IJCSI International Journal of Computer Science Issues, Volume 13, Issue 5, September 2016
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org https://doi.org/10.20943/01201605.153158

Enhanced Parallel Skyline on Multi-core Architecture with Low Memory Space Cost

Jean Pepe M. Buanga[1], Simon Ntumba Badibanga[2], Richard Kabamba Ilunga[3]

[1] Yanshan University
Qinhuangdao, China
University of Kinshasa
Kinshasa, DRC

[2] University of Kinshasa
Kinshasa, DRC

[3] University of Kinshasa
Kinshasa, DRC

Abstract

Skyline is an important proposal for expressing user preferences. Based on multi-core computation, we propose a new algorithm that improves parallel skyline query algorithm (PSQ), by reducing the memory space cost, and efficiently increasing the speed-up of computation.

Keywords: *skyline query; multi-core processors; parallel computing*

1. Introduction

Skyline query is an efficient data analysis tool for helping users make intelligent decisions over complex data in different and often conflicting criteria. The skyline query is suitable for multi-core architecture since a large number of comparisons between data points can be performed independently. Multi-core systems run multiple tasks at the same time. Each task will be executed by a separate core in parallel, thus, boosting the performance [9].

So far, several algorithms for parallel skyline query have been developed for example; Parallel Skyline (PSkyline) [8] and Parallel Skyline Query algorithm (PSQ) [11].

As an overview, this paper proposes an Enhanced Parallel Skyline Query algorithm using low memory space, called E-PSQ, which is applied to shared memory architecture in order to efficiently speed up the computation. The remainder of this paper is organized as follows: Section 2 discusses the previous works that are related to ours; Section 3 analyzes the problem of parallel skyline algorithm (PSQ), as well as gives its corresponding solution and proposes a new algorithm that enhances PSQ; Section 4 shows experimental results, and Section 5 draws conclusions.

2. Related works

Sungwoo Park et al. [3] proposed a Parallel Skyline algorithm (PSkyline), based on the divide-and-conquer strategy [2]. It uses no index structures and divides a dataset linearly into smaller blocks of the same size, and computes local skyline points of each block by using a sequential skyline query algorithm, then merges local skyline points sequentially. PSkyline achieved a speedup approximately proportional to the number of cores. Compared to BBS [4], PSQ ran faster than sequential BBS and parallel BBS.

Meng-Zong Liou et al. [4] proposed a Parallel Skyline Query algorithm for multi-core architecture (**PSQ)**, which have been applied to shared memory architecture. It sorts the given dataset in descending order for each coordinate component separately, and finds out a terminating point [6] for eliminating redundant computations of skyline query[7] and [10]. The performance of PSQ algorithm has been compared to the PSkyline algorithm with eight cores by considering runtime and speed-up factors. The experimental results showed that the PSQ algorithm runs faster than the PSkyline algorithm when the dimensionality of the dataset is high or the number of data points is large.

3. Algorithms

3.1. Existing Parallel Skyline Query Algorithm (PSQ) problem

In PSQ Algorithm, the main problem is that the whole Dataset is scanned in memory space; and this issue increases the

utilization cost of memory space. This problem is described in three parts of this algorithm, as follows:

First, in order to sort each attribute separately, the whole Dataset is scanned in memory space.

parallel for *i* := 1 **to** d **do**

$S_i \leftarrow$ Sort ***D*** in non-ascending order according to the i-th dimension

end parallel

Second, the test $if\,(rank_i(p) \prec rank_i(t))$ concerns the comparison of all the tuples *p* except the terminating tuple *t*, belonging to Dataset ***D***, where $rank_i(p)$ denotes the rank of tuple p on *i*-th attribute, $rank_i(t)$ denotes the rank of terminating tuple on *i*-th attribute. Indeed, not all tuples are concerned by this test. The tuples concerned by this test are only the ones that have rank not greater than rank of terminating tuple. Given number of tuples in each attribute (*n*), d-Dataset(***D***), number of cores processor (*p*). If each core processor manipulates one attribute, then, the number of computation per core processor is *n* comparisons, *n* communications with local memory space to read all tuples. For all cores processors, we have $n \times p$ memory cases. Its complexity space is $O(n \times p)$. For example, n = 1,000,000, with eight cores and d = 6. We need very large physical memory space to compute skyline tuples with these given data, but, the suitable memory is very expensive. This algorithm could encounter bottleneck because of memory space, if there is not enough memory space.

