Adapting Skyline Computation to the MapReduce Framework: Algorithms and Experiments*

Boliang Zhang[1], Shuigeng Zhou[1], and Jihong Guan[2]

[1] School of Computer Science, and Shanghai Key Lab of Intelligent Information Processing, Fudan University, Shanghai, China
[2] Dept. of Computer Science & Technology, Tongji University, Shanghai, China
{boliangzhang,sgzhou}@fudan.edu.cn, jhguan@tongji.edu.cn

Abstract. This paper addresses the problem of skyline computation under the MapReduce framework. As a parallel programming model for data-intensive computing applications, MapReduce runs on a cluster of commercial PCs with the main idea of task decomposition and result reduction. Based on different data partitioning strategies, three MapReduce style skyline computation algorithms are developed: MapReduce based BNL (MR–BNL), MapReduce based SFS (MR–SFS) and MapReduce based Bitmap (MR–Bitmap). Extensive experiments are conducted to evaluate and compare the three algorithms under different settings of data distribution, dimensionality, buffer size and cluster size.

Keywords: Cloud computing; MapReduce; Skyline computation.

1 Introduction

Skyline computation, aiming at multi-objective decision originally, has a variety of applications in database area nowadays. Suppose you go to some seaside city for a holiday [1], and you need to find a hotel that is both cheap and close to the beach. Apparently, you can not have it in both ways. In fact, those hotels that are not worse than others in both ways are acceptable. We call the set of interesting hotels above the *skyline*, each of which is a *skyline point*. Data visualization is another important application of skyline. For instance, we can show the landscape outline of Manhattan by computing the set of buildings that are higher or closer to the river, which constitutes the skyline of Manhattan. This is just where the name "skyline" comes.

The skyline operator was first proposed by Börzsönyi et al. [1] in 2001. In the past years, approaches to skyline computation have been published extensively in major database conferences, including SIGMOD, VLDB and ICDE. Although

* This work was supported by National Natural Science Foundation of China under grants No. 60873040 and No. 60873070. Jihong Guan was also supported by the Shuguang Scholar Program of Shanghai Education Development Foundation under grant No. 09SG23.

J. Xu et al. (Eds.): DASFAA Workshops 2011, LNCS 6637, pp. 403–414, 2011.

skyline computation has been studied well under traditional definition, there are still some issues worthy of further investigation. To name a few, 1) *quality*. Usually, the number of skyline points grows exponentially with data dimensionality, and it also depends on data distribution. So how to retrieve a small number of high quality (or most representative) skyline points is absolutely not a trivial issue. 2) *Progressiveness and user preferences*. Many skyline applications require progressive response and to consider users' specific preferences while delivering the results. 3) *Efficiency*. Due to the dimension curse, skyline computation is very expensive for large-scale and high-dimensional datasets. Thus, how to speed up skyline computation is still a meaningful research direction.

This paper addresses the efficiency issue of skyline computation from parallel computation perspective. Concretely, we employ the MapReduce framework [2] to support skyline computation. MapReduce, as a parallel programming model developed by Google, runs on a large cluster of commercial PCs to process large datasets. The MapReduce program will carefully partition the input file, automatically schedule tasks over the cluster, deal with machine failures and manage communications between nodes. With its scalability, along with the reliability provided by GFS (Google File System) [3], we are able to process large scale of data well.

The combination of skyline computation and the MapReduce framework in this paper is mainly motivated by the two facts: 1) with the development of cloud computing, supporting data management and query processing in cloud platforms is emerging as a new trend of database research and commercialization, and 2) As a typical parallel programming model, MapReduce is receiving increasingly recognition in both academia and industry, and more and more applications are being developed under this framework. As most existing skyline computation algorithms are devised for centralized or distributed applications, and the MapReduce framework is a new parallel programming model with unique features, the combination can not be done well in a simple and straightforward way. Contributions of this paper include three aspects:

- Adapting skyline computation to the MapReduce framework for improving efficiency, to the best of our knowledge, is possibly the first of such work.
- Three MapReduce style algorithms for skyline computation are developed based on different data partitioning strategies.
- Extensive experiments are conducted to evaluate and compare the three algorithms under different settings of data distribution, dimensionality, buffer size and cluster size.

