k Queries. To construct an index efficiently, skyline and convex hull methods are representative methods. These methods construct an index as a list of layers and consist of objects which are not dominated by each other. The computing cost of skyline methods is much lower than that of convex hull methods; however, the number of objects in each layer of skyline methods is much larger than that of convex hull methods. Thus, the skyline methods are mainly used in the applications where insertion, update, and deletion operations are frequently occurring on objects. Since such applications need to construct skyline more frequently, they require small computing time. On the other hand, objects in convex hull methods are not updated often. Thus, these methods are used in the applications where top-k query processing is performed. This is because a layer in convex hull methods consists of small number objects, which results in rapid processing of top-k queries. In this paper, we focus on reducing the index construction of skyline in which data is frequently updated.

2.2.1. Skyline Methods. The skyline methods are useful when answering top-k queries by accessing only a subset of the database. These methods have an advantage of low index building cost. The skyline operation was first introduced by Köhler et al. [8] and there have been a number of variations of it. The data space partitioning technique is used in many skyline methods for early pruning objects which are not included in skyline. There are several algorithms for constructing skyline that apply space partitioning technique. Grid-based data space partitioning has been commonly used in distributed and parallel skyline processing [8]. The angle-based space partitioning approach (ABSP) [7] is proposed by using hyperspherical coordinates of data objects and improves grid-based space partitioning. Köhler et al. [8] proposed a novel approach called PPPS, which reduces the computing time of ABSP by coordinating the objects using hyperplane projection.

There are also other algorithms for constructing skyline and the representative methods are block nested loops (BNL) [13], SFS [6], and linear elimination sort for skyline (LESS) [14]. BNL sequentially reads the input relation and saves in a window w. When an object o is read, it is compared to objects in w. If an object in w dominates o, BNL eliminates o. Otherwise, o dominates some objects in w; these are deleted from w and o is added to [13]. The SFS algorithm [6] improves BNL by presorting the input relation according to the entropy value of object. LESS is an improvement of SFS that essentially combines aspects of a number of the established algorithms [14]. LESS discards some dominating objects earlier; thus this has the advantage of reducing the number of pairwise comparisons between the objects than