Third, with whole Dataset in memory space, ***CheckSkyline*** procedure checks the dominance relationship between skyline tuples candidates so that it outputs the final skyline tuples progressively.

In Conclusion, this algorithm uses a high memory cost that could lead to bottleneck of system because of lack of memory space.

3.2. The proposed solution

In this work, we focused on second part of PSQ to reduce memory space. At this step, all points are not concerned by this test. Knowing terminating point and its rank, our principle is to create a small structure data as new dataset that will contain only interested points or candidate's skyline from each attribute separately, and then, eliminate all duplicates tuples. This structure will be used in the rest of algorithm, and then, the memory space cost will be reduced. This new dataset is denoted as ***ND*** (Algorithm 1).

Algorithm 1. Created small structure data as new dataset

```
1   parallel for i := 1 to d do
2     Determine ranktt
3   end parallel
4   k = 0
5   ND[0] = tt
6   parallel for i := 1 to d do
7     for j := 1 to ranktt
8       k = k + 1
9         ND_k = SD_ij      // SD_ij , Sorted Dataset
10  end parallel
11  for i := 1 to k do
12    remove all duplicate tuples from ND
13  end for
```

We illustrate our proposed solution with a concrete example. Consider a team of baseball players from Chicago White Sox in American League described in Table 1. There are 20 baseball players. Each player's performance is recorded by four attributes; these attributes are average (G), on base percentage (OBP), slugging percentage (SLG), At Bats (AVG). These attributes can be used to evaluate the performance of a baseball player, and interpreted as a 4-Dimensional point.

Table 1: Example of baseball Players of Chicago White Sox

No	Players	AVG	OBP	SLG	G
1.	Bill	0.828	0.34	0.488	73
2.	Bob	0.701	0.33	0.467	69
3.	Edey	0.662	0.3	0.3	65
4.	Ken	0.69	0.344	0.356	71
5.	Eve	0.301	0.485	0.564	80
6.	Fiston	0.243	0.451	0.587	75
7.	Gregoire	0.223	0.415	0.521	145
8.	Hercule	0.265	0.485	0.487	50
9.	Ignacio	0.285	0.432	0.654	60
10.	John	0.276	0.428	0.432	65
11.	Anne	0.297	0.471	0.614	75
12.	Bin	0.314	0.453	0.578	108
13.	Chris	0.309	0.447	0.556	112
14.	Dom	0.256	0.46	0.533	132
15.	Elie	0.303	0.487	0.566	82

16.	Fabrice	0.245	0.453	0.589	77
17.	Georges	0.225	0.417	0.523	147
18.	Harris	0.267	0.487	0.489	52
19.	Isaac	0.287	0.434	0.656	61
20.	Jason	0.278	0.43	0.434	67

After computing our module, the new Dataset has 16 tuples, less than the first Dataset with 20 tuples (Fig.1). For a large amount of dataset, the number of tuples will be more reduced in order to keep only the interested tuples, which will be used during the running of an algorithm. Therefore, we gain in the use of low memory space, and communication cost between core processor and memory space.