The rest of this paper is organized as follows. Section 2 describes the basic information of skyline computation and the MapReduce framework. Section 3 introduces three MapReduce style algorithms for skyline computation in cloud platforms. Section 4 presents the results of experimental evaluation. Section 5 reviews the related work. Section 6 draws conclusion and highlights future work.

2 Preliminaries

In this section, we present the definition and some basic properties of skyline, and a brief introduction to the MapReduce framework.

2.1 Skyline: Definition and Properties

Given a d-dimensional dataset $S = (S_1, S_2, \ldots, S_d)$, suppose that there exists an ordering relation on each dimension, denoted by $\preceq$ and $\prec$. For arbitrary values x and y in S_i, $x \preceq y$ means that x is not worse than y, and $x \prec y$ means that x is better than y.

Definition 1 (Dominate). *Given two data points in a d-dimensional dataset, say p and q, we say p dominates q iff for every* $i \in [1, d]$ *we have* $p_i \preceq q_i$ *and there exists at least one* j *such that* $p_j \prec q_j$.

Definition 2 (Skyline point). *Given a data set S, a point* $p \in S$, *if there exists no other point q such that q dominates p, we say p is a skyline point in S. All skyline points in S constitute the skyline of dataset S.*

From the definition above, we can easily infer some properties of skyline. If p dominates q in S, then for any scoring function $\varphi : S \to R$ that is monotonically increasing in all dimensions, we have $\varphi(p) > \varphi(q)$. Furthermore, for every φ, if $\varphi(p)$ reaches the maximum score, then p must be a skyline point. The opposite also holds.

2.2 The MapReduce Framework

In a MapReduce program, users specify a *map* function to generate a large amount of key/value pairs, and a *reduce* function to collect all intermediate values associated with the same intermediate key. Fig. 1 shows the workflow of a MapReduce program. Before the map phase starts, the input file is divided into logical splits for parallelization. Before the reduce phase, a shuffle procedure is carried out to release network pressure.

For instance, suppose that we want to count the occurrences of each word in a document. The *map* function simply outputs the word associated with 1, and the *reduce* function merges the same word and counts its occurrences.

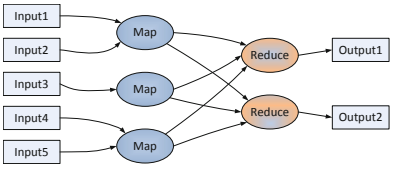

Fig. 1. The workflow of a MapReduce program

3 MapReduce-Based Skyline Computation Algorithms

Since MapReduce is a parallel programming model, the first consideration is how we partition the computation task. A straightforward way is to divide the original dataset into several subsets. Both NN [4] and D & C [1] algorithms use this idea. We call it horizontal partitioning. Another way is to partition data in a vertical fashion, i.e., each subset representing one single dimension of the dataset. Balke et al. [5] exploited sorted lists of all dimensions. Bitmap [6] utilizes this scheme to establish its bitmap structure.

Followed the two data partitioning strategies mentioned above, we design three algorithms based on three existing centralized approaches. They are MapReduce based BNL (MR-BNL), MapReduce based SFS (MR-SFS) and MapReduce based Bitmap (MR-Bitmap). All of them are comprised of two cascading MapReduce jobs. For convenience, we assume that in a d-dimensional dataset S, the size of S is N, and there are R distinct values per dimension, denoted as $[1, 2, \ldots, R]$.

3.1 MR-BNL

BNL [1] is an iterative algorithm that repeatedly reads a set of data records for processing. It keeps a window containing incomparable records in main memory to collect candidate skyline records. When a record p is read, it is compared with records in the window. According to the value of p, either of the following situations may happen. Record p is discarded or added into the window, or the points in the window dominated by p are deleted. With fixed data size N and dimension d, the runtime of BNL depends on the window size and the ordering degree of the original data. The window size influences the number of iterations, which is also a dominant factor since I/O cost far exceeds CPU cost.

Our MapReduce based BNL algorithm (MR-BNL) is a two-stage method. The first stage is to divide the whole data into small disjoint subsets. For each subset, we run a BNL procedure to compute the skyline. In the second stage, the local skylines are merged and filtered, thus the global skyline is obtained.