(a) Approximate skylining step

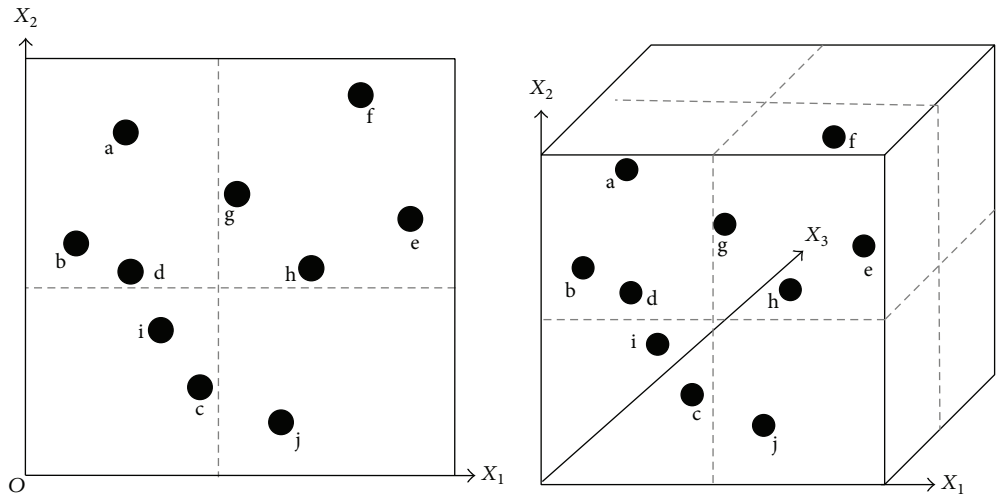
(b) Grid-based partitioning step

(c) Hyperplane-based partitioning step

(d) Local skylining step

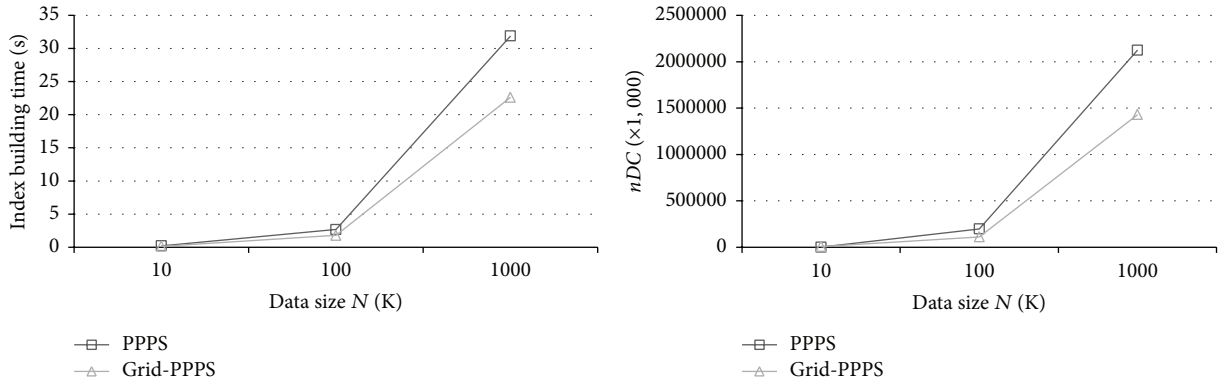
(e) Merging step

Figure 2: The overall procedure for processing Grid-PPPS in the two-dimensional data space.



(a) Grid-based partitioning in two-dimensional data space (b) Grid-based partitioning in three-dimensional data space

Figure 3: The example of grid-based partitioning.



(a) Computing time as N is varied ($d = 8$) (b) nDC as N is varied ($d = 8$)

Figure 4: The comparison of the computing time and nDC as N is varied related to Experiment 1.

SFS. However, the number of comparisons is still large. There also has been a growing interest in distributed [15, 16] and parallel [17, 18] skyline computation lately.

2.2.2. Other Methods. The convex hull methods construct the layer of edge objects in a convex hull shape and discard other objects. The layer size of convex hull methods is smaller than that of skyline methods; however, the index building time of convex hull methods is higher than that of skyline methods. The representative convex hull methods are ONION [19] and HL-Index [5]. ONION [19] builds convex hull as an index by constructing a boundary with the edge objects. That is, the objects of the first layer encircle the other objects. ONION builds a second layer in the same manner and finally constructs a list of layers as a result. HL-Index [5] builds a convex hull as ONION does and sorts lists additionally for retrieving top-k results efficiently.

In order to reduce the index building time of convex hull methods, there are some methods that combine convex hull and skyline methods. For example, Ihm et al. [20] proposed the approximate convex skyline (AppCS) method that constructs skyline over the entire objects and then partitions it. Further, AppCS builds an approximate convex hull in each partitioned region with virtual objects. Another method that focuses on reducing index building time of convex hull is proposed in [21]. The authors proposed a method called approximate convex hull index (aCH-Index) that computes the skyline over the entire set of objects, partitions the region into multiple subregions to reduce the computing time of convex hull in all origins, and then computes the convex hull in each subregion.