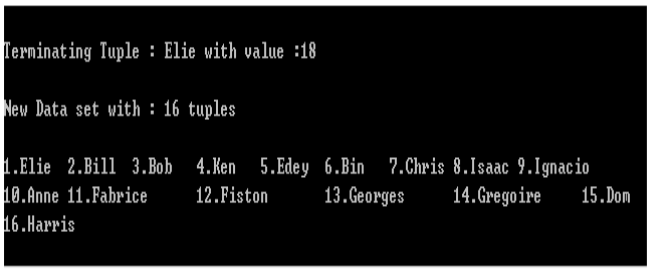

Fig.1. Execution result of our proposed solution

3.3. Proposed Algorithm: Enhanced parallel skyline query (E-PSQ)

Table 2: Frequently Used Symbols

Symbol	Meaning
D	Dataset
SD_i	i-th Sorted Dataset per attribute
ND	New Dataset from dataset ***D***
d	Dimension of dataset
q,t,p	Tuple
Tt	Terminating tuple
Ranktt	Rank of Terminating tuple
K	New size of dataset
SA_r	r-th Subset of tuples per attribute

Algorithm 2. Enhanced Parallel skyline query algorithm (E-PSQ)

Input: A list of tuples ***D***
Output: The skyline tuples of ***D***

1 **parallel for** *i* := 1 **to** d **do**
2 $S_i \leftarrow$ Sort ***D*** in ascending or descending order according to the i^{th} dimension
3 **end parallel**
4 $tt \leftarrow$ Determine a terminating tuple
5 **parallel for** *i* := 1 **to** d **do**
6 Determine *ranktt*
7 **end** parallel
8 *k* = 0
9 ***ND***[0] = *tt*
10 **parallel for** *i* := 1 **to** d **do**
11 **for** *j* := 1 **to** *ranktt*
12 *k* = *k* + 1
13 $ND_k = S_{ij}$
14 **end for**
15 **end parallel**
16 **for** *i* := 1 **to** *k* **do**
17 remove all duplicate tuples
18 **end for**
19 **parallel for each** $t \in ND$ **do**
20 **for** *i* := 1 **to** d **do**
21 **if** $(rank_i(t) \prec = rank_i(tt))$
22 find r with $rank_r(t) = \min\{rank_j(t) / 1 \le j \le d\}$
23 insert *t* into SA_r
24 **break**
25 **end for**
26 **end parallel**
27 **for** *i* := 1 **to** d **do**
28 **parallel for each** $p \in SA_i$ do
29 CheckSkyline(*i*, *t*)
30 **end parallel**
31 **end for**

Algorithm 3. Checking final skyline tuples

CheckSkyline(*i*,*t*)
1 **for each** $q \in SA_i$ with $rank_i(q) \prec rank(t)$ do
2 **if**(*q* is not marked as a non-skyline point)
3 **if**(*q* dominates *t*)
4 *t* is marked as non-skyline point
5 **Break**
end for
6 **if**(*t* is not marked as a non-skyline point)
7 *t* is marked as a skyline point
8 Output *t*
9

4. Experimental Results

4.1. Experimental Environment

We measure the performance of the proposed parallel algorithm called *E-PSQ* on a machine running OpenMP under C++ Visual Studio 2010 on Windows 7. The machine has AMD A8-5545M APU with Radeon(tm) HD Graphics (4 CPUs), ~1.7GHz, CPU-Z version 1.74.0.x64, Memory: 4096MB RAM.

To evaluate the proposed parallel indexing schemes, we used Matlab version 2014, and generated Hypercube data graphics.

As for the performance metrics, we measure execution time and speedup by varying successively dimension of Dataset (*d*), size of Dataset (*n*) and number of cores (*nthr*).

4.2. Results of experimentation

Related to our experiment, we ran our proposed program and PSQ on real dataset, Baseball players, and then, we compared their results. For each experiment, the test was ten at a time, and its discussion was based on average value.

A. Effect of varying dimensions

In this experiment, we test the effect of algorithms on varying dimensions with fixed-size of dataset (n = 100) and number of cores (c = 4) in order to output the Execution Time.

On hypercube model (Fig. 2), the convergence of the two lines represents the difference of Execution time between computation with PSQ and E-PSQ, in case of dimensionality; this difference is denoted as Δt ($\Delta t = t_{psq} - t_{e-psq}$, where t_{psq} is execution time with PSQ algorithm, and t_{e-psq} is execution time with E-PSQ algorithm).