The NN algorithm [4] divides the global space into 2^d subspaces according to the first nearest neighbor. The D & C algorithm [1] recursively partitions the data space according to the median of a certain dimension. In MR-BNL, we combine these ideas together such that the input dataset is divided into 2^d subspaces based on a carefully chosen point. Considering load balance, the median of each dimension is chosen as the point of division. Thus, each dimension is divided into two parts: the higher part with larger values in the dimension and the lower part with smaller values. If we mark the higher as 1 and the lower as 0, then each subspace can be identified by a d-bit *flag*. In the merging stage, we exploit the flags to reduce unnecessary comparisons.

Fig. 2 shows a two-dimensional dataset. The lines $x = 0.5$ and $y = 0.5$ divide the space into four parts, whose flags are 00, 01, 10 and 11, respectively. The data records in subspace 00 dominate that in subspace 11, while subspace 01 is incomparable with subspace 10. There are total $C_4^2 = 6$ pairs of subspaces, with 16.67% incomparable pairs. This ratio grows with dimension d, denoted by

$\gamma(d)$. $\gamma(d)$ rises to 88.92% when d reaches 10, which is calculated by a simple program. Since bitwise operation costs only a little, the cost of comparisons in the merging stage can be approximately reduced by a factor of $\gamma(d)$.

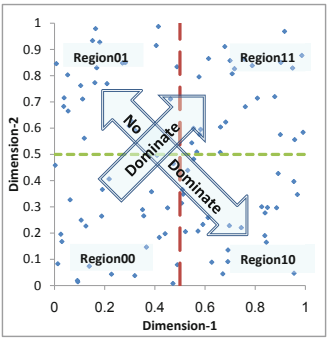

Fig. 2. Subspace and dominating relationship

Algorithm 1. MR-BNL

```
    INPUT: the original data set S
    OUTPUT: the skyline of data set S
 1: Division Job
 2:   Map Task
 3:   for each point P_i in dataset S
 4:     compute P_i's subspace flag F_i
 5:     output (F_i,P_i)
 6:   Reduce Task
 7:   for each subspace flag F_i
 8:     compute local Skyline SP_k using BNL
 9:     output (F_i, SP_k) in file t
10:
11: Merging Job
12:   Map Task
13:   for each point P_i in file t
14:     output (null, (F_i, P_i))
15:   Reduce Task
16:   compute global Skyline SP_k using BNL with pre-comparison
17:   output (SP_k, null)
```

The procedure of MR-BNL is outlined in Algorithm 1. In line 14, the key of the map output is null so that all records are gathered in one reduce call. The *pre-comparison* procedure does judgements in advance by using subspace flags.

The bottleneck lies in the second stage where the merging is done by one single node. The performance may be significantly degraded in the case that the size of local skyline produced by the first stage is huge.

3.2 MR-SFS

One shortcoming of BNL is that a record, once inserted into the window, might be discarded later. Therefore, this non-skyline record occupies window space, which may increase the iteration times. Sort-Filter-Skyline (SFS) [7] was devised to overcome the drawback. The input data is sorted at first in a topological order compatible with the skyline criterion. Thus, once a record r is inserted into the window, no record following it will dominate r and r is a skyline point. Therefore, at the end of each iteration, we can output all records inside. The iteration times of SFS is optimal, i.e., $\lceil \frac{N}{W} \rceil$ where N represents the number of data records and W is the window size in number of records.

We implement the MapReduce based SFS algorithm by doing some modification over MR-BNL: presorting. However, MapReduce can only sort by the keys of intermediate key/value pairs. In order to achieve secondary sort for the values set in the reduce call, we exploit the grouping and comparing features of MapReduce and the APIs provided by the Hadoop platform [8]. First, we assign key/value pairs of the same key to one reducer. Second, we put together the values of the same key. Third, user defined *comparators* are used to determine the order of records in the reduce phase.

To implement the three steps above, for any record r, we embed its key and value together, forming a new *Composite*. Then, we create two *comparators* and one *partitioner* for the *Composite*. In step 1, the *partitioner* partitions *Composite* in accordance with its key. In step 2, we use a *KeyComparator* to compare only the key of *Composite*. In step 3, we use a *KeyValueComparator* to compare not only the key but also the value of *Composite*, and yet, the key has the priority. Algorithm MR-SFS can be easily transformed from MR-BNL according to the above methods. Due to space limit, here we omit the pseudo-codes of MR-SFS.