3. Grid-PPPS

In this section, we explain the proposed methods, Grid-PPPS. As explained in Section 2.1, the PPPS [8] improves

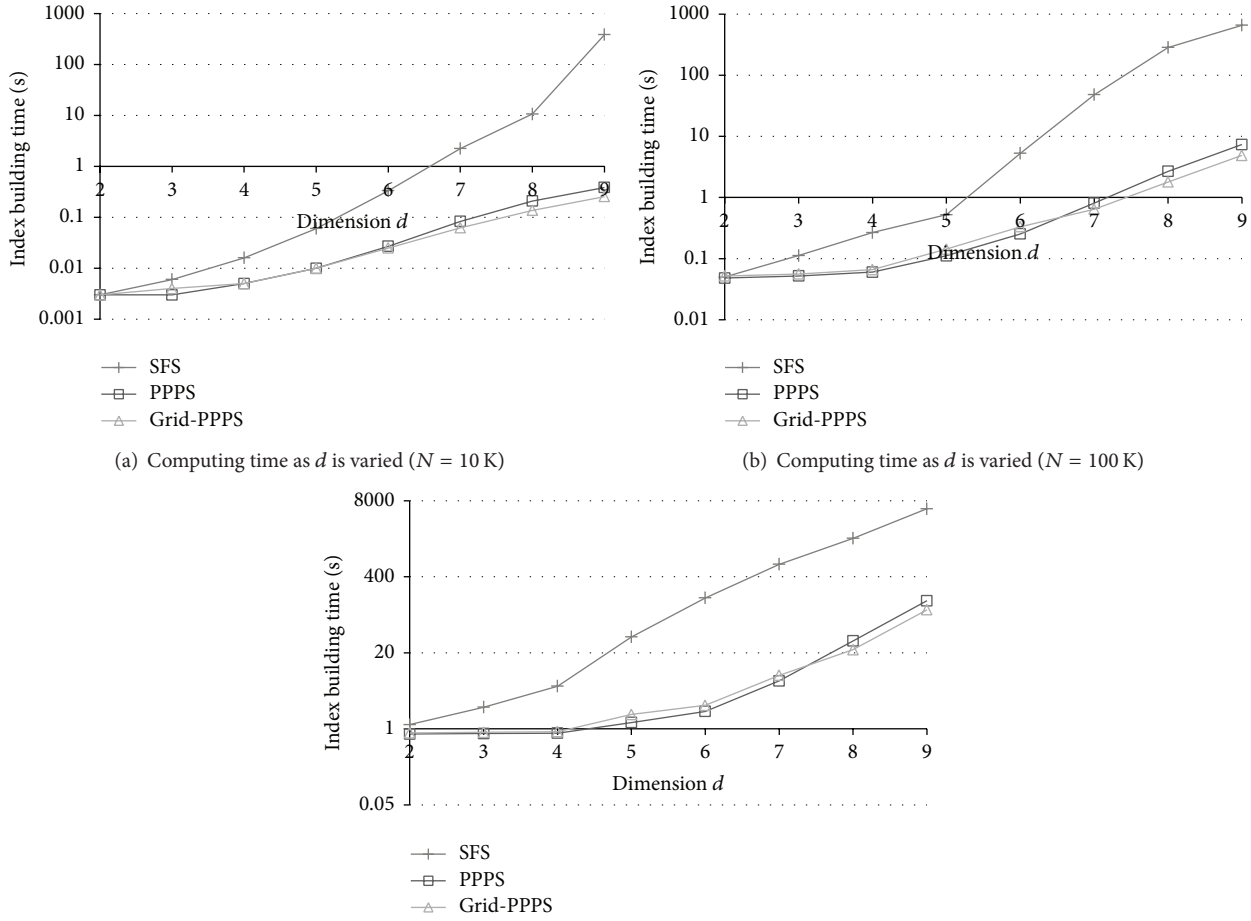
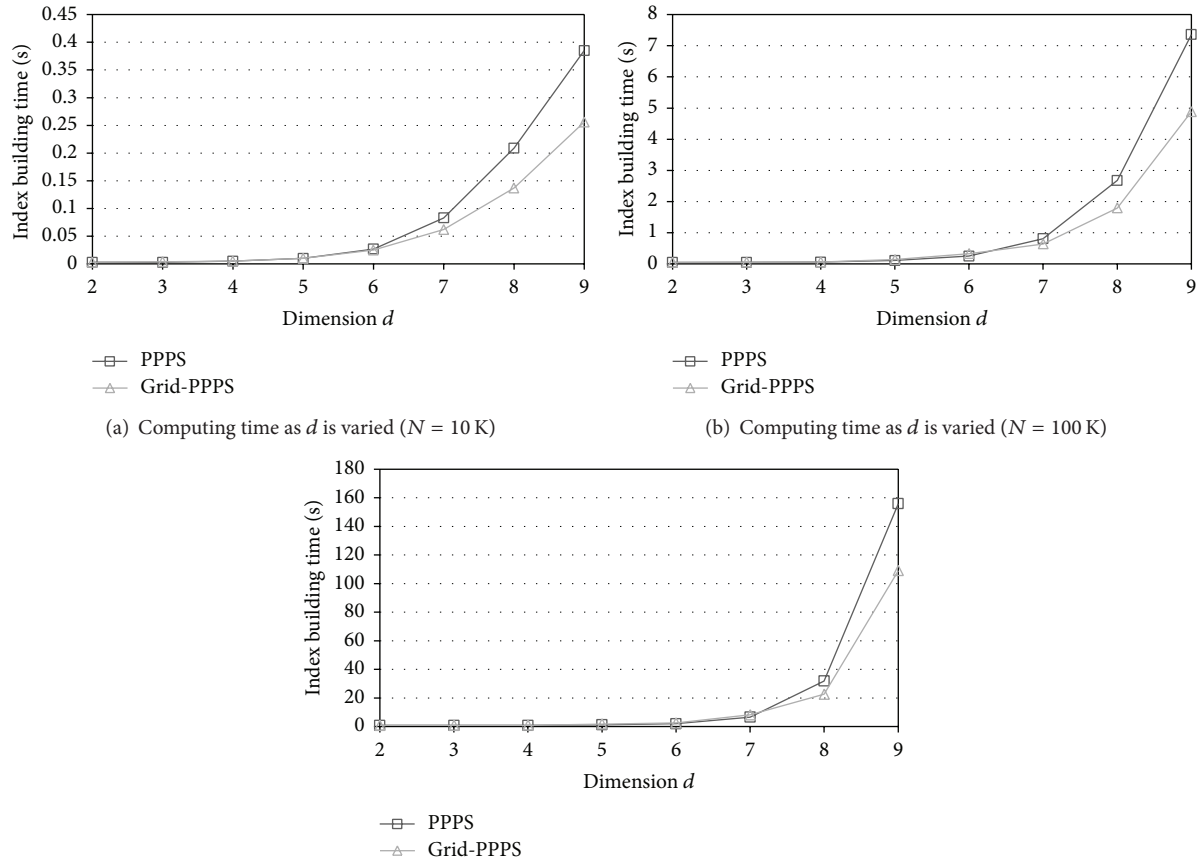