From this hypercube, we describe three cases, as follows:

1. If $\Delta t \prec 0$, it means that the computation with E-PSQ has a higher Execution time than PSQ. In this case, it's not advantageous to compute with E-PSQ.
2. If $\Delta t = 0$, it means that both the computations with E-PSQ and PSQ have same Execution time.
3. If $\Delta t \succ 0$, it means that the computation with E-PSQ has lesser Execution time than with PSQ. In this case, it's better to compute with E-PSQ; and then, our algorithm run very fast.

By looking back to Fig. 2, Δt decreases slightly. We observe that for $d \prec 14$, Δt is larger than with $d \succ 14$; Hence, it's better to compute with E-PSQ for $d \prec 14$.

Shows how E-PSQ runs faster than PSQ.

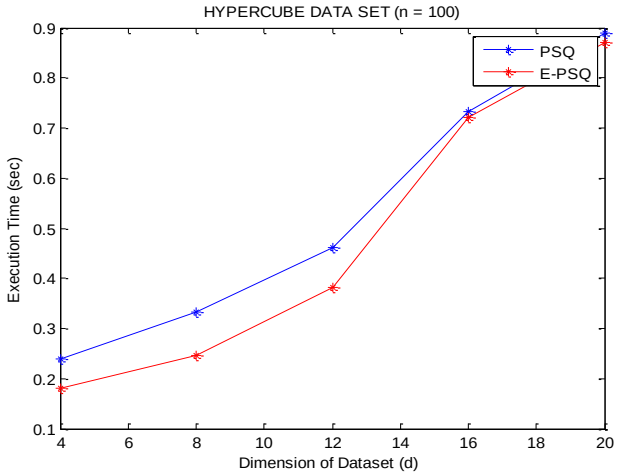

Fig.2. Varying Dimensions

B. Effect of varying size

In this experiment, we test the effect of algorithm on varying number of tuples with fixed dimension (d) and number of cores (c = 4) in order to output the Execution Time.

On Fig. 3, we observe two cases:

1. For $n \prec 60$, the Execution time is very small, less than 0.15 sec., for both computations with PSQ and E-PSQ; but, E-PSQ has execution time less than PSQ, although their difference time is small.

2. For $n \succ 70$, the Execution time increases highly, for both computations with PSQ and E-PSQ; but, this time with E-PSQ, is less than with PSQ. This means that E-PSQ is better than PSQ, and E-PSQ runs faster than PSQ.

The execution time of E-PSQ decreases slighter than that of PSQ when most of tuples of given Dataset are not dominated; in other case, the execution time decreases largely.

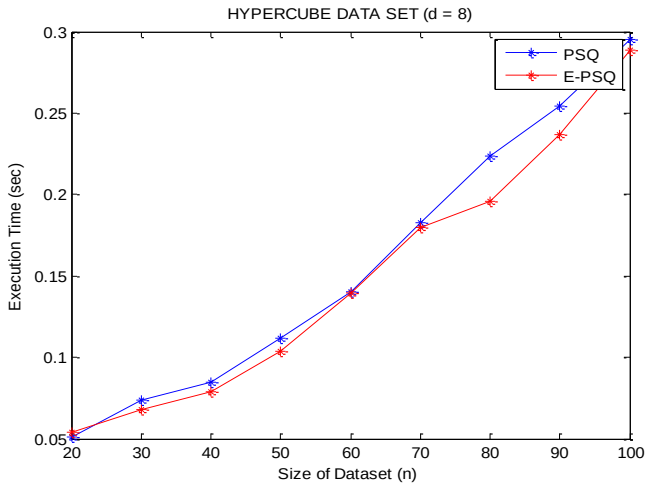

Fig.3. Varying size of Dataset

C. Effect of Number of cores

In this experiment, we test the effect of algorithm on varying number of cores with fixed dimension (d), fixed-size of Dataset (n) and number of cores (c = 4), in order to output the Execution Time.