3.3 MR-Bitmap

In Bitmap [6], every point is mapped into a bit vector, and the whole dataset forms the bitmap structure. The bitmap structure reflects the order of data records in all dimensions, ignoring their magnitudes. Every comparison in Bitmap is a lightweight bitwise operation. However, in order to examine a record, we need to compare it with all the other records. Bitmap supports progressive skyline computation since a point can be returned to the user immediately once it is identified as a member of the skyline. But the spatial cost of Bitmap is quite large. In particular, let M_i be the number of distinct values in dimension i, the size of Bitmap structure $S_b{=}N \times \sum_{i=1}^{d} M_i$ (bit). In the extreme case, for each dimension i, if $M_i{=}N$ (N is size of data set), then the space cost will be dN^2. If the bitmap structure can not be kept in main memory, the performance will be terribly degraded due to frequent I/O accesses.

The basic idea of MR-Bitmap keeps in line with that of Bitmap. In the first job, we build the bitmap structure. In the second job, we examine each point

Algorithm 2. MR-Bitmap

```
    INPUT: the original data set S
    OUTPUT: the skyline of data set S
 1: Building Job
 2:   Map Task
 3:   for each point P_i in data set S
 4:     for each attribute A_j in P_i
 5:     output (j, (A_j, P_i's byte offsets))
 6:   Reduce Task
 7:   for each dimension k
 8:     generate sorted attribute list L_k in descending order
 9:     for each distinct value v in L_k
10:       generate bit-slice and write to HDFS
11:
12: Examining Job
13:   Map Task
14:   for each point P_i in data set S
15:     assign P_i to a reducer R_i
16:   Reduce Task
17:   for each reducer R_i
18:     load bitmap b into memory as much as possible
19:     for each point P_i in R_i
20:       check P_i using bitmap b
```

based on the bitmap structure. Algorithm 2 outlines the procedure of the MR-Bitmap algorithm.

Line $8 \sim 10$ is the procedure of generating bitmap, and each reducer task is responsible for one dimension. Each distinct value in a dimension will produce a bit-slice. Line 15 determines how many reducers will be launched for the examining process. Let R_n be this number. Each reducer needs to read bitmap into memory. If R_n is too large, loading cost becomes dominant, which may degrade efficiency. On the contrary, if R_n is too small, the degree of parallelism is too low. In practice, R_n is set to the number of cluster nodes. In line 20, we use the same procedure as in [6] to judge whether point P_i is a skyline point.

If the bitmap structure cannot fit into main memory, we adopt cache-like strategy. When accessing some bit-slices, we check whether it is hit in memory, if not, we read it from disk. So it is natural to store the bitmap in column manner.

4 Performance Evaluation

In this section, we will evaluate and compare the proposed algorithms by a series of experiments with different data distributions, dimensionalities, buffer sizes and cluster sizes.

4.1 Experimental Setting

We set up a cluster of 8 commodity PCs, each of which has an Intel Duo Core 3.00GHz CPU, 4GB memory and Windows XP OS. We use Hadoop 0.20.2, and compile the source codes under JDK 1.6 in Eclipse 3.3.2.

We use similar datasets as in [1], each of which contains 100000 data records. The size of the record is fixed to 100 bytes. For each dimension, we generate integers from 1 to 100. Three data distributions are considered as follows:

- **Independent:** all dimensions follow uniform distribution, independently and identically distributed.
- **Correlated:** points perform well in one dimension are also good in the other dimensions.
- **Anti-related:** points perform well in one dimension are bad in one or all of the other dimensions.

Evaluation Metrics. We evaluate the algorithms by skyline ratio, runtime and I/O cost in different situations, including data distributions, dimensionality, distinction, buffer size and cluster size. Followings are the details:

- Skyline ratio: the number of skyline points over the total number of points.
- Distinction: the number of distinct values per dimension.
- Buffer size: a logical concept extended from the window in BNL, which means how many records can be kept in main memory. For MR-Bitmap, the bitmap structure can be kept in main memory in general.
- Cluster size: the number of nodes in a cluster, including one master node.