Figure 5: The comparison of the computing time of Grid-PPPS and SFS as d and N are varied related to Experiment 2.

the indexing building time of ABSP [7]. However, PPPS has a high index building time in high-dimensional databases. The Grid-PPPS reduces the time complexity of the PPPS. The Grid-PPPS is constructed by five steps as shown in Figure 2: (a) approximate skylining step, (b) grid-based partitioning step, (c) hyperplane-based partitioning step, (d) local skylining step, and (e) merging step. For the convenience of the explanation, Figure 2 shows the procedure of processing Grid-PPPS in two-dimensional region. We explain each step in detail from Sections 3.1 to 3.5.

3.1. Approximate Skylining Step. In the first step, Grid-PPPS constructs approximate skyline. This step is shown in Figure 2(a). Computing the exact skyline of all tuples set T can be expensive, since each tuple should be compared to many other tuples. However, we can prune several tuples with the few comparisons. We prune the objects by calculating the entropy value of each object. We select several tuples, which have low entropy value, and then make a small set $S \subset T$ with those tuples. By the small set S, some tuples in T are dominated by S, and those tuples can be eliminated safely. Since we pick the tuples according to entropy value, we can discard more tuples. Finally, we can get approximate skyline. Importantly, for fixed size S, computing the approximate skyline can be performed in a linear time with a single pass over the dataset [8].