On Fig.4, the divergence of the two lines represents the difference of speedup between computation with PSQ and E-PSQ; this difference is denoted as Δsp ($\Delta sp = \Delta sp_{e-psq} - \Delta sp_{psq}$, where Δsp_{psq} is execution time with PSQ algorithm, Δsp_{e-psq} is execution time with E-PSQ algorithm).

We observe that for $c \prec 3$, Δsp increases slightly; while for $c \succ 3$, Δsp increases largely. It means that a large number of cores lead to increases largely the speedup of our algorithm.

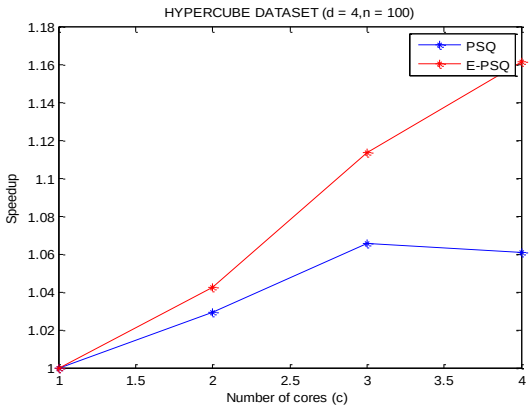

Fig.4. Varying of speedup related to number of cores

5. Conclusion

In this paper, based on multi-core system, we have presented Enhanced Parallel Skyline Queries (E-PSQ), a new approach to reduce the Dataset with only tuples that can be managed throughout running of PSQ algorithm, in order to retrieve skyline tuples. To this, we created a new Dataset from Dataset by keeping only interested tuples that can be analyzed to retrieve skyline set. This method has some advantages for instance, to get a large memory space for managing big data without increasing physical memory, to reduce the communication cost between core processors and memory space and to speed up the computation. As shown by our experimental evaluation, Enhanced PSQ was very efficient and ran faster than PSQ, with a low memory space cost.

6. References

[1] S. Borzsonyi et al., "The skyline operator", *Proc. 17th International Conference on Data Engineering*, 2001, pp 421–430.

[2] J. Chomicki et al., "Skyline with Presorting", *Proc. 19th International Conference on Data Engineering*, 2003, pp 717–719.

[3] D. Kossmann et al., "Shooting stars in the sky: an online algorithm for skyline queries", *Proc.28th International Conference on Very Large Data Bases*, 2002, pp 275–286.

[4] D. Papadias et al., "Progressive skyline computation in database systems", *ACM Transactions on Database Systems*, vol. 30, Issue 1, 2005, pp 41–82.

[5] W.T. Balke et al., "Efficient Distributed Skylining for Web Information Systems", *Proc. 9th International Conference on Extending Database Technology*, 2004, pp 256–273.

[6] E.Lo et al., "Progressive skylining over web-accessible databases", *Data & Knowledge Engineering*, vol. 57, 2006, pp. 122–147.

[7] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases", *The International Journal on Very Large Data Bases*, vol. 16, no. 4, 2007, pp. 523-544.

[8] S. Park et al., "Parallel Skyline Computation on Multi-core Architectures", *IEEE International Conference on Data Engineering*, 2009, pp. 760 - 771.

[9] Fasiku A. I. et al., "Performance Evaluation of Multi-core Processors", *International Journal of Engineering and Technology*, vol. 4, No. 1, January, 2014.

[10] B. Babcock and C. Olston, "Distributed Top-k Monitoring", *Proc. ACM SIGMOD Int'l conference on Management of data*, 2003, pp. 28-39.

[11] M. Liou et al., "Parallel Skyline Queries on Multi-core Systems", *Proc. Int'l Conference on Parallel and Distributed Computing, Applications and Technologies*, 2013, pp. 287-292.