4.2 Experimental Results

Skyline Ratio. First, we vary the dimensionality d from 2 to 10 for three data distributions, and the results are presented in Fig. 3. We can see that the skyline ratio of Correlated grows negligible with d. On the contrary, the skyline ratio of Anti-related grows exponentially with d, reaching 58% when $d = 10$. As for Independent, its curve lies between the other two distributions. We then consider the relation between skyline ratio and data distinction. We vary the number of distinct values per dimension R from 20 to 20000 in a 5-dimension independent dataset. The result is shown in Fig. 4. As R increases, skyline ratio also increases.

Runtime and I/O cost. Here we measure the runtimes of three algorithms for three distributions with d=2, 5, 8 and 10, and the I/O cost of anti-related. Here I/O cost includes read/write bytes upon both local disk and HDFS. The buffer size is set to 5 MB and the cluster size is 4. Fig. 5 shows I/O cost of Anti, while Fig. 6, 7 and 8 show the runtimes of different distributions.

When the bitmap structure can fit into memory, the runtime of MR-Bitmap is linear with dimensionality and irrelevant to data distribution. The runtime of MR-Bitmap only depends on the size of bitmap and dimensionality. With respect to I/O cost, MR-Bitmap outnumbers the others a lot since it needs to write and

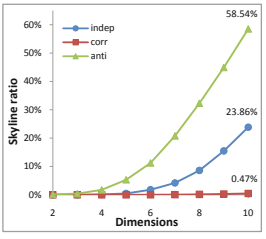

Fig. 3. Skyline ratio vs. dimensionality

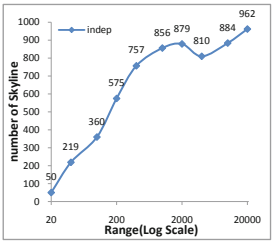

Fig. 4. Skyline ratio vs. data distinction

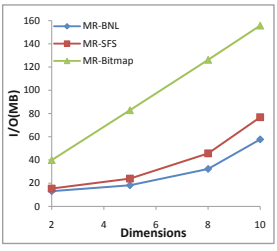

Fig. 5. I/O cost of Anti

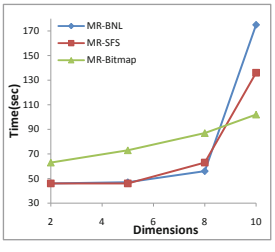

Fig. 6. Runtime of Indep

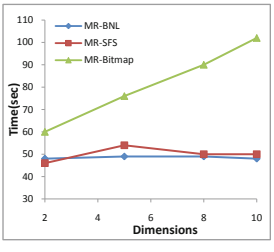

Fig. 7. Runtime of Corr

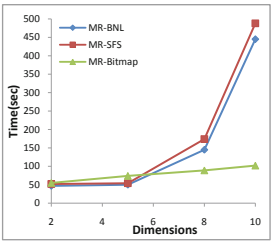

Fig. 8. Runtime of Anti

read bitmap to and from HDFS. The curves of MR-BNL and MR-SFS are very close. Unfortunately, we do not see the superiority of SFS in performance. On one hand, the sorting strategy of MR-SFS causes extra I/Os. As we can see in Fig. 5, I/O cost of MR-SFS outnumbers that of MR-BNL. On the other hand, the buffer size is half of the total number of records, so the number of iterations is small. Furthermore, Both MR-BNL and MR-SFS are sensitive to data distribution. MR-Bitmap performs better when dimensionality is large, because MR-BNL and MR-SFS concentrate on one node in the second phase, degenerating to centralized ones.

Effect of Buffer Size. We change the buffer size from 0.1 MB to 10 MB in a 8-dimensional anti-related dataset under a cluster of 4 nodes. The result is illustrated in Fig. 9. The runtimes of MR-BNL and MR-SFS drop drastically when the available buffer size increases from 0.1 MB (1000 records at a time). When the buffer size reaches 3 MB and larger, the runtime keeps almost constant since the number of iterations changes little. For MR-Bitmap, we conduct only two experiments, corresponding to 5% and 10% of bitmap not being loaded into main memory respectively. The results are disappointedly 958s and 1567s. The key point lies in that examining records may access HDFS several times randomly when the bitmap structure can not be totally loaded into main memory.