3.2. Grid-Based Partitioning Step. In the major step that is grid-based partitioning step, Grid-PPPS partitions the data space into b subspaces using grid-based partitioning technique. A grid is something which is in a pattern of straight lines that cross over each other, forming squares. Many applications are using grid-base technique, since it is simple and has low computing cost [22–24]. The grid-based partitioning scheme is based on recursively dividing some dimension of the data space into two parts [7]. The computing time of grid-based partitioning is lower than other partitioning techniques, because grid-based partitioning is simple and cheap to compute. Thus, we partition objects, which are obtained from approximate skylining step into b spaces with grid-based partitioning technique. Figure 3(a) shows the example of grid-based partitioning in two-dimensional data space, and three-dimensional example is shown in Figure 3(b).



(a) Computing time as d is varied (N = 10 K)

(b) Computing time as d is varied (N = 100 K)

(c) Computing time as d is varied (N = 1000 K)

Figure 6: The comparison of the computing time of the Grid-PPPS and PPPS as d and N are varied related to Experiment 2.

3.3. Hyperplane-Based Partitioning Step. In the hyperplane-based partitioning step, Grid-PPPS partitions space into c subspaces using hyperplane-based partitioning, which is proposed in PPPS [8]. We first calculate the formula of the hyperplane such as $x_1 + x_2 + \cdots + x_d = 1$. Next, we project tuples onto the hyperplane, and (1) shows the calculation of projection. Finally we partition space, which consists of projected tuples, into c subspaces:

$$(x_1 \cdots x_d) \longmapsto (x_1 \cdots x_d) \times \frac{1}{x_1 + \cdots + x_d}. \quad (1)$$

3.4. Local Skylining Step. In the local skylining step, Grid-PPPS computes the local skyline in each subspace$_{ij}$. We call local skyline in subspace$_i$ as subskyline$_i$ and use SFS algorithm [6] for computing subskyline. For the construction of local skyline, the dominating calculation, which determines whether the object is in the skyline or not, should be computed between two objects. Grid-PPPS filters out the objects by grid-based partitioning step, and, thus, the number of dominating calculation decreases.