Effect of Distinction. For an 8-dimension anti-related dataset with a cluster of 4 nodes, we vary the number of distinct values per dimension R from 10 to

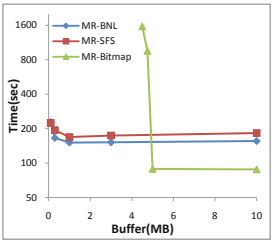

Fig. 9. Runtime vs. buffer size

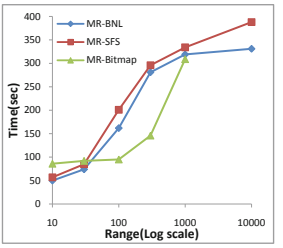

Fig. 10. Runtime vs. distinction

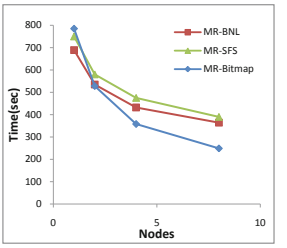

Fig. 11. Runtime vs. cluster size

10000. We do not test MR-Bitmap when R=10000 since the bitmap is too large. The results are presented in Fig. 10, MR-Bitmap outperforms MR-BNL and MR-SFS when R increases from 30 to 1000. When R exceeds 1000, the bitmap structure is large and MR-Bitmap has the largest I/O cost.

Effect of Cluster Size. At last, we change the nodes in the cluster. Theoretically, the runtime of a purely parallel algorithm is in inverse proportion to the cluster size. In practice, the performance will be negatively impacted by the cost of scheduling among nodes. In this experiment, we use a 10-dimension anti-related dataset, with values changing from 1 to 1000 in each dimension. We expand the size of the cluster from 1 to 8, Fig. 11 shows the results. As expected, 1) the increase of cluster size improves stably but not linearly the computation efficiency, 2) MR-Bitmap outperforms MR-BNL and MR-SFS in efficiency because the former is concurrent in both stages.

5 Related Work

Here we review the related work from two aspects: skyline computation and MapReduce-based data management and query processing.

5.1 Skyline Computation

Up to now, a number of algorithms for skyline computation have been developed, such as block-nested-loops (BNL) [1], sort-filter-skyline (SFS) [7], Bitmap [6], nearest neighbor (NN) [4] and branch and bound skyline (BBS) [9]. It was admitted that BBS performs best regarding to runtime and space cost when dimensionality is not too large. In addition, there are some works that either expand the notion of skyline or solve the problem in different application scenarios.

Chan et al. [10] proposed k-dominant skyline and developed efficient ways to compute it in high-dimensional space. Lin et al. [11] proposed n-of-N skyline query to support online query on data streams, i.e., to find the skyline of the set composed of the most recent n elements. In the cases where the datasets are very large and stored distributedly, it is impossible to handle them in a centralized fashion. Balke et al. [5] first mined skyline in a distributed environment by

partitioning the data vertically. Wang et al. [12] and Deng et al. [13] proposed skyline computation under P2P and road networks, respectively. Recently, Zhu et al. [14] presented an efficient skyline computation algorithm with low network bandwidth consumption in general distributed environments.

5.2 Data Management and Query Processing under the MapReduce Framework

Existing MapReduce-like systems, as efficient implementations towards specific problems, mainly focus on mass data analysis and processing. The data of such applications often has weak relationships and regular structures, which is beneficial to being processed by highly parallel subtasks. Furthermore, the processing or analysis tasks are not complicated in general. In this aspect, Google's Sawzall [15] and Yahoo!'s Pig [16] are two typical systems. And yet, for complex query processing under MapReduce framework, such as skyline and kNN, few works have been reported.

Some recent works are also worthy of being mentioned. For sharing input among a batch of queries, Nykiel et al. [17] proposed the MRshare framework, which merges several MapReduce jobs into a single one through dynamic programming to optimize overall efficiency. Unsatisfied with HadoopDB's modification over Hadoop, Dittrich et al. [18] proposed the Hadoop++, which merely overrides a few user defined functions (UDFs) to obtain advantages brought about by index and join techniques in traditional DBMSs. Considering that MapReduce paradigm does not provide adequate support for iterative programs, Bu et al. [19] put forward HaLoop, a modified version of Hadoop, which serves applications with iterative approaches, such as k-means, pagerank and recursive queries.