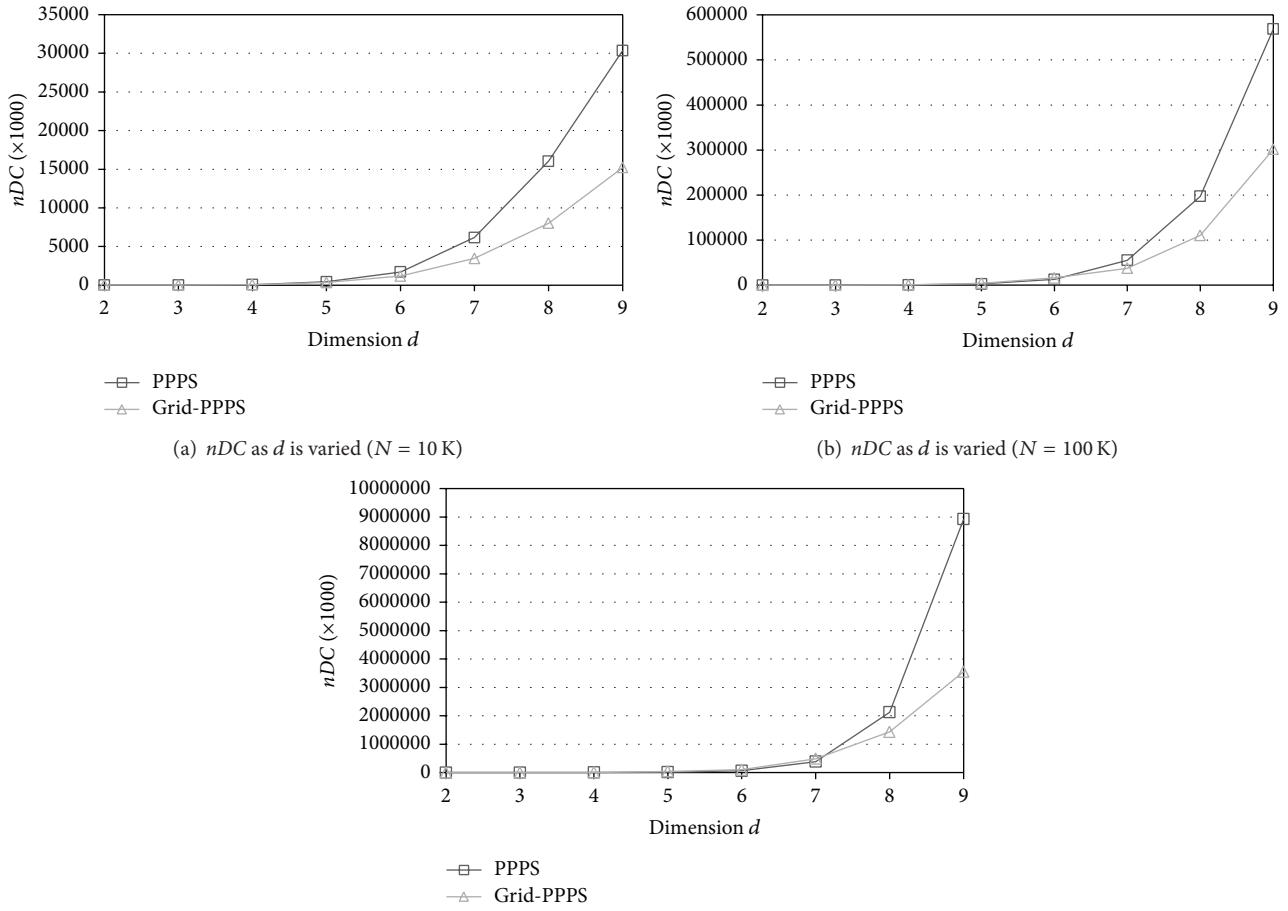
3.5. Merging Step. In the last step, Grid-PPPS combines the subskylines in each subspace. We build a layer by merging the subskylines. Since Grid-PPPS computes subskyline points once again, it combines the subskylines and builds a result layer without losing tuples and overlapping.

4. Performance Evaluation

In this section, we first explain the data and environment in Section 4.1 and then present the results of experiments in Section 4.2.

4.1. Experimental Data and Environment. We have implemented the proposed method using C++. We conduct all the experiments on an Intel i5-760 quad core processor running at 2.80 GHz Linux PC with 16 GB of main memory. We use the uniform dataset for all of our experiment data. We use 10 K, 100 K, and 1000 K data size. We experiment our data in two through nine dimensions.

4.2. Result of Experiments. We compare the computing time and the *nDC* (number of domination calculation) of the Grid-PPPS with the existing methods PPPS [8] and SFS [6]. We use the wall clock time as the measure of the computing time. We measure the computing time *nDC* on the synthetic dataset while varying the data size N and the dimension d.



(a) *nDC* as *d* is varied (N = 10 K)

(b) *nDC* as *d* is varied (N = 100 K)

(c) *nDC* as *d* is varied (N = 1000 K)

Figure 7: The comparison of the *nDC* and *d* and N are varied related to Experiment 3.

The result of the skyline constructed by Grid-PPPS is exactly the same as PPPS. Grid-PPPS improves the index building time of PPPS in large and high-dimensional dataset. When data has 10 K size and under six attributes, the index building time of Grid-PPPS is a little higher than PPPS, because of partitioning step. The number of filtered tuples in Grid-PPPS is similar to PPPS in the small and low-dimensional dataset. However, Grid-PPPS constructs an index much quickly in large and high-dimensional dataset as shown in experiments.