6 Conclusion

This paper addresses skyline computation in cloud computing environments. Based on the MapReduce framework, we have presented three MapReduce style algorithms for skyline computation: MR-BNL, MR-SFS and MR-Bitmap. To evaluate these algorithms, we have conducted a series of experiments under different experimental settings. Experimental results show that MR-BNL and MR-SFS are good in most cases but still suffer from dimensional curse in parallel environments. MR-Bitmap performs well regardless of data distributions, especially when the bitmap can fit into the memory of a single node.

It is worthy of being mentioned that this work can be expanded in a number of directions. First, from the perspective of parallel computing, how to extract as many independent subtasks as possible from the problem is crucial and needs to be further investigated. This is also true in the problem of this paper. Second, the progressiveness of Skyline computation should be thought highly of, since users are very likely to make a decision after receiving only a few recommendations rather than waiting for the whole results. To shorten response time is not an easy

task because the fixed start-up time of a Hadoop job is around 15 seconds. Last but not the least, to design efficient algorithms in cloud computing environments by using indices and exploiting features of splitting and sorting, are all promising research topics.

References

1. Börzsönyi, S., Kossmann, D., Stocker, K.: The Skyline operator. In: Proceedings of ICDE, pp. 421–430 (2001)
2. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large cluster. In: Proceedings of OSDI, pp. 137–150 (2004)
3. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google file system. In: Proceedings of SOSP, pp. 29–43 (2003)
4. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: An online algorithm for Skyline queries. In: Proceedings of VLDB, pp. 275–286 (2002)
5. Balke, W. T., Güntzer, U., Zheng, J.: Efficient Distributed Skylining for Web Information Systems. In: Hwang, J., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 256–273. Springer, Heidelberg (2004)
6. Tan, K.L., Eng, P.K., Ooi, B.C.: Efficient progressive Skyline computation. In: Proceedings of VLDB, pp. 301–310 (2001)
7. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: Proceedings of ICDE, pp. 717–719 (2003)
8. White, T.: Hadoop: The Definitive Guild. O'Reilly, Sebastopol (2009)
9. Papadias, D., Tao, Y., Fu, G., et al.: Progressive Skyline Computation in Database Systems. ACM TODS 30(1), 41–82 (2005)
10. Chan, C., Jagadish, H.V., Tan, K.L., et al.: Finding k-dominant Skylines in high dimensional space. In: Proceedings of SIGMOD, pp. 503–514 (2006)
11. Lin, X., Yuan, Y., Wang, W., et al.: Stabbing the sky: Efficient Skyline computation over sliding windows. In: Proceedings of ICDE, pp. 502–513 (2005)
12. Wang, S., Ooi, B.C., Tung, A., et al.: Efficient Skyline query processing on peer-to-peer networks. In: Proceedings of ICDE, pp. 1126–1135 (2007)
13. Deng, K., Zhou, X., Shen, H.: Multi-source Skyline query processing in road networks. In: Proceedings of ICDE, pp. 796–805 (2007)
14. Zhu, L., Tao, Y., Zhou, S.: Distributed Skyline Retrieval with Low Bandwidth Consumption. IEEE Transactions on Data and Knowledge Engineering 21(3), 321–334 (2009)
15. Pike, R., Dorward, S., Griesemer, R., et al.: Interpreting the data: Parallel analysis with Sawzall. Journal of Scientific Programming 13(4), 277–298 (2005)
16. Olston, C., Reed, B., Srivastava, U., et al.: Pig latin: a not-so-foreign language for data processing. In: Proceedings of SIGMOD, pp. 1099–1110 (2008)
17. Nykiel, T., Potamias, M., Mishra, C., et al.: MRShare: Sharing Across Multiple Queries in MapReduce. In: Proceedings of VLDB, vol. 3(1), pp. 494–505 (2010)
18. Dittrich, J., Quiane-Ruiz, J.-A., Jindal, A., et al.: Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). In: Proceedings of VLDB, vol. 3(1), pp. 518–529 (2010)
19. Bu, Y., Howe, B., Balazinska, M.: HaLoop: Efficient Iterative Data Processing on Large Clusters. In: Proceedings of VLDB, pp. 285–296 (2010)