Experiment 1. Computing time and *nDC* as data size N is varied.

Figure 4(a) shows the computing time of Grid-PPPS and PPPS as N is varied from 10 K to 1000 K. The result increases in log scale as shown in Figure 4. The computing time of the Grid-PPPS improves by 1.41–1.52 times over the PPPS. Figure 4(b) shows the *nDC* of Grid-PPPS and PPPS as N is varied from 10 K to 1000 K. The *nDC* of Grid-PPPS improves 1.49–2.00 times over the PPPS.

Experiment 2. Computing time as dimension *d* and data size N are varied.

Figures 5(a), 5(b), and 5(c) show the computing time of Grid-PPPS and PPPS as *d* is varied from 2 to 9 and N is varied from 10 K to 1000 K. The result increases in log scale as shown in Figure 5. Figure 5(a) shows the computing time of the Grid-PPPS improves by 0.75–1.52 times over the PPPS as *d* is varied and N is 10 K. Figure 5(b) shows the computing time of the Grid-PPPS improves by 0.77–1.51 times over the PPPS as *d* is varied and N is 100 K. Figure 5(c) shows the computing time of the Grid-PPPS improves by 0.73–1.43 times over the PPPS as *d* is varied and N is 1000 K. In order to show the precise difference between Grid-PPPS and PPPS, we conduct the experiments shown in Figure 6.

Experiment 3. The *nDC* as dimension *d* and data size N are varied.

Figures 7(a), 7(b), and 7(c) show the *nDC* of Grid-PPPS and PPPS as *d* is varied from 2 to 9 and N is varied from 10 K to 1000 K. The result increases in log scale as shown in Figure 7. Figure 7(a) shows the *nDC* of the Grid-PPPS improves by 1.00–2.01 times over the PPPS as *d* is varied and N is 10 K. Figure 7(b) shows the *nDC* of the Grid-PPPS improves by 0.68–1.89 times over the PPPS as *d* is varied and N is 100 K. Figure 7(c) shows the *nDC* of the Grid-PPPS

improves by 0.65–1.49 times over the PPPS as d is varied and N is 1000 K.

5. Conclusion

As more and more sensors get connected to the Internet, the IoT applications generate enormous amounts of data. In order to solve this problem, in this paper, we have proposed to use a top-k query processing to find the best results among vast amount of data. In order to efficiently handle top-k queries, we have proposed a new skyline method called Grid-PPPS, which performs grid-based partitioning first on data space and then partitions it once again using hyperplane projection. We have compared the proposed method with the state-of-the-art methods, such as PPPS and SFS. The results of experiments demonstrate several times improvement in most cases.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012003797).

References

[1] C. Perera, A. Zaslavsky, C. H. Liu, M. Compton, P. Christen, and D. Georgakopoulos, "Sensor search techniques for sensing as a service architecture for the internet of things," *IEEE Sensors Journal*, vol. 14, no. 2, pp. 406–420, 2014.

[2] C. Zhu, Q. Zhu, C. Zuzarte, and W. Ma, "Developing a dynamic materialized view index for efficiently discovering usable views for progressive queries," *Journal of Information Processing Systems*, vol. 9, no. 4, pp. 511–537, 2013.

[3] Y. Park, K. Whang, B. S. Lee, and W. Han, "Efficient evaluation of partial match queries for XML documents using information retrieval techniques," in *Proceedings of the Database Systems for Advanced Applications (DASFAA '05)*, pp. 95–112, April 2005.

[4] R. M. Hwang, S. K. Kim, S. An, and D. W. Park, "The architectural pattern of a highly extensible system for the asynchronous processing of a large amount of data," *Journal of Information Processing Systems*, vol. 9, no. 4, pp. 511–537, 2013.

[5] J. Heo, J. Cho, and K. Whang, "The hybrid-layer index: a synergic approach to answering Top-k queries in arbitrary subspaces," in *Proceedings of the 26th international Conference on Data Engineering*, pp. 445–448, March 2010.

[6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proceedings of the 19th International Conference on Data Engineering*, pp. 717–719, March 2003.

[7] A. Vlachou, C. Doulkeridis, and Y. Kotidis, "Angle-based space partitioning for efficient parallel skyline computation," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 227–238, June 2008.

[8] H. Köhler, J. Yang, and X. Zhou, "Efficient parallel skyline processing using hyperplane projections," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pp. 85–94, June 2011.

[9] Y. Ma, J. Rao, W. Hu et al., "An efficient index for massive IOT data in cloud environment," in *Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM '12)*, pp. 2129–2133, 2012.

[10] Z. Huang, Y. Xiang, D. Wang, and B. Zhang, "Efficient dynamic SKYCUBE computation in the internet of things," in *Proceedings of the International Conference on Computer and Communication Technologies in Agriculture Engineering (CCTAE '10)*, pp. 308–311, June 2010.

[11] M. A. Elkheir, M. Hayajneh, and N. A. Ali, "Data management for the internet of things: design primitives and solution," *Sensors*, vol. 13, no. 11, pp. 15582–15612, 2013.

[12] C. W. Tsai, C. F. Lai, M. C. Chiang, and L. T. Yang, "Data mining for internet of things: a survey ," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 77–97, 2014.

[13] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings of the 17th International Conference on Data Engineering*, pp. 421–430, April 2001.

[14] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets," in *Proceedings of the 31st International Conference on Very Large Data Bases*, pp. 229–240, September 2005.

[15] K. Hose, C. Lemke, and K.-U. Sattler, "Processing relaxed skylines in PDMS using distributed data summaries," in *Proceedings of the 2006 Conference on Information and Knowledge Management*, pp. 425–434, November 2006.

[16] S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu, "Efficient skyline query processing on peer-to-peer networks," in *Proceedings of the 2007 International Conference on Data Engineering*, pp. 1126–1135, April 2007.

[17] A. Cosgaya-Lozano, A. Rau-Chaplin, and N. Zeh, "Parallel computation of skyline queries," in *Proceedings of the 21st International Symposium on High Performance Computing Systems and Applications*, pp. 1–7, May 2007.

[18] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi, "Parallelizing skyline queries for scalable distribution," in *Proceedings of the 2006 Conference on Extending Database Technology*, pp. 112–130, 2006.

[19] Y. C. Chang, L. Bergman, V. Castelli, C. S. Li, M. L. Lo, and J. R. Smith, "The onion technique: indexing for linear optimization queries," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of data*, pp. 391–402, 2000.

[20] S. Y. Ihm, K. E. Lee, A. Nasridinov, J. S. Heo, and Y. H. Park, "Approximate convex skyline: a partitioned layer-based index for efficient processing top-k queries," *Knowledge-Based Systems*, vol. 61, pp. 13–28, 2014.

[21] S. Y. Ihm, A. Nasridinov, and Y. H. Park, "An efficient index building algorithm for selection of aggregator node in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2014, Article ID 520428, 8 pages, 2014.

[22] J. W. K. Gnanaraj, K. Ezra, and E. B. Rajsingh, "Smart card based time efficient authentication scheme for global grid computing," *Human-Centric Computing and Information Sciences*, vol. 3, no. 16, pp. 1–14, 2013.

[23] S. Hong and J. Chang, "A new *k*-NN query processing algorithm based on multicasting-based cell expansion in location-based services," *Journal of Convergence*, vol. 4, no. 4, pp. 1–6, 2013.

[24] H. I. Kim, Y. K. Kim, and J. W. Chang, "A grid-based cloaking area creation scheme for continuous LBS queries in distributed systems," *Journal of Convergence*, vol. 4, no. 1, pp. 23–30, 2